PhD-FSTM-2022-088
The Faculty of Sciences, Technology and Medicine

DISSERTATION

Defence held on 07/09/2022 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG EN INFORMATIQUE

by

Jaekwon LEE
Born on 23rd November 1986 in Goesan (South Korea)

WCET AND PRIORITY ASSIGNMENT ANALYSIS OF REAL-TIME SYSTEMS USING SEARCH AND MACHINE LEARNING

Dissertation defense committee

Dr. Lionel BRIAND, Dissertation Supervisor
Professor, University of Luxembourg, Luxembourg

Dr. Fabrizio PASTORE, Chairman
Professor, University of Luxembourg, Luxembourg

Dr. Seung Yeob SHIN, Vice Chairman
Research Scientist, University of Luxembourg, Luxembourg

Dr. Manuel ORIOL, Member
Professor, Schaffhausen Institute of Technology, Switzerland

Dr. Mehrdad SAADATMAND, Member
Senior Researcher, RISE Research Institutes of Sweden, Sweden

Acknowledgement

First and foremost, I would like to express my deepest gratitude to my supervisor, Prof. Lionel Briand, and co-advisors Prof. Shiva Nejati and Dr. Seung Yeob Shin. This paper would have never been accomplished without their continuous support and encouragement and dedicated feedback during my research. They also gave me the opportunity to join the SVV group at the University of Luxembourg and to research at the University of Ottawa. This opportunity broadened my perspective regarding my studies and personal life.

I would also like to extend my sincere appreciation to my industrial partner companies, LuxSpace and Blackberry, especially Yago Isasi Parache, Chris Hobbs, and Alexandre Koppel. With the valuable data and feedback, I could understand industrial requirements and successfully finish all the research.

Additionally, I would like to give special thanks to Prof. Woosung Jung and Prof. Dongsun Kim. They have laid the foundation for my PhD research. I am also grateful to Dr. Kisub Kim for reviewing my paper and for the valuable feedback and Dr. Maris Jukss for helping me adopt the new working environment. I would also thank Mr. Alvaro Mario Veizaga Campero and all my colleagues from the SVV group for being good friends during this PhD journey. Thanks to them, my PhD journey could be more enjoyable and delightful.

I sincerely appreciate all my PhD defense committee members, including Prof. Fabrizio Pastore, Prof. Manuel Oriol, and Prof. Mehrdad Saadatmand for their efforts to examine my dissertation as well as for their advice and questions. It was my great honor to have them on my defense committee.

Last but not least, I would like to share all the honor with my lovely wife, Juhee, for being all the journey together. Although she has been struggling with language barriers in this country and the uncertainty of living abroad, she has been supporting and advising me. I will never forget all your dedication. I finalize my acknowledgment by saying thanks to my parents and my dear brother.

Jaekwon Lee
University of Luxembourg
September 2022

Abstract

Real-time systems have become indispensable for human life as they are used in numerous industries, such as vehicles, medical devices, and satellite systems. These systems are very sensitive to violations of their time constraints (deadlines), which can have catastrophic consequences. To verify whether the systems meet their time constraints, engineers perform schedulability analysis from early stages and throughout development. However, there are challenges in obtaining precise results from schedulability analysis due to estimating the worst-case execution times (WCETs) and assigning optimal priorities to tasks.

Estimating WCET is an important activity at early design stages of real-time systems. Based on such WCET estimates, engineers make design and implementation decisions to ensure that task executions always complete before their specified deadlines. However, in practice, engineers often cannot provide a precise point of WCET estimates and they prefer to provide plausible WCET ranges.

Task priority assignment is an important decision, as it determines the order of task executions and it has a substantial impact on schedulability results. It thus requires finding optimal priority assignments so that tasks not only complete their execution but also maximize the safety margins from their deadlines. Optimal priority values increase the tolerance of real-time systems to unexpected overheads in task executions so that they can still meet their deadlines. However, it is a hard problem to find optimal priority assignments because their evaluation relies on uncertain WCET values and complex engineering constraints must be accounted for.

This dissertation proposes three approaches to estimate WCET and assign optimal priorities at design stages. Combining a genetic algorithm and logistic regression, we first suggest an automatic approach to infer safe WCET ranges with a probabilistic guarantee based on the worst-case scheduling scenarios. We then introduce an extended approach to account for weakly hard real-time systems with an industrial schedule simulator. We evaluate our approaches by applying them to industrial systems from different domains and several synthetic systems. The results suggest that they are possible to estimate probabilistic safe WCET ranges efficiently and accurately so the deadline constraints are likely to be satisfied with a high degree of confidence. Moreover, we propose an automated technique that aims to identify the best possible priority assignments in real-time systems. The approach deals with multiple objectives regarding safety margins and engineering constraints using a coevolutionary algorithm. Evaluation with synthetic and industrial systems shows that the approach significantly outperforms both a baseline approach and solutions defined by practitioners. All the solutions in this dissertation scale to complex industrial systems for offline analysis within an acceptable time, i.e., at most 27 hours.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Motivation

Real-time systems have become indispensable for human life as they are being used in numerous industries such as vehicles, medical devices, and satellite systems [119]. These systems generally interact with the environment around them and exhibit dynamic behaviors according to environmental changes. For example, a satellite system, which is required to keep its antenna in the direction of a ground station, controls actuators based on specific information like the direction of the Sun or the magnetic field of the Earth by using sensors. Such systems are developed into multiple tasks that run in parallel and are repeatedly activated by time-triggers or event-triggers [72]. Unlike applications on personal computers, these real-time systems need to satisfy not only their functionality but also time constraints (deadlines). If time constraints are violated, even when functional requirements are met, systems can cause catastrophic consequences for themselves and related services. In order to minimize the chance of accidents or maximize the quality of service, systems need to be analyzed with respect to the schedulability of their tasks.

Schedulability analysis is an important mechanism for verifying time constraints in real-time systems during development. It verifies whether all tasks in a real-time system complete their execution before their deadlines for worst-case scenarios. To accurately test schedulability, many engineers run their systems with test cases that are expected to execute worst-case scenarios [3]. Such testing can be performed after completing task implementations. However, developing real-time systems preferably requires determining their schedulability earlier, especially when engineers develop software and hardware in parallel. To deal with schedulability analysis, Liu and Layland [118] developed a task model as a set of properties such as priority, deadline, inter-arrival time, and worst-case execution time (WCET). They then analyze the worst response time based on the model. However, such an approach can only handle periodic tasks, where activations (arrivals) regularly occur. There exist extensions [18, 22] with sporadic (irregularly activated) tasks and multi-frame tasks, respectively. Although these approaches improved the schedulability bound and relaxed assumptions on the task model, they are still hard to apply at early stages, due to the uncertainty

in determining the properties in the task model. Especially, practitioners face challenges when estimating WCET values and determining an effective priority assignment.

The problem of estimating WCET values is known as a hard problem in general [42]. Among the properties of the task model, periods and deadlines can be captured from the system requirements. However, estimating WCET is difficult and often practically impossible to accurately determine due to the following reasons: (1) hardware complexity (such as pipelines, caches, and multiple cores) (2) software architecture (such as structure and dependencies), and (3) finding worst-case inputs [103, 156]. Hence, engineers estimate plausible WCET ranges rather than exact values. In addition, at the early stage of development, it is more difficult as engineers do not even have (complete) implementations. However, estimating WCET early on is necessary because they provide targets driving design and implementation choices. For example, based on WCET estimates, engineers can make choices using either a relational database or an in-memory data storage. Such a decision needs to be made at early stages to find optimal configurations of hardware devices between the cost of the system development and its performance. It is more critical for projects that develop software and hardware components in parallel, which is common in the aerospace, automotive, and healthcare domains.

The most common approaches to estimating WCET are measurement-based [183, 51, 153]. They run a large number of executions on the targeted hardware or an accurate simulator based on worst-case inputs and measure maximum execution time. As they need executable source code, this is only applicable at later stages of implementation. Analytical approaches attempt to alleviate this issue [73, 169, 133, 86]. They measure WCET based on code execution path analysis and hardware specifications. Some other approaches also tried to measure WCET based on a timing model of machine instructions [84, 7, 33]. These analytical approaches enable WCET estimation at somewhat earlier stages of implementation, but still rely on source code implementations. Therefore, engineers still require a novel approach to estimate WCET values at early design stages.

Priority assignment is also an important activity since it determines the scheduling efficiency of real-time tasks. Most real-time operating systems (RTOS) use priority-based preemptive scheduling policies because they provide predictable schedulability [50]. In these policies, assigning optimal priorities can lead to cost-effective hardware and improved reliability in real-time systems [61]. However, finding the optimal priority assignment is known as a hard problem [20] as it entails schedulability testing for all combinations of task priority assignments.

An optimal priority assignment (OPA) approach attempted to solve the problem using an exhaustive search by pruning infeasible priority assignments [20]. Due to its efficiency, it is extended to allow offsets [19], aperiodic tasks having arbitrary deadlines [170], non-preemptive scheduling policy [78], and multi-core [59]. However, OPA only guarantees to find a feasible priority assignment. There are more works to improve optimality [56] and include different types of systems [62, 46, 58, 55]. However, the priority assignment problem is still challenging because of new, more complex hardware platforms and industrial requirements [54]. In addition to their inherent complexity, these approaches rely on schedulability tests, which is also a hard problem. Engineers, thus, want to assign priorities that lead to more robust systems when their tasks face additional execution times due to unexpected situations. For example, satellite systems can be exposed to uncertain environments, such as space radiations and solar winds, which are difficult to test on the ground. Moreover, engineers may need to account for additional constraints, such as assigning higher priorities to more critical tasks.

1.2 Contributions

In this dissertation, we propose the following approaches to address the challenges for WCET estimation and priority assignment. They mainly aim at early development stages, but can also be applied at later development stages.

1. *Estimating probabilistic safe WCET ranges at design stages*: Estimating WCET values as a range at early stages is the common practice of real-time system development. To specify more precise WCET ranges, we provide an automated technique to determine for what WCET values the system is likely to meet its deadlines, and hence operate safely with a probabilistic guarantee. The approach combines a search algorithm for generating worst-case scheduling scenarios with polynomial logistic regression for inferring probabilistic safe WCET ranges. We evaluated the approach by applying it to three industrial systems from different domains and several synthetic systems. The approach efficiently and accurately estimates probabilistic safe WCET ranges within which deadlines are likely to be satisfied with a high degree of confidence.

   This work has led to a research paper published in ACM Transactions on Software Engineering and Methodology (TOSEM) [113].

2. *Estimating probabilistic safe WCET ranges for weakly hard real-time systems at design stages*: In practice, real-time systems are commonly developed with complex scheduling policies, including task partitioning on multi-core platforms. Many of them can tolerate some extent of deadline misses, i.e., weakly hard constraints. To properly support these systems, we improve the above approach to infer safe WCET ranges with a probabilistic guarantee under complex systems. The approach applies a multi-objective search algorithm for generating worst-case scheduling scenarios by maximizing the violation of weakly hard constraints. To deal with industrial systems, the approach also accounts for context-switching times and an industrial scheduling simulator, such as adaptive partitioning scheduler (APS). We evaluated the approach by applying it to an industrial system in the satellite domain and several synthetic systems. The results indicate that the approach estimates safe WCET ranges with a probability of violating deadline constraints with a high degree of confidence in weakly hard real-time systems with APS functions. The execution time of the approach is acceptable in practice according to experiments with a large number of complex synthetic systems.

   This work will be submitted to a conference in 2022.

3. *Optimal priority assignment using a coevolution-based approach*: In practice, priority assignments result from an interactive process between the development and testing teams. Inspired by such interaction, we propose an automated technique that finds optimal priority assignments using a multi-objective and a competitive coevolutionary algorithm. The approach aims to identify the best possible priority assignments in real-time systems, accounting for multiple objectives regarding safety margins and engineering constraints. We evaluate this approach by applying it to six industrial systems from different domains and several synthetic systems. The results indicate that this approach significantly outperforms both a baseline approach and solutions defined by practitioners. The approach also scales to complex industrial systems as an offline analysis method that attempts to find near-optimal solutions within acceptable time, i.e., less than 16 hours.

This work has led to a research paper published in the journal of Empirical Software Engineering (EMSE) [110].

1.3 Outline

The rest of this dissertation is organized as follows. Chapter 2 describes the necessary background information, including a real-time task model and algorithms that we use in our approaches. Chapter 3 and 4 introduce the approaches to estimate probabilistic safe WCET ranges at early stages. Chapter 5 presents a coevolution approach for optimal priority assignment. Chapter 6 concludes this dissertation and discusses potential future works.

Chapter 2

Background

This chapter provides details about the common concepts and techniques that are used throughout this thesis. Section 2.1 defines the conceptual model for real-time systems. Section 2.2 explains the genetic algorithms and coevolutionary algorithms. Section 2.3 describes supervised machine learning and logistic regression.

2.1 Real-time systems

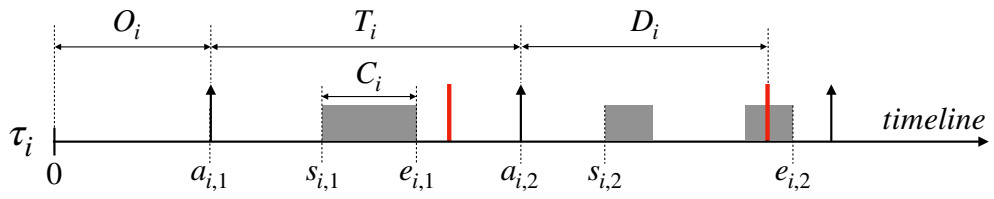
Real-time systems need to be developed to provide their own services that satisfy the required time constraints. Such systems achieve these goals by decomposing multiple tasks that affect each other. To properly guarantee whether the systems satisfy their time constraints, schedulability analysis is necessary. In this section, we introduce a real-time task model and how real-time tasks are scheduled based on the model.

2.1.1 Task model

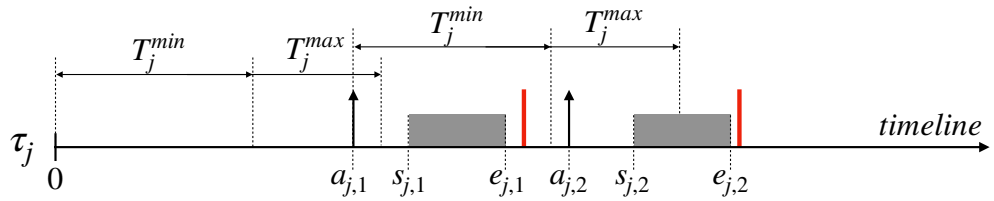
We define the real-time task model following concepts commonly used in studies in the literature [20, 50, 177]. A real-time system consists of a set of n tasks. Each task, denoted by τ_i, is a unit of work that is activated repeatedly and needs to complete its execution before its deadline. A task τ_i can be identified as *periodic* or *aperiodic* depending on the frequency of its activation.

Figure 2.1 describes the temporal parameters for each (a) periodic task τ_i and (b) aperiodic task τ_j, where $i, j \in [1, n]$, as well as their behaviors. Periodic tasks are typically activated by a time-based trigger, which occurs at regular intervals. To determine the time behavior of a periodic task τ_i, τ_i is identified following temporal parameters:

- Offset (O_i): is the time for the first activation (arrival) of periodic task τ_i. In the Figure 2.1a, $a_{i,1}$ for the task τ_i is the same value as O_i. Engineers use this parameter to prevent all tasks from running at the same time, which leads to the worst-case scenario.



(a) Periodic task



(b) Aperiodic task

Figure 2.1: Scheduling for periodic and aperiodic tasks.

- Period (T_i): is the time interval between each arrival of task τ_i after the first arrival. As task τ_i is activated regularly, the kth arrival time ($a_{i,k}$) is determined by $O_i + (k-1) \times T_i$.

- Execution time (C_i): is the amount of time required to execute the work of task τ_i without being disturbed by other tasks. It is typically applied to the worst-case execution time (WCET) to consider the worst-case scenario. After a task arrives at $a_{i,k}$, the task will get a resource, i.e., a processing core, and start its work when the resource is available. In the Figure 2.1a, the first arrival started at $s_{i,1}$ and ended at $e_{i,1}$. The difference between $s_{i,1}$ and $e_{i,1}$ becomes the C_i. The task execution can be preempted by the other tasks; when this occurs, then the completion time is delayed, e.g., $e_{i,2}$ as seen in Figure 2.1a.

- Deadline (D_i): is the time constraint in which task τ_i must complete its execution. As D_i is a relative value, the absolute deadline value for each task arrival is determined by $a_{i,k} + D_i$. If a task arrival has completed its execution after the absolute deadline, the task arrival violates the time constraint. For example, in Figure 2.1a, the first arrival of task τ_i meets its deadline while the second arrival misses it, i.e., $e_{i,2} > a_{i,2} + D_i$. This time constraint must be satisfied in the hard real-time tasks. However, some tasks can tolerate occasional deadline misses depending on the operating context of a system, which are called soft or weakly hard real-time tasks.

Aperiodic tasks have the same properties as periodic tasks except for their activation-related properties. These tasks have irregular activation from external stimuli. Although they do not have limitations in the task arrivals in general, for the sake of analysis, we introduce inter-arrival times $[T_i^{min}, T_i^{max}]$, which are the minimum and maximum time interval between two consecutive arrivals of τ_i, instead of T_i. Figure 2.1b describes the behaviors of an aperiodic task τ_j according to the inter-arrival times. Based on the $[T_j^{min}, T_j^{max}]$, each task arrival time $a_{j,k}$ is determined by the $k-1$th arrival time of τ_j. Specifically, for $k > 1$, $a_{j,k} \in [a_{j,k-1} + T_j^{min}, a_{j,k-1} + T_j^{max}]$ and for $k = 1$, $a_{j,1} \in [T_j^{min}, T_j^{max}]$, where $a_{j,k} \leq \mathbf{t}$. Note that aperiodic tasks also have the same parameters as periodic tasks, but they do not use offset, i.e., O_j=0 as the first arrival time of the task already varies.

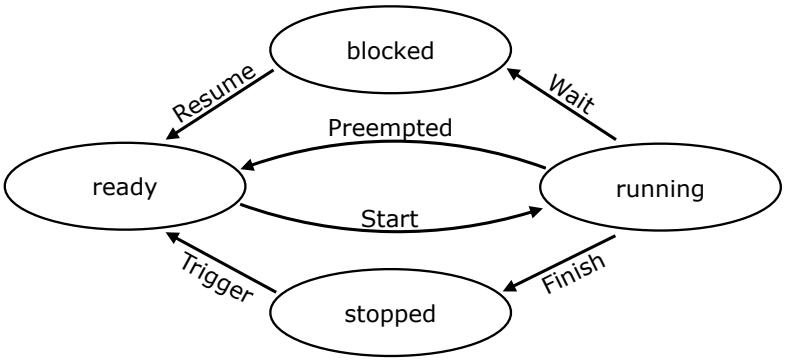


Figure 2.2: The lifecycle of a task state transition model.

In real-time analysis, sporadic tasks are often used for the tasks that have irregular task arrivals and hard time constraints [119]. However, we do not introduce additional notations for the sporadic tasks because we have enough properties to describe them.

2.1.2 Scheduling in real-time systems

A real-time operating system (RTOS) schedules a set of n tasks in Γ using a priority-based scheduling policy [31]. In this policy, each task τ_i is assigned a priority value to determine the execution order at each time point. We denote task priority by P_i. Specifically, a task τ_i can preempt another task τ_j when the priority value of τ_j is less than the value of τ_i, i.e., $P_j < P_i$. According to the priority assignments, RTOS manages their tasks using task states.

Task states. Figure 2.2 describes the task state transition model. While an RTOS usually defines more task states, we only show some of them for the modeling: *stopped*, *ready*, *running*, and *blocked*. An explanation of each state and the transition is provided below.

- *stopped.* When a system starts, each real-time task is set to the *stopped* state. A task can be in the *ready* state due to (time-based or event-based).
- *ready.* When a task becomes the *ready* state, a scheduler in an RTOS puts it into the *ReadyQueue* which keeps a list of tasks that are waiting so they can be assigned to the processing core. According to its scheduling policy, the scheduler assigns a processing core to a task when the processing core is available. With a fixed-priority assignment scheduling policy, the highest priority task among the tasks in the *ReadyQueue* is assigned a processing core. Based on the first-in, first-out (FIFO) method, the first arrived task in *ReadyQueue* is assigned first. A task that is assigned a processing core is in the *running* state when it begins its execution.
- *running.* A task in the *running* state is actually executing its work on a processing core. The number of running tasks is the same as the number of processing cores. During its execution, a task can be preempted when a higher priority task has arrived in *ReadyQueue*. Then, a *running* task goes into the *ready* state again and waits for the next available time. When all the task execution are completed, the task moves to the *stopped* state to wait for the next activation.
- *blocked.* The task execution can be stopped temporarily for a temporal event or an external event. For example, when a task calls for a sleep function in its routine, it puts itself into the *blocked* state. When

the specified time for the sleep function has elapsed, the task goes into the *ready* state. When a task requests a shared resource, it may need to be blocked until the resource becomes available. Tasks in the *blocked* state are not managed by a scheduler until their requests are satisfied.

Schedulability analysis. Based on the task and state models, the schedulability of a real-time system is determined by analyzing the schedule results. A *schedulable* system is defined as the scenario in which the schedule results have no deadline violation ($e_{i,k} \leq a_{i,k} + D_i$) for all task arrivals of all the tasks in the system. A theoretical schedulability analysis approach is suggested based on utilization [118]. Let $\Gamma = \{\tau_1, \tau_2, \cdots, \tau_n\}$ be a set of tasks to be scheduled. A utilization for a task τ_1, denoted by u_i, is calculated by its WCET and period, i.e., $u_i = C_i/T_i$. The total utilization of tasks for the given system Γ is calculated as below:

$$U = \sum_{i=1}^{|\Gamma|} C_i/T_i$$

For aperiodic tasks, the utilization is calculated by C_i/T_i^{min}, assuming the task arrives at every T_i^{min} which is the conservative scenario. If the tasks follow the rate monotonic priority assignment, the approach recommends the total utilization U to be less than about 0.7, which is a sufficient condition [118]. However, the theoretical approach is too conservative because of inaccuracies in estimating the timing parameters and simplifying the assumptions regarding aperiodic tasks and task relationships. Hence, simulation-based schedulability analysis has been applied [36, 5].

2.2 Genetic algorithm

A genetic algorithm (GA) is one of the meta-heuristic algorithms used to solve optimization problems. GA is a population-based global optimization algorithm. Many software engineering problems, such as software testing and design, have been formulated to solve optimization problems and have been successfully applied [89]. Inspired by the principles of genetics and natural selection in biology, GA builds an initial population and iterates evolving it by applying genetic operators, such as selection, crossover, mutation, and replacing. This approach has the following advantages: it deals with a vast number of and different types of variables, it provides alternative solutions instead of one best solution, and it allows for working on parallel computers [94]. These are the reasons why GA is available to a vast number of domains. In this section, we introduce the basic concepts of GA with single-objective. We then explain some GA variants, such as multi-objective GA and coevolutionary algorithms.

2.2.1 Single-objective GA

GA was invented by John Holland in 1975 [97]. The GA searches for the best solution in a multidimensional search space by investigating the fitness values of the candidate solutions with a single objective. Algorithm 2.1 describes a pseudo-code of the GA. The algorithm first randomly generates an initial population P equal to the population size $popsize$ (lines 4-7). The algorithm then repeats the evolution process by evaluating P (lines 12-16), breeding a new population Q (lines 19-25), and replacing the population P with Q (line 28). The algorithm stops the evolution process when it finds the ideal solution or reaches the assigned time budget.

To generate a population P, GA randomly creates each individual, which is considered to be a candidate solution (line 6). An individual is also called a *chromosome*. A chromosome is usually

Algorithm 2.1: A pseudo-code of single-objective genetic algorithm (GA)

```
1  popsize ← desired population size
2
3  // Build initial population
4  P ← {}
5  for popsize times do
6  |  P ← P ∪ {CreateNewIndividual()}
7  end for
8
9  Best ← □
10 repeat
11 |  // Evaluate each individual
12 |  for each individual P_i ∈ P do
13 |  |  AssessFitness(P_i)
14 |  |  if Best = □ or Fitness(P_i) > Fitness(Best) then
15 |  |  |  Best ← P_i
16 |  end for
17 |
18 |  // Breed new population
19 |  Q ← {}
20 |  for popsize/2 times do
21 |  |  Parent P_a ← SelectParent(P)
22 |  |  Parent P_b ← SelectParent(P)
23 |  |  Children C_a, C_b ← Crossover(P_a, P_b)
24 |  |  Q ← Q ∪ {Mutation(C_a), Mutation(C_a)}
25 |  end for
26 |
27 |  // replace the population with new population
28 |  P ← Replace(P, Q)
29 until Best is the ideal solution or we have run out of time
30 return Best
```

represented by a fixed-length vector, which is also called *gene*s. Each gene can be any type of variable such as a boolean, an integer, a floating-point, and even another vector depending on the applications. This representation is important because it also affects the breeding operators. The GA allows for a flexible-length vector, but it requires careful design of the breeding operators. When GA creates an individual, it combines the value of the genes that are randomly selected within their criteria, e.g., a random value in [0,100] for an integer type of gene.

Given the population P, GA assesses a fitness value for each individual $P_i \in P$ (line 13). The fitness value represents how valuable the individual is as a solution. Thus, the fitness function of GA is the key attribute needed to find the optimal solution. The function requires to provide a good guidance of the search, by distinguishing the differences among the individuals as well as by covering all the available individuals. Based on the fitness values, the GA selects the *Best* individual in the population based on the fitness values (lines 14-15).

The breeding of the population P generates a new population G for the next generation using genetic operators. To do this, GA selects two individuals P_a and P_b to be parents. Given two parents, GA produces two children C_a and C_b using a crossover operator. The children are appended to the new population after applying a mutation operator. This process produces the same number of new individuals with the population P, since GA repeats it $popsize/2$ times (line 20).

The selection operator determines which individual would be the parent among the given population P. Tournament selection is the most popular algorithm due to its simplicity [122]. It randomly selects two individuals and chooses the one with the highest fitness. It repeats this process by selecting an individual and comparing it with the last surviving individual. The more the tournament repeats, the more likely it will choose an individual with the highest fitness value.

The crossover operator produces children by making them inherit the traits of their parents. A simple algorithm is the one-point crossover. Assuming that a chromosome is a fixed-length vector, it selects the same point from both parents and mixes genes from the point to the end of the vector. As the operator selects the crossover point randomly, the criteria for each gene can be violated if the value of each gene has related to the other genes. In this case, a repair process can be added. The operator is subject to a crossover rate that determines whether the operator is applied. If the operator is not applied, the children become a copy of their parents.

The mutation operator changes some genes in an individual based on the mutation rate. The operator iterates each gene and changes when a random value is higher than the mutation rate. This operation provides an opportunity to jump to the other search area so that the GA can be prevented from finding a local optimal. However, the higher mutation rate makes GA to be a random search. Thus, it is recommended to use a lower mutation rate. Depending on the application, the mutation operator may also need to consider the repair process.

After breeding a new population Q, replacing the population determines which individuals will be delivered to the next iteration (generation). A simple way of replacing can be merging two populations and selecting only the top fitness individuals. However, this can lead to a premature search due to the lack of diversity in the population. To deal with this, some variants considering elitism are suggested, such as the steady-state GA, which produces only one or two children and replaces some existing individuals in each iteration [121].

2.2.2 Further variants of GA

GAs have been successfully applied in many applications, including software testing [180, 179, 30], requirements [71], and design optimization [66, 88, 90]. Despite the success of GA, there are more complex real-world problems that require complex structures. We introduce some of those variants: multi-objective GA and coevolutionary GA.

Multi-objective GA. In some applications, the problem can have multiple objectives. For example, when deciding on a system design, RTS engineers may face a problem between cost and performance efficiency. This kind of decision-making problem needs to find a cost-effective solution by considering the trade-offs.

Finding the Pareto optimal provides candidate solutions that are considered equally viable for all objectives [94]. Figure 2.3 shows the individuals (black dots) and Pareto optimal (Pareto front) in two-objective search spaces. The Pareto front is

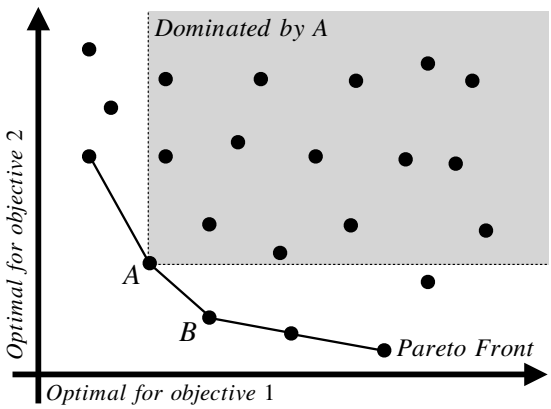


Figure 2.3: Pareto front

the group of individuals that are not dominated by any other individuals. Individual *A* in Figure 2.3 dominates many other individuals, butcannot dominates individual *B* as it is less optimal for Objective 1 while it is more optimal for Objective 2.

A non-dominated sorting GA II (NSGA-II) is one of the popular multi-objective GA algorithms that is uesd to find the Pareto front [63]. Based on the fitness values for each objective, NSGA-II calculates Pareto ranks, which are groups that have the same level of Pareto optimality. The first ranked group becomes the Pareto front. NSGA-II uses the following techniques for its superiority: (1) keeping an archive of the lower-ranked individuals to improve the diversity of the population and (2) introducing crowding-distance between individuals to reduce computations and decrease user's efforts to investigate the trade-off of the large number of solutions. Similarity, other variants, such as SPEA2 [198], PAES [102], and PESA-II [49], have also been developed to solve many complex problems.

Coevolutionary GA. This GA variant evolves multiple populations simultaneously. Coevolutionary GA (coevolution) is also widely applied in bug fixing [15, 185], software design [162, 148], and software planning (job assignment) [150]. The coevolution is usually applied to the problems that occur when the fitness functions of each population interactively affect each other. Historically, coevolution has been classified as *competitive* and *cooperative* [16].

Competitive coevolution is inspired by predators and preys in nature. For example, faster preys survive more easily and generate offspring as they can avoid threats from predators. This impacts the evolution of predators to improve them faster or more intelligently. In the competitive coevolution algorithm, populations of candidates and conditions are formulated. The algorithm assesses the fitness values of the candidate population against the condition population (and vice versa). The algorithm then breeds each population. Finally, it calculates another type of fitness, called external fitness, to monitor the improvement of the candidate population, which is an important step. Unfortunately, formulating external fitness is still an challenging, which is an obstacle for increasing the popularity of the coevolution [146]. Based on the external fitness value, the algorithm produces the *Best* solution.

Cooperative coevolution is used to solve high-dimensional or enormous optimization problems by decomposing the problem into small pieces [147]. Many optimization problems in the real-world are too difficult when an attempt is made to search for the solution with a single population. Like a divide-and-conquer strategy [161], if we can divide a complex problem into sub-problems, we can solve each small problem and then merge the solutions to solve the original problem. Cooperative coevolution separates a complex problem into multiple populations and evolves them with each fitness function. The algorithm then formulates a joint fitness function to consider all the fitness values together so it can find the optimal solution to the original problem. This approach is effective in reducing the computational cost and complexity of the problem.

2.3 Supervised machine learning

Machine learning has played an essential role in many research and industrial domains recently [8]. Supervised machine learning is one category of machine learning techniques that infers a model based on labeled datasets. This type of technique is used to solve *classification* or *regression* problems. The classification technique is a method to predict the data to which it will belong. Many techniques are used to achieve this, such as Naive Bayes [151], Support Vector Machine (SVM) [155], K-Nearest Neighbor [104], and Random Forest [35]. The regression technique is a method to explicitly model the

relationship between an outcome variable and independent variables. Representative techniques are Linear regression [182] and Logistic regression [100].

Among these approaches, logistic regression is valuable because it is adaptable to not only regression problems but also classification problems [164]. Additionally, logistic regression provides an interpretable model that can explain the probabilistic relationship between the dependent and independent variables [100]. Therefore, many studies in the field of software engineering have been applied it [37, 167, 160]. In this section, we provide a brief explanation of logistic regression.

Logistic regression. Logistic regression is a type of statistical tool that builds models for classification and prediction. This model explains a categorical outcome variable. Categorical variables can easily be found in many real-world contexts, such as living or dying, smoking or not, the presence or absence of disease, and level of income [142]. These variables have extremely small levels of values, especially binary variables, which only have a value of 0 or 1. To analyze these variables in relation to other factors (independent variables), it is not possible to construct a model directly like linear regression. Therefore, logistic regression overcomes the issue by taking advantage of the log transformation of odds (also called logit).

Logit enables a binary variable to be continuous in a real number domain based on the probability that a specific value of the variable occurs. Let an outcome variable Y be binary and let the independent variables be X=$\{X_1, X_2, \cdots, X_n\}$. The odds refer to the ratio of the probability that an event Y will occur to the probability that it will not occur, i.e., $p/(1-p)$, where p is a value of a conditional probability $P(Y = 1|X)$. The results of the odds are ranged $[0, \infty]$ and are 1 when the $p = 0.5$. To remove this asymmetric result, the natural log function $ln()$ applies to the odds. It allows its results to lie on $[-\infty, \infty]$ in the real number domain according to the probability p. Now, it is possible to build a model between the logit and independent variables X as follows, like linear regression:

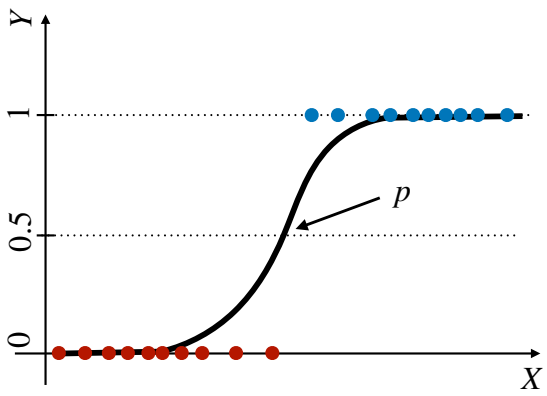


Figure 2.4: Logistic regression model

$$ln\left(\frac{p}{1-p}\right) = c_0 + \sum_{i=1}^{n} c_i X_i$$

where c_0 is a constant and each c_i is the coefficient corresponding to the independent variable X_i. If we rearrange the equation with p as a subject, which can explain the value of Y between 0 and 1 as continuous, the equation becomes as follows:

$$p = \frac{exp^{(c_0 + \sum_{i=1}^{n} c_i X_i)}}{1 + exp^{(c_0 + \sum_{i=1}^{n} c_i X_i)}}$$

Figure 2.4 shows the shape of the model on the independent variables X and probability p for the outcome variable Y. Based on the model, logistic regression infers the coefficients c (i.e., $\{c_0, c_1, \cdots, c_n\}$) using maximum likelihood estimation (MLE) [136] based on labeled data.

Chapter 3

Estimating Probabilistic Safe WCET Ranges of Real-Time Systems at Design Stages

3.1 Introduction

Safety-critical systems, e.g., those used in the aerospace, automotive and healthcare domains, require that their executions always complete before their specified deadlines in all execution scenarios, including the worst cases. The systems that must perform their operations in such a timely manner are known as real-time systems (RTS) [39]. To ensure that a real-time system meets its deadlines, we need an accurate estimation of the worst-case execution times (WCET) of software tasks that concurrently run in the system. For instance, the Anti-lock Braking System (ABS) of a vehicle has to activate within milliseconds after the driver brakes. However, an ABS taking more time for activation than the estimated WCET may result in a vehicle skid due to the wheels locking up.

Accurately estimating WCET values of a real-time system is particularly important at early design stages when real-time tasks are not yet fully implemented. Accurate WCET estimates greatly support engineers during development as they provide targets driving design and implementation choices. Based on the WCET estimates, engineers can make design and implementation decisions, e.g., using either a relational database or an in-memory data storage. In addition, when engineers develop software and hardware components in parallel, which is common in the aerospace, automotive, and healthcare domains, accurately estimated WCET values at early design stages help engineers find optimal configurations of hardware devices, e.g., CPUs, sensors, and actuators, by accounting for time constraints and performance requirements of the system.

Real-time tasks have various parameters such as task priorities, deadlines, inter-arrival times, and WCET values [163, 50]. Among the task parameters, WCET values are typically difficult to accurately estimate at early design stages. The other parameters, however, can be specified or estimated with a high

degree of precision even at early stages. For example, task priorities are typically determined by the selected scheduling policy, e.g., rate monotonic [118], or based on the task criticality levels (i.e., more critical tasks are prioritized over the less critical ones). Task deadlines are typically decided by system requirements. Task inter-arrival times, i.e., the time interval between consecutive task executions, usually depend on system environmental events triggering task executions. In contrast, the WCET values of some tasks may depend on various factors such as implementation decisions, task durations, real-time operating systems, and hardware components. However, these factors may not be fully known at early stages of development, making it difficult to precisely estimate WCET values for real-time tasks [84, 7, 33]. As a result, engineers tend to provide ranges for WCET values instead of point estimates.

The problem of estimating WCET values is, in general, a hard problem. WCET values of real-time tasks impact every possible task schedules. WCET values depend on the content and implementation of tasks and not the schedule. But they impact how the tasks are scheduled. The space of all possible task schedules is very large. In our context, the problem becomes computationally more expensive when WCET values are uncertain and are specified as value ranges instead of single values. Specifically, provided with WCET value ranges, engineers need to have ways to determine for what WCET values, within the given ranges, the system is likely to miss or satisfy its deadline constraints. If engineers know that deadlines are likely met for all or most of the expected WCET ranges, they can consider a wider choice of design and implementation options. Otherwise, in situations where only tight WCET sub-ranges seem acceptable, developers may have to consider more expensive hardware, decreased functionality or performance, or more restricted design and implementation choices.

The WCET estimation problem of real-time systems has been widely studied in the past [29, 84, 85, 7, 87, 33]. To our knowledge, however, most of the existing WCET estimation methods often fail to provide WCET estimates at early design stages because they require inputs that can be defined only at a later point in time such as task implementations and hardware devices. Some model-based approaches [47, 166, 10] try to solve the WCET estimation problem exhaustively by applying a model checker to a real-time model, e.g., parametric timed automata, of the system under analysis. These exhaustive methods are applicable once real-time system models are available. However, such approaches tend to suffer from the state-space explosion problem [48] as the number of software tasks and their different states increase. More recently, stress testing and simulation-based approaches [36, 6] have been proposed to stress RTS and generate test scenarios where their deadline constraints are violated. Such approaches cast the schedulability test problem as an optimization problem to find worst-case task execution scenarios exhibiting deadline misses. However, none of the existing simulation-based approaches account for uncertainties in WCET values and therefore do not handle WCET value ranges. Our work complements the simulation-based stress testing approach and extends it to account for uncertainties in WCET values.

Contributions. In this chapter, we propose a Safe WCET Analysis method For real-time task schEdulability (SAFE) to estimate WCET ranges under which tasks are likely to be schedulable with a probabilistic guarantee. Our approach is based on a stress testing approach [36] using meta-heuristic search [122] in combination with polynomial logistic regression models. Specifically, we use a genetic algorithm [122] to search for sequences of task arrivals which likely lead to deadline misses. Then, logistic regression [100], a statistical classification technique, is applied to infer a *safe WCET border* in the multidimensional WCET space with a probabilistic guarantee. This border aims to partition the given WCET ranges into *safe* and *unsafe* sub-ranges for a selected deadline miss probability p, and thus enables

engineers to investigate trade-offs among different tasks' WCET values. WCET ranges are deemed to be probabilistically safe if tasks, within such ranges, have a high probability to complete their executions before their specified deadlines. In this chapter, for the sake of simplicity, we refer to probabilistically safe WCET ranges as safe WCET ranges. We evaluated our approach by applying it to a complex, industrial satellite system developed by our industry partner, LuxSpace, as well as two industrial systems from different domains and several synthetic systems. Results show that our approach can efficiently and accurately compute safe WCET ranges. SAFE scales to complex industrial systems as an offline analysis method. Execution times of SAFE on our industrial systems are practically acceptable, i.e., at most 27h. To our knowledge, SAFE is the first attempt to estimate safe WCET ranges within which real-time tasks are likely to meet their deadlines for a given level of confidence, while enabling engineers to explore trade-offs among tasks' WCET values. Our full evaluation package is available online [112].

Organization. The remainder of this chapter is structured as follows: Section 3.2 motivates our work. Section 3.3 defines our specific schedulability analysis problem in practical terms. Section 3.4 describes SAFE. Section 3.5 evaluates SAFE. Sections 3.6 compares SAFE with related work. Section 3.7 concludes this chapter.

3.2 Motivating case study

We motivate our work with a mission-critical real-time satellite system, named Attitude Determination and Control System (ADCS), which LuxSpace, a leading system integrator for microsatellites and aerospace systems, has been developing over the years. ADCS determines the satellite's attitude and controls its movements [69]. ADCS controls a satellite in either autonomous or passive mode. In the autonomous mode, ADCS must orient a satellite in proper position on time to ensure that the satellite provides normal service correctly. In the passive mode, operators are able to not only control satellite positions but also maintain the satellite, e.g., upgrading software. Such a maintenance operation does not necessarily need to be completed within a fixed hard deadline; instead, it should be completed within a reasonable amount of time, i.e., soft deadlines. Hence, ADCS is composed of a set of tasks having real-time constraints with hard and soft deadlines.

Engineers at LuxSpace conduct real-time schedulability analysis across different development stages. At an early design stage, when task implementations and system hardware are not available, the engineers use a theoretical schedulability analysis technique [118] which determines that a set of tasks is schedulable if CPU utilization of the task set is less than a threshold, e.g., 69%. As mentioned earlier, at an early design stage, engineers estimate task WCETs as ranges and often assign large values to the upper bounds of such ranges. To be on the safe side, engineers tend indeed to be conservative in their analysis.

Engineers, however, are still faced with the following issues: (1) An analytical schedulability analysis technique, e.g., utilization-based schedulability analysis [118], typically indicates whether or not tasks are schedulable. However, engineers need additional information to understand how tasks miss their deadlines. For instance, a set of tasks may not be schedulable for a few specific sequences of task arrivals. (2) Engineers estimate WCETs without any systematic support; instead, they often rely on their experience of developing tasks providing similar functions-to-develop. This practice typically results in imprecise estimates of WCET ranges, which may cause serious problems, e.g., significantly changing tasks at later development stages. To this end, LuxSpace is interested in SAFE as a way to address these issues in analyzing schedulability.

3.3 Problem description

This section first formalizes task, task relationship, scheduler, and schedulability concepts. We then describe the problem of identifying safe WCET ranges under which tasks likely meet their deadline constraints at a certain level of confidence; i.e., tasks are schedulable with a certain probability.

Task. A real-time system is composed of a set of n tasks that should complete their executions within specified deadlines after they are activated (or arrived). We denote by τ_i a real-time task indexed by i in the range from 1 to n. Every real-time task τ_i has the following properties: priority denoted by P_i, deadline denoted by D_i, and worst-case execution time (WCET) denoted by C_i. Task priority P_i determines if an execution of a task is preempted by another task. Typically, a task τ_i preempts the execution of a task τ_j if the priority of τ_i is higher than the priority of τ_j, i.e., $P_i > P_j$.

The deadline of a task τ_i relative to its arrival time is denoted by D_i. A task deadline can be either *hard* or *soft*. A hard deadline of a task τ_i specifies that τ_i *must* complete its execution within a deadline D_i after τ_i is activated. While violations of hard deadlines are not acceptable, depending on the operating context of a system, violating soft deadlines may be tolerated to some extent. Note that, for notational simplicity, we do not introduce new notations to distinguish between hard and soft deadlines. In this chapter, we refer to a hard deadline as a deadline. Section 3.4 further discusses how our approach manages hard and soft deadlines.

We denote by C_i^{min} and C_i^{max}, respectively, the minimum and the maximum WCET values of a task τ_i. As discussed in the introduction, at an early development stage, it is difficult to provide exact WCET values of real-time tasks. Hence, we assume that engineers specify WCETs using a range of values, instead of single values, by indicating estimated minimum and maximum values that they think each task's WCET can realistically take.

In this chapter, real-time tasks are either *periodic* or *aperiodic*. Periodic tasks, which are typically triggered by timed events, are invoked at regular intervals specified by their *period*. We denote by T_i the period of a periodic task τ_i, i.e., a fixed time interval between subsequent activations (or arrivals) of τ_i. Aperiodic tasks have irregular arrival times and are activated by external stimuli which occur irregularly, and hence, in general, there is no limit on the arrival times of an aperiodic task. However, in real-time analysis, we typically specify a minimum inter-arrival time denoted by T_i^{min} and a maximum inter-arrival time denoted by T_i^{max} indicating the minimum and maximum time intervals between two consecutive arrivals of an aperiodic task τ_i. In real-time analysis, sporadic tasks are often separately defined as having irregular arrival intervals and hard deadlines [119]. In our conceptual definitions, however, we do not introduce new notations for sporadic tasks because the deadline and period concepts defined above are sufficient to characterize sporadic tasks. Note that for a periodic task τ_i, we have $T_i^{min} = T_i^{max} = T_i$. Otherwise, for an aperiodic task τ_j, we have $T_j^{max} > T_j^{min}$.

Task relationship. The execution of a task τ_i depends not only on its own parameters described above, e.g., priority P_i and period T_i, but also on its relationships with other tasks. Relationships between tasks are typically determined by task interactions related to accessing shared resources [5], such as memory, file, and IO devices. Specifically, if two tasks τ_i and τ_j access a shared resource in a mutually exclusive way, τ_i may be blocked from executing for the period during which τ_j accesses the resource. We denote by $dp(\tau_i, \tau_j)$ the resource-dependency relation between tasks τ_i and τ_j that holds if τ_i and τ_j have mutually exclusive access to a shared resource such that they cannot be executed in parallel or preempt each other, but one can execute only after the other has completed its access to the resource. We note

that resource-dependency relations are defined at the level of tasks, following prior works [120, 11, 64] describing the industrial case study systems used in our experiments (see Section 3.5.2). The $dp(\tau_i, \tau_j)$ relation is symmetric, i.e., $dp(\tau_i, \tau_j) = dp(\tau_j, \tau_i)$.

Scheduler. Let Γ be a set of tasks to be scheduled by a real-time scheduler. A scheduler then dynamically schedules executions of tasks in Γ according to the tasks' arrivals and the scheduler's scheduling policy over the scheduling period $\mathbb{T} = [0, \mathbf{t}]$. We denote by $a_{i,k}$ the kth arrival time of a task $\tau_i \in \Gamma$. The first arrival of a periodic task τ_i does not always occur immediately at the system start time 0. Such offset time from the system start time 0 to the first arrival time $a_{i,1}$ of τ_i is denoted by O_i. For a periodic task τ_i, the kth arrival of τ_i within $\mathbb{T}$ is $a_{i,k} \leq \mathbf{t}$ and is computed by $a_{i,k} = O_i + (k-1) \cdot T_i$. For an aperiodic task τ_j, $a_{j,k}$ is determined based on the $k-1$th arrival time of τ_j and its minimum and maximum arrival times. Specifically, for $k > 1$, $a_{j,k} \in [a_{j,k-1} + T_j^{min}, a_{j,k-1} + T_j^{max}]$ and, for $k = 1$, $a_{j,1} \in [T_j^{min}, T_j^{max}]$ where $a_{j,k} \leq \mathbf{t}$.

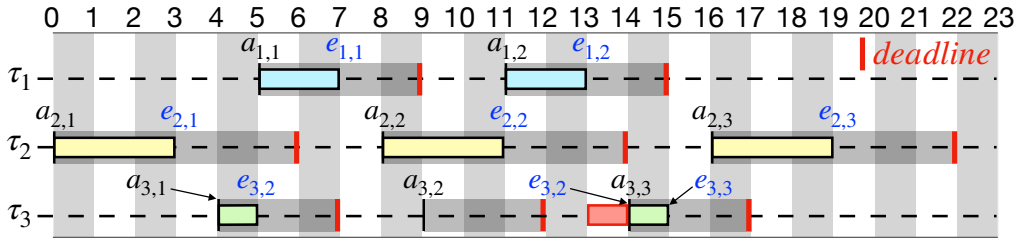
A scheduler reacts to a task arrival at $a_{i,k}$ to schedule the execution of τ_i. Depending on a scheduling policy (e.g., rate monotonic scheduling policy [118] and single-queue multi-core scheduling policy [17]), an arrived task τ_i may not start its execution at the same time as it arrives when a higher priority task is executing. Also, task executions may be interrupted due to preemption. We denote by $e_{i,k}$ the end execution time for the kth arrival of a task τ_i. Depending on actual worst-case execution time of a task τ_i, denoted by C_i, within its WCET range $[C_i^{min}, C_i^{max}]$, the $e_{i,k}$ end execution time of τ_i satisfies the following: $e_{i,k} \geq a_{i,k} + C_i$.

During the system operation, a scheduler generates a *schedule scenario* which describes a sequence of task arrivals and their end time values. We define a schedule scenario as a set S of tuples $(\tau_i, a_{i,k}, e_{i,k})$ indicating that a task τ_i has arrived at $a_{i,k}$ and completed its execution at $e_{i,k}$. Due to the randomness of task execution times and aperiodic task arrivals, a scheduler may generate a different schedule scenario in different runs of a system.

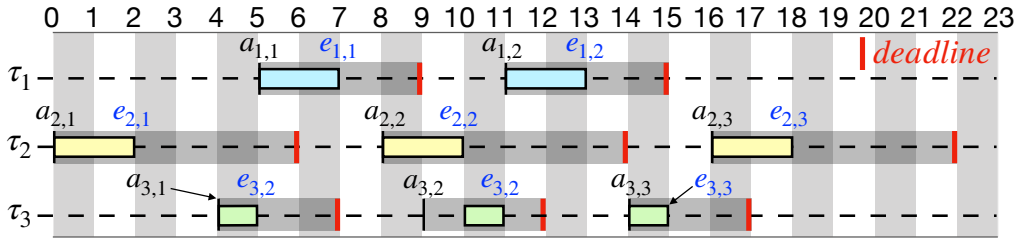
Figure 3.1 shows two schedule scenarios produced by a scheduler over the $[0, 23]$ time period of a system run. Both Figure 3.1a and Figure 3.1b describe executions of three tasks, τ_1, τ_2, and τ_3 arrived at the same time stamps (see $a_{i,k}$ in the figures). In both scenarios, the aperiodic task τ_1 is characterised by: $T_1^{min} = 5$, $T_1^{max} = 10$, $D_1 = 4$, and $C_1^{min} = C_1^{max} = 2$. The periodic task τ_2 is characterised by: $T_2 = 8$ and $D_2 = 6$. The aperiodic task τ_3 is characterised by: $T_3^{min} = 3$, $T_3^{max} = 20$, $D_3 = 3$, and $C_3^{min} = C_3^{max} = 1$. The priorities of the three tasks satisfy the following: $P_1 > P_2 > P_3$. In both scenarios, task executions can be preempted depending on their priorities. We note that a WCET range of the τ_2 task is set to $C_2^{min} = 1$ and $C_2^{max} = 3$ in Figure 3.1a, and $C_2^{min} = 1$ and $C_2^{max} = 2$ in Figure 3.1b. Then, Figure 3.1a can be described by the $S_a = \{(\tau_1, 5, 7), \ldots, (\tau_2, 0, 3), \ldots, (\tau_3, 9, 14), (\tau_3, 14, 15)\}$ schedule scenario; and Figure 3.1b by $S_b = \{(\tau_1, 5, 7), \ldots, (\tau_2, 0, 2), \ldots, (\tau_3, 9, 11), (\tau_3, 14, 15)\}$.

Schedulability. Given a schedule scenario S, a task τ_i is *schedulable* if τ_i completes its execution before its deadline, i.e., for all $e_{i,k}$ observed in S, $e_{i,k} \leq a_{i,k} + D_i$. Let Γ be a set of tasks to be scheduled by a scheduler. A set Γ of tasks is then schedulable if for every schedule S of Γ, we have no task $\tau_i \in \Gamma$ that misses its deadline.

As shown in Figure 3.1a, a deadline miss occurs after the second arrival of τ_3, i.e., $e_{3,2} > a_{3,2} + D_3$. During $[a_{3,2}, a_{3,2} + D_3]$ period, the τ_3 task cannot execute because the other tasks τ_1 and τ_2 with higher priorities are executing. Thus, τ_3 is not schedulable in the schedule scenario of Figure 3.1a. This scheduling problem can be solved by restricting tasks' WCET ranges as discussed below.



(a) Deadline miss



(b) No deadline miss

Figure 3.1: Example schedule scenarios of three tasks, τ_1, τ_2, and τ_3, running on a single core system. (a) τ_3 is not schedulable, i.e., $e_{3,2} > a_{3,2} + D_3$. (b) All the three tasks are schedulable. When τ_2 executes over 3 (WCET) time units, it causes a deadline miss of τ_3. When the WCET is reduced to 2, the three tasks are schedulable even for the same sequence of task arrivals.

Problem. Uncertainty in task WCET values at an early development stage is a critical issue preventing the effective design and assessment of mission-critical real-time systems. Upper bounds of WCETs correspond to worst-case WCET values and have a direct impact on deadline misses as larger WCET values increase their probability. Lower bounds of WCETs are estimates of tasks' best-case WCET values, below which task implementations are likely not feasible. Our approach aims to determine the maximum upper bounds for WCET under which tasks are likely to be schedulable, at a given level of risk, and thus provides an objective to engineers implementing the tasks. Specifically, for every task $\tau_i \in \Gamma$ to be analyzed, our approach computes a new upper bound value for the WCET range of τ_i (denoted by C_i^{max*}) such that $C_i^{max*} \leq C_i^{max}$ and by restricting the WCET range of τ_i to C_i^{max*} we should, at a certain level of confidence, no longer have deadline misses. That is, tasks Γ become schedulable, with a certain probability, after restricting the maximum WCET value of τ_i to C_i^{max*}. For instance, as shown in Figure 3.1b, restricting the maximum WCET of τ_2 from $C_2^{max} = 3$ to $C_2^{max*} = 2$ enables all the three tasks to be schedulable.

We note that, in our context, both arrival time ranges for aperiodic tasks and WCET ranges for all tasks are represented as continuous intervals. Since our approach works based on sampling values from these continuous ranges, our approach cannot be exhaustive and cannot provide a guarantee that the tasks can always be schedulable after restricting their WCET ranges. Our approach instead relies on sampling values within the WCET and arrival time ranges, simulating the scheduler behavior using the sampled values and observing whether, or not, a deadline miss occurs. In lieu of exhaustiveness, we rely on statistical and machine learning techniques to provide probabilistic estimates indicating how confident we are that a given set of tasks are schedulable.

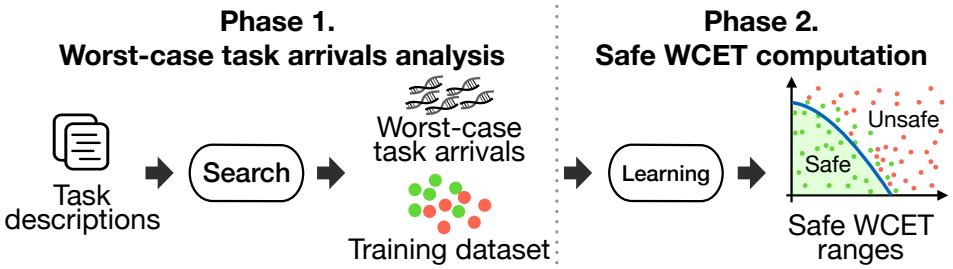


Figure 3.2: An overview of our Safe WCET Analysis method For real-time task schEdulability (SAFE).

3.4 Approach

Figure 3.2 shows an overview of our Safe WCET Analysis method For real-time task schEdulability (SAFE). Phase 1 of SAFE aims at searching worst-case task-arrival sequences. A task-arrival sequence is worst-case if deadline misses are maximized or, when this is not possible, tasks complete their executions as close to their deadlines as possible. Building on existing work, we identify worst-case task-arrival sequences using a search-based approach relying on genetic algorithms. Phase 2 of SAFE, which is the main contribution of this chapter, aims at computing safe WCET ranges under which tasks are likely to be schedulable. To do so, relying on logistic regression and an effective sampling strategy, we augment the worst-case task-arrival sequences generated in Phase 1 to compute safe WCET ranges with a certain deadline miss probability, indicating a degree of risk. We describe in detail these two phases next.

3.4.1 Phase 1: worst-case task arrivals

The first phase of SAFE finds worst-case sequences in the space of possible sequences of task arrivals, defined by their inter-arrival time characteristics. As SAFE aims to provide conservative, safe WCET ranges, we optimize task arrivals to maximize task completion times and deadline misses, and indirectly minimize safe WCET ranges (see the safe area visually presented in Figure 3.2). We address this optimization problem using a single-objective search algorithm. Following standard practice [74], we describe our search-based approach for identifying worst-case task arrivals by defining the solution representation, the scheduler, the fitness function, and the computational search algorithm. We then describe the dataset of sequences generated by search and then used for training our logistic regression model to compute safe WCET ranges in the second phase of SAFE.

Our approach in Phase 1 is based on past work [36], where a specific genetic algorithm configuration was proposed to find worst-case task arrival sequences. One important modification though is that we account for uncertainty in WCET values through simulations for evaluating the magnitude of deadline misses.

Representation. Given a set Γ of tasks to be scheduled, a feasible solution is a set A of tuples $(\tau_i, a_{i,k})$ where $\tau_i \in \Gamma$ and $a_{i,k}$ is the kth arrival time of a task τ_i. Thus, a solution A represents a valid sequence of task arrivals of Γ (see valid $a_{i,k}$ computation in Section 3.3). Let $\mathbb{T} = [0, \mathbf{t}]$ be the time period during which a scheduler receives task arrivals. The size of A is equal to the number of task arrivals over the $\mathbb{T}$ time period. Due to the varying inter-arrival times of aperiodic tasks (Section 3.3), the size of A will vary across different solutions.

Scheduler. SAFE uses a simulation technique for analyzing the schedulability of tasks to account for the uncertainty in WCET values and scalability issues. For instance, an inter-arrival time of a software update task in a satellite system is approximately at most three months. In such cases, conducting an analysis based on an actual scheduler is prohibitively expensive. Instead, SAFE uses a real-time task scheduling simulator, named SafeScheduler, which samples WCET values from their ranges for simulating task executions and applies a scheduling policy, i.e., single-queue multi-core scheduling policy [17], based on discrete simulation time events. Note that we chose the single-queue multi-core scheduling policy for SafeScheduler since our case study systems (described in Section 3.5.2) rely on this policy.

SafeScheduler takes a feasible solution A for scheduling a set Γ of tasks as an input. It then outputs a schedule scenario as a set S of tuples $(\tau_i, a_{i,k}, e_{i,k})$ where $a_{i,k}$ and $e_{i,k}$ are the kth arrival and end time values of a task τ_i, respectively. Recall from Section 3.3 that SafeScheduler computes task arrivals based on periodic tasks' offsets and periods and aperiodic tasks' inter-arrival times.For each task τ_i, SafeScheduler computes $e_{i,k}$ based on its scheduling policy and a selected WCET value for τ_i within the WCET range $[C_i^{min}, C_i^{max}]$, while accounting for resource-dependency relationships (see Section 3.3). Hence, each run of SafeScheduler for the same input solution A will likely produce a different schedule scenario.

SafeScheduler implements a single-queue multi-core scheduling policy [17], which schedules a task τ_i with explicit priority P_i and deadline D_i. When tasks arrive, SafeScheduler puts them into a single queue that contains tasks to be scheduled. At any simulation time, if there are tasks in the queue and multiple cores are available to execute tasks, SafeScheduler first fetches a task τ_i from the queue in which τ_i has the highest priority P_i. SafeScheduler then allocates task τ_i to any available core. Note that if task τ_i shares a resource with a running task τ_j in another core, i.e., the $dp(\tau_i, \tau_j)$ resource-dependency relationship holds, SafeScheduler follows standard task-blocking rules [119], i.e., τ_i will be blocked until τ_j releases the shared resource.

SafeScheduler works under the assumption that context switching time is free, which is also a working assumption in many scheduling analysis methods [118, 20, 64]. Note that the assumptions are practically valid and useful at an early development step in the context of real-time analysis. For instance, our collaborating partner accounts for the waiting time of tasks due to context switching between tasks through adding some extra time to WCET ranges at the task design stage. Note that SAFE can be applied with any scheduling policy, including those that account for context switching time and multiple queues.

Fitness. Given a feasible solution A for a set Γ of tasks, we formulate a fitness function, $f(A, \Gamma^\delta, ns)$, to quantify the degree of deadline misses regarding a set $\Gamma^\delta \subseteq \Gamma$ of target tasks, where ns is a number of SafeScheduler runs to account for the uncertainty in WCET. SAFE provides the capability of selecting target tasks Γ^δ as practitioners often need to focus on the most critical tasks. We denote by $dist(\tau_i, k)$ the distance between the end time and the deadline of the kth arrival of task τ_i and define $dist(\tau_i, k) = e_{i,k} - a_{i,k} + D_i$ (see Section 3.3 for the notation end time $e_{i,k}$, arrival time $a_{i,k}$, and deadline D_i).

To compute the $f(A, \Gamma^\delta, ns)$ fitness value, SAFE runs SafeScheduler ns times for A and obtains ns schedule scenarios $S_1, S_2, \ldots, S_{ns}$. For each schedule scenario S_h, we denote by $dist_h(\tau_i, k)$ the distance between the end and deadline time values corresponding to the kth arrival of the task τ_i observed in S_h. We denote by $lk(\tau_i)$ the last arrival index of a task τ_i in A. SAFE aims to maximize the $f(A, \Gamma^\delta, ns)$

Table 3.1: An example operation of SafeCrossover. It swaps all task arrivals of task τ_1 and τ_2 between two parent solutions A_p and A_q to produce offspring A'_p and A'_q.

	Task τ_1	Task τ_2	Task τ_3
Parent A_p	$(\tau_1, 5)$, $(\tau_1, 11)$	$(\tau_2, 8)$, $(\tau_2, 16)$	$(\tau_3, 4)$, $(\tau_3, 10)$
Parent A_q	$(\tau_1, 3)$, $(\tau_1, 7)$, $(\tau_1, 14)$	$(\tau_2, 6)$, $(\tau_2, 13)$	$(\tau_3, 5)$, $(\tau_3, 8)$, $(\tau_3, 13)$
Child A'_p	$(\tau_1, 3)$, $(\tau_1, 7)$, $(\tau_1, 14)$	$(\tau_2, 6)$, $(\tau_2, 13)$	$(\tau_3, 4)$, $(\tau_3, 10)$
Child A'_q	$(\tau_1, 5)$, $(\tau_1, 11)$	$(\tau_2, 8)$, $(\tau_2, 16)$	$(\tau_3, 5)$, $(\tau_3, 8)$, $(\tau_3, 13)$

fitness function defined as follows:

$$f(A, \Gamma^{\delta}, ns) = \sum_{h=1}^{ns} \max_{\tau_i \in \Gamma^{\delta},\ k \in [1, lk(\tau_i)]} dist_h(\tau_i, k)/ns$$

We note that soft deadline tasks also require to execute within reasonable execution time ranges. Hence, engineers also estimate safe WCET ranges for soft deadline tasks. As the above fitness function returns a quantified degree of deadline misses, SAFE uses such function for both soft and hard deadline tasks.

Computational search. SAFE employs a steady-state genetic algorithm [122]. The algorithm breeds a new population for the next generation after computing the fitness of a population. The breeding for generating the next population is done by using the following genetic operators: (1) *Selection.* SAFE selects candidate solutions using a tournament selection technique, with the tournament size equal to two which is the most common setting [77]. (2) *Crossover.* Selected candidate solutions serve as parents to create offspring using a crossover operation. (3) *Mutation.* The offspring are then mutated. Below, we describe our crossover and mutation operators.

Crossover. A crossover operator is used to produce offspring by mixing traits of parent solutions. SAFE modifies the standard one-point crossover operator [122] as two parent solutions A_p and A_q may have different sizes, i.e., $|A_p| \neq |A_q|$. Let $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of tasks to be scheduled. Our crossover operator, named SafeCrossover, first randomly selects an aperiodic task $\tau_i \in \Gamma$. For all $j \in [1, i]$ and $\tau_j \in \Gamma$, SafeCrossover then swaps all τ_j arrivals between two solutions A_p and A_q. As the size of Γ is fixed for all solutions, SafeCrossover can cross over two solutions that may have different sizes.

Table 3.1 shows an example operation of SafeCrossover using a system with three aperiodic tasks, τ_1, τ_2, and τ_3. Let two parent solutions A_p and A_q be as follows: $A_p = \{(\tau_1, 5), \ldots, (\tau_2, 8), \ldots, (\tau_3, 10)\}$ and $A_q = \{(\tau_1, 3), \ldots, (\tau_2, 6), \ldots, (\tau_3, 13)\}$, where (τ_i, t) denotes task τ_i arrives at time t. Given the two parents A_p and A_q, SafeScheduler randomly selects a task, i.e., τ_2 in this example, then it swaps all arrivals of τ_1 and τ_2 between A_p and A_q. As shown in Table 3.1, SafeCrossover then generates the offspring A'_p and A'_q as follows: $A'_p = \{(\tau_1, 3), \ldots, (\tau_2, 6), \ldots, (\tau_3, 10)\}$ and $A'_q = \{(\tau_1, 5), \ldots, (\tau_2, 8), \ldots, (\tau_3, 13)\}$. The shaded (resp. unshaded) cells in Table 3.1 indicate which task arrivals in child A'_q (resp. A'_p) come from which parent.

Mutation operator SAFE uses a heuristic mutation algorithm, named SafeMutation. For a solution A, SafeMutation mutates the kth task arrival time $a_{i,k}$ of an aperiodic task τ_i with a mutation probability. SafeMutation chooses a new arrival time value of $a_{i,k}$ based on the $[T_i^{min}, T_i^{max}]$ inter-arrival time range of τ_i. If such a mutation of the kth arrival time of τ_i does not affect the validity of the k+1th arrival time of τ_i, the mutation operation ends. Specifically, let $a^*_{i,k}$ be a mutated value of $a_{i,k}$. In case $a_{i,k+1} \in [a^*_{i,k} + T_i^{min}, a^*_{i,k} + T_i^{max}]$, SafeMutation returns the mutated A solution.

After mutating the kth arrival time $a_{i,k}$ of a task τ_i in a solution A, if the k+1th arrival becomes invalid, SafeMutation corrects the remaining arrivals of τ_i. We denote by $a^*_{i,k}$ the mutated kth arrival time of τ_i. For all the arrivals of τ_i after $a^*_{i,k}$, SafeMutation first updates their original arrival time values by adding the difference $a^*_{i,k} - a_{i,k}$. Let $\mathbb{T} = [0, \mathbf{t}]$ be the scheduling period. SafeMutation then removes some arrivals of τ_i if they are mutated to arrive after $\mathbf{t}$ or adds new arrivals of τ_i while ensuring that all tasks arrive within $\mathbb{T}$.

Given the offspring presented in Table 3.1, SafeMutation, for example, mutates a child solution $A'_p = \{(\tau_1, 3), (\tau_1, 7), (\tau_1, 14), \ldots, (\tau_3, 10)\}$. Let $[T_1^{min}, T_1^{max}] = [2, 8]$ be the inter-arrival time range of task τ_1, $\mathbb{T} = [0, 18]$ be the time period during which SafeScheduler receives task arrivals, and SafeMutation selects the second arrival of task τ_1, i.e., $(\tau_1, 7)$ in Table 3.1, to mutate. Based on the inter-arrival time range of τ_1, SafeMutation randomly chooses a new arrival time, e.g., 5, for the second arrival of τ_1. The third arrival $(\tau_1, 14)$ of τ_1 then became invalid due to the mutated second arrival $(\tau_1, 5)$; i.e., τ_1 cannot arrive at time 14 because $14 \notin [5 + 2, 5 + 8]$, where $[T_1^{min}, T_1^{max}] = [2, 8]$. According to the correction procedure described above, the third arrival of τ_1 is modified to $(\tau_1, 12)$ as $12 = 14 + (5 - 7)$, where 14, 5, and 7 are, respectively, the original thrid arrival time of τ_1, the original second arrival time of τ_1, and the mutated second arrival time of τ_1. As SafeScheduler can receive new arrivals of τ_1 after time 12, SafeMutation may add new arrivals of τ_1 based on the inter-arrival time range of τ_1.

We note that when a system is only composed of periodic tasks, SAFE will skip searching for worst-case arrival sequences as arrivals of periodic tasks are deterministic (see Section 3.3), but will nevertheless generate the labeled dataset described below. When needed, SAFE can be easily extended to manipulate varying offset (and period) values for periodic tasks, in a way identical to how we currently handle inter-arrival times.

labeled dataset. SAFE infers safe WCET ranges using a supervised learning technique [152] which requires a labeled dataset, namely logistic regression. In our context, a supervised learning technique creates a model that correlates tasks' WCET values with schedulability results indicating whether these tasks meet their deadlines or not. Supervised learning is conducted based on pairs of tasks' WCET values and a schedulability result, i.e., a labeled dataset. Specifically, SAFE uses logistic regression because it allows engineers to have probabilistic interpretation of safe WCET ranges and to investigate trade-off relationships among different tasks' WCETs. Section 3.4.2 describes this learning process in detail.

Recall from the fitness computation described above, SAFE runs SafeScheduler ns times to obtain schedule scenarios S=$\{S_1, S_2, \ldots, S_{ns}\}$, and then computes a fitness value of a solution A based on S. We denote by W_h a set of tuples (τ_i, C_i) representing that a task τ_i has the C_i WCET value in the S_h schedule scenario. Let $\vec{L}$ be a labeled dataset to be created by the first phase of SAFE. We denote by ℓ_h a label indicating whether or not a schedule scenario S_h has any deadline miss for any of the target tasks in Γ^δ, i.e., ℓ_h is either *safe* or *unsafe* which denotes, respectively, no deadline miss or deadline miss. For each fitness computation, SAFE adds ns number of tuples (W_h, ℓ_h) to $\vec{L}$. Specifically, for a schedule scenario S_h, SAFE adds $(W_h, unsafe)$ to $\vec{L}$ if there are $\tau_i \in \Gamma^\delta$ and $k \in [1, lk(i)]$ such that $dist_h(\tau_i, k) > 0$; otherwise SAFE adds $(W_h, safe)$ to $\vec{L}$.

3.4.2 Phase 2: safe ranges of WCET

In Phase 2, SAFE computes safe ranges of WCET values under which target tasks are likely to be schedulable. To do so, SAFE applies a supervised machine learning technique to the labeled dataset

Algorithm 3.1: SafeRefinement. An algorithm for computing safe WCET ranges under which target tasks are schedulable. The algorithm consists of three steps as follows: "reduce complexity", "handle imbalanced dataset", and "refine model" steps.

Input: - $\vec{L}$: labeled dataset obtained from the SAFE search
- G: Worst solutions obtained from the SAFE search
- ns: Number of WCET samples per solution
- nl: Number of logistic regression models
- pt: Precision threshold

Output: - m: Safe WCET model
- p: Probability of deadline misses

1: //step 1. reduce complexity
2: $\vec{L}^r \leftarrow$ `ReduceDimension(`$\vec{L}$`)` //feature reduction
3: $m \leftarrow$ `StepwiseRegression(`$\vec{L}^r$`)` //term selection
4: $p \leftarrow$ `Probability(`m, $\vec{L}^r$`)`
5: //step 2. handle imbalanced dataset
6: $\vec{L}^b \leftarrow$ `HandleImbalance(`$\vec{L}^r$, m`)`
7: //step 3. refine model
8: **for** nl times **do**
9: //step 3.1. add new data instances
10: **for** each solution $A \in G$ **do**
11: $\{S_1, S_2, \ldots, S_{ns}\} \leftarrow$ `RunSafeScheduler(`A, m, p, ns`)`
12: **for** each scenario $S_h \in \{S_1, S_2, \ldots, S_{ns}\}$ **do**
13: **if** S_h has any deadline miss **then**
14: $\vec{L}^b \leftarrow$ `Add(`$\vec{L}^b$, $(WCET(S_h), unsafe)$`)`
15: **else**
16: $\vec{L}^b \leftarrow$ `Add(`$\vec{L}^b$, $(WCET(S_h), safe)$`)`
17: **end if**
18: **end for**
19: **end for**
20: //step 3.2. learn regression model
21: $m \leftarrow$ `Regression(`m, $\vec{L}^b$`)`
22: $p \leftarrow$ `Probability(`m, $\vec{L}^b$`)`
23: **if** `PrecisionByCrossValidation(`$m, \vec{L}^b$`)` $> pt$ **then**
24: **break**
25: **end if**
26: **end for**
27: **return** m, p

generated by Phase 1 (Section 3.4.1). Specifically, Phase 2 executes SafeRefinement (Algorithm 3.1) which has following steps: complexity reduction, imbalance handling, and model refinement.

Complexity reduction. The "reduce complexity" step in Algorithm 3.1 reduces the dimensionality of a labeled dataset $\vec{L}$ obtained from the first phase of SAFE (line 2). It predicts initial safe WCET ranges based on the WCET variables for the tasks in Γ (line 3) that have the most significant effect on deadline misses for target tasks. A labeled dataset $\vec{L}$ obtained from the first phase of SAFE contains tuples (W, ℓ) where W is a set of WCET values for tasks in Γ and ℓ is a label of W indicating either no deadline miss (safe) or deadline miss (unsafe) (Section 3.4.1). Note that some WCET values in W may not be relevant to determine ℓ. Hence, $\vec{L}$ may contain irrelevant variables to predict ℓ. To decrease computational

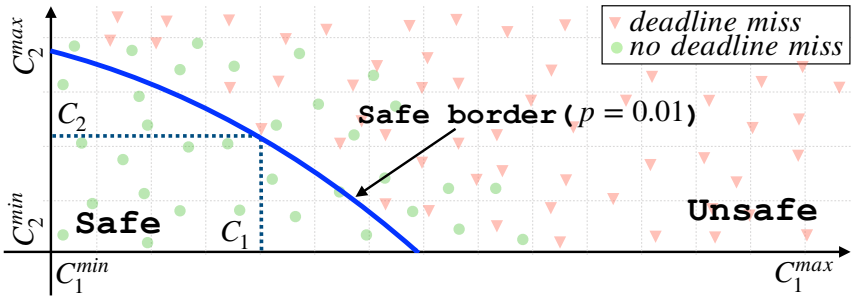


Figure 3.3: A safe border line of WCET values for the τ_1 and τ_2 tasks. The safe border is determined by a deadline miss probability of 0.01. C_1 and C_2 determine safe WCET ranges of τ_1 and τ_2 under which they likely satisfy their deadlines.

complexity for the remaining steps, SafeRefinement creates a reduced dataset $\vec{L}^r$ which contains the same number of data instances (tuples) as $\vec{L}$ while including only WCET values with a significant effect on ℓ. To that end, SafeRefinement employs a standard feature reduction technique: random forest feature reduction [35] which has been successfully applied to high-dimensional data [141, 96]. Given the labeled dataset $\vec{L}$, random forest creates a set of decision trees based on the parameter values such as the number of trees and tree depth. Decision trees obtained by random forest allow us to rank features, i.e., task WCETs, based on their importance as measured by Gini impurity [35]. Hence, by setting a particular threshold for importance, we can select a subset of the features. Note that Section 3.5.6 describes the parameter values for the feature reduction step in detail.

After reducing the dimensionality of the input dataset $\vec{L}$ in Algorithm 3.1, resulting in the reduced dataset $\vec{L}^r$, SafeRefinement learns an initial model to predict safe WCET ranges. SafeRefinement uses logistic regression [100] because it enables a probabilistic interpretation of safe WCET ranges and the investigation of relationships among different tasks' WCETs. For example, Figure 3.3 shows a *safe border* determined by an inferred logistic regression model m with a probability p of deadline misses. Note that a safe range, e.g., $[C_1^{min}, C_1]$ of task τ_1 in Figure 3.3, is determined by a point on the safe border in a multidimensional WCET space. A safe border distinguishes safe and unsafe areas in the WCET space. After inferring a logistic regression model m from the input dataset, SafeRefinement selects a probability p maximizing the safe area under the safe border determined by m and p while ensuring that all the data instances, i.e., sets of WCET values, classified as safe using the safe border are actually observed to be safe in the input dataset, i.e., no false positives (lines 3–4). We note that engineers can also select an adequate probability, which may yield false positives or not maximize the area under the safe border, depending on their needs.

SafeRefinement uses a second-order polynomial response surface model (RSM) [135] to build a logistic regression model. RSM is known to be useful when the relationship between several explanatory variables (e.g., WCET variables) and one or more response variables (e.g., safe or unsafe label) needs to be investigated [135, 124]. RSM contains linear terms, quadratic terms, and 2-way interactions between linear terms. Let V be a set of WCET variables v_x in $\vec{L}^r$. Then, the logistic regression model of SafeRefinement is defined as follows:

$$\log \frac{p}{1-p} = c_0 + \sum_{x=1}^{|V|} c_x v_x + \sum_{x=1}^{|V|} c_{xx} v_x^2 + \sum_{y>x} c_{xy} v_x v_y$$

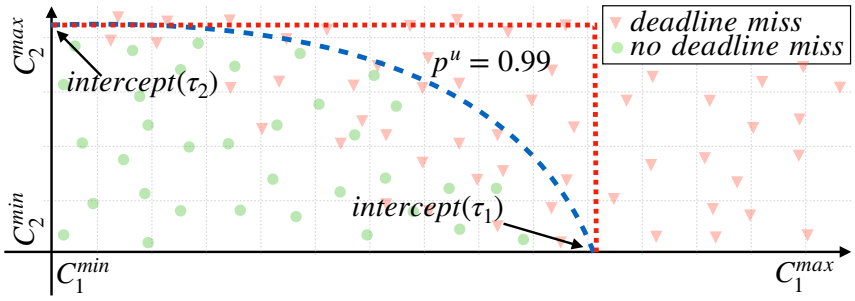


Figure 3.4: Handling imbalanced dataset by excluding unsafe WCET values based on logistic regression intercepts.

As shown in the above equation, an RSM equation, i.e., the right-hand side, built on the reduced dataset $\vec{L}^r$ has a higher number of dimensions, i.e., the number of coefficients to be inferred, than $|V|$ as RSM additionally accounts for quadratic terms (v_x^2) and 2-way interactions ($v_x v_y$) between linear terms. Hence, SafeRefinement employs a stepwise regression technique (line 3), e.g., stepwise AIC (Akaike Information Criterion) [191], in order to select significant explanatory terms from the RSM equation. This allows the remaining "refine model" step of SafeRefinement to execute efficiently as it requires to run SafeScheduler and logistic regression multiple times within a time budget (line 8), both operations being computationally expensive.

Imbalance handling. Recall from Section 3.4.1 that SAFE searches for worst-case sequences of task arrivals and is guided by maximizing the magnitude of deadline misses, when they are possible. Therefore, the major portion of $\vec{L}$, the dataset produced by the first phase of SAFE, is a set of task arrival sequences leading to deadline misses. Supervised machine learning techniques (including logistic regression) typically produce unsatisfactory results when faced with highly imbalanced datasets [24]. SafeRefinement addresses this problem with the "handle imbalanced dataset" step in Algorithm 3.1 (lines 5–6) before refining safe WCET ranges. SafeRefinement aims to identify WCET ranges under which tasks are likely to be schedulable. This entails that WCET ranges under which tasks are highly unlikely to be schedulable can be safely excluded from the remaining analysis. Specifically, SafeRefinement prunes out WCET ranges with a high probability of deadline misses above a high threshold p^u and thus creates a more balanced dataset $\vec{L}^b$ compared to the original imbalanced dataset $\vec{L}^r$ (line 6). SafeRefinement automatically finds a minimum probability p^u which leads to a safe border classifying no false unsafe (negative) instances in $\vec{L}^r$. SafeRefinement then updates the maximum WCET C_i^{max} of a task τ_i based on the intercept of the logistic regression model m (with a probability of p^u) on the WCET axis for τ_i. Figure 3.4 shows an example dataset $\vec{L}^r$ with a safe border characterised by a high deadline miss probability, i.e., $p^u = 0.99$, to create a more balanced dataset $\vec{L}^b$ within the restricted ranges $[C_1^{min}, intercept(\tau_1)]$ and $[C_2^{min}, intercept(\tau_2)]$.

Model refinement. The "refine model" step in Algorithm 3.1 refines an inferred logistic regression model by sampling additional schedule scenarios selected according to a strategy that is expected to improve the model. As described in Section 3.4.1, the SAFE search produces a set G (population) of worst-case arrival sequences of tasks Γ which likely violate deadline constraints of target tasks $\Gamma^\delta \subseteq \Gamma$. For each arrival sequence A in G, SafeRefinement executes SafeScheduler ns times to add ns new data instances to the dataset $\vec{L}^b$ based on the generated schedule scenarios and their schedulability results (lines 9–19). After adding $ns \cdot |G|$ new data instances to $\vec{L}^b$, SafeRefinement runs logistic regression again to infer a refined logistic regression model m and computes a probability p that ensures no false

safe instances (positives) in $\vec{L}^b$ and maximizes the safe area under the safe border defined by m and p (lines 20–25).

In the second phase of SAFE, SafeScheduler selects WCET values for tasks in Γ to compute a schedule scenario based on a distance-based random number generator, which extends the standard uniform random number generator. The distance-based WCET value sampling aims at minimizing the Euclidean distance between the sampled WCET points and the safe border defined by the inferred model m and the selected probability p. SafeScheduler iteratively computes new WCET values using the following distance-based sampling procedure: (1) generating nr random samples in the WCET space, (2) computing their distance values from the safe border, and (3) selecting the closest point to the safe border.

SafeRefinement stops model refinements either by reaching an allotted analysis budget (line 8 of Algorithm 3.1) or when a precision reaches an acceptable level pt, e.g., 0.99 (lines 23–25). SafeRefinement uses the standard precision metric [187] as described in Section 3.5.5. In our context, practitioners need to identify safe WCET ranges at a high level of precision to ensure that identified safe WCET ranges can be trusted. To compute a precision value, SafeRefinement uses a standard k-fold cross-validation [187]. In k-fold cross-validation, $\vec{L}^b$ is partitioned into k equal-size splits. One split is retained as a test dataset, and the remaining k-1 splits are used as a training dataset. The cross-validation process is then repeated k times to compute a precision of inferred safe borders which are determined by a logistic regression model m and a probability p (lines 21 and 22)

Selecting WCET ranges. A safe border defined by an inferred logistic regression model and a deadline miss probability of p represents a (possibly infinite) set of points, corresponding to safe WCET ranges of tasks, e.g., $[C_1^{min}, C_1]$ and $[C_2^{min}, C_2]$ in Figure 3.3. In practice, however, engineers need to choose a specific WCET range for each task to conduct further analysis and development. How to choose optimal WCET ranges depends on the system context. At early stages, however, such contextual information may not be available. Hence, SAFE proposes a *best-size point*, i.e., WCET ranges, on a safe border which maximizes the volume of the hyperbox the point defines. In general, the larger hyperbox, the greater flexibility the engineers have in selecting appropriate WCET values. Choosing the point with the largest volume is helpful when no domain-specific information is available to define other selection criteria. In general the inferred safe border enables engineers to investigate trade-off among different tasks' WCET values.

3.5 Evaluation

We evaluate SAFE using an industrial case study from the satellite domain. Our full evaluation package is available online [112].

3.5.1 Research questions

RQ1 (baseline comparison): *How does SAFE perform compared with a baseline approach?* With RQ1, we investigate whether SAFE can outperform WCET estimation based on random search. Note that such RQ is an important *sanity check* for search-based solutions in general [89, 12]. Our conjecture is that SAFE, although computationally expensive, will significantly outperform a random search solution with respect to estimating safe WCET ranges with a higher degree of confidence.

Table 3.2: Description of the three industrial subject systems: number of periodic and aperiodic tasks, resource dependencies, and platform cores. The full task descriptions are available online [112].

System	Periodic tasks	Aperiodic tasks	Dependencies	Cores
ADCS	15	19	0	1
ICS	3	3	3	3
UAV	12	4	4	3

RQ2 (effectiveness of distance-based sampling): *How does SAFE, based on distance-based sampling, perform compared with random sampling?* We compare our distance-based sampling procedure described in Section 3.4.2 and used in the second phase of SAFE with a naive random sampling. Our conjecture is that distance-based sampling, although expensive, is needed to improve the quality of the training data used for logistic regression. RQ2 assesses this conjecture by comparing distance-based and random sampling.

RQ3 (usefulness): *Can SAFE identify WCET ranges within which tasks are highly likely to satisfy their deadline constraints?* In RQ3, we investigate whether SAFE identifies acceptably safe WCET ranges in practical time. We further discuss our insights regarding the usefulness of SAFE from the feedback obtained from engineers in LuxSpace.

RQ4 (scalability): *Can SAFE find safe WCET ranges for large-scale systems with a practical time budget?* In this RQ, we study the relationship between the execution time of SAFE and the parameters of study subjects. We use several synthetic subjects to be able to freely control key real-time systems' parameters.

3.5.2 Industrial study subjects

We evaluated SAFE by applying it to our motivating case study subject, i.e., the satellite attitude determination and control system (ADCS) described in Section 3.2, as well as two industrial study subjects from the literature [145, 171]. Table 3.2 summarizes the relevant attributes of these subjects, presenting the number of periodic and aperiodic tasks, resource dependencies, and processing cores. The subjects are characterized by real-time parameters, e.g., priorities, WCETs, periods, and deadlines, described in Section 3.3. The full task descriptions of the subjects are available online [112]. The main missions of the three subjects are described as follows:

- ADCS is a satellite system that aims at orienting a satellite in a proper position on time to ensure that the satellite provides normal service correctly (see Section 3.2). LuxSpace, our industry partner, developed ADCS for an ESA project.

- ICS is an ignition control system that checks the status of an automotive engine and corrects any errors of the engine [145]. The system was developed by Bosch GmbH[1].

[1] Bosch GmbH: https://www.bosch.com/

- UAV is a mini unmanned air vehicle that follows dynamically defined way-points and communicates with a ground station to receive instructions [171]. The system was developed in a collaboration with the University of Poitiers France and ENSMA[2].

LuxSpace is a leading system integrator of micro satellites and aerospace systems. ADCS includes a set of 15 periodic and 19 aperiodic tasks. Eight tasks out of the 19 aperiodic tasks are constrained by hard deadlines, i.e., sporadic tasks. Out of the 34 tasks, engineers provided single WCET values for eight tasks. For the remaining 26 tasks, engineers estimated WCET ranges due to uncertain decisions, e.g., implementation choices and hardware specifications, made at later development stages (see Section 3.2). The differences between the estimated WCET maximum and minimum values across the 26 tasks vary from 0.1ms to 20000ms. Our collaboration with LuxSpace enabled us to discuss SAFE results with engineers to draw important qualitative conclusions and to assess the benefits of SAFE (see Section 3.5.7).

For the experiments with ICS and UAV, we used the task descriptions reported in a previous study [64] and modified their tasks' WCETs from point values to ranges. Though the problem of schedulability analysis of real-time tasks has been widely studied [64, 7, 87, 33, 175], none of the prior work addresses the same problem (see Section 3.3) as that addressed by SAFE. Hence, the public study subjects in the literature do not fit our study's requirements. In particular, none of the public real-time system case studies [64] contains estimated WCET ranges in their task descriptions. These ranges, however, are necessary to apply SAFE and to evaluate its effectiveness. In order to evaluate SAFE in various and realistic system contexts, we chose to apply SAFE to existing industrial subjects, i.e., ICS and UAV, described in prior work [64] and made necessary changes only to task WCETs of the subjects as described below. Compared to ADCS, ICS and UAV have different task characteristics, such as resource dependencies and number of processing cores.

We note that estimating (practically valid) WCET ranges requires significant domain expertise. For public domain case study systems such as ICS and UAV, however, we do not have any access to the engineers who have developed those subjects. Hence, we chose to apply a simple and straightforward method to convert a point WCET value to a WCET range as follows: (Step 1) We first check whether the system under analysis is schedulable or not. For a task τ_i in the system, we denote by C_i an original point WCET value of τ_i. (Step 2) If the system is evaluated to be schedulable, it indicates that the system's tasks may be able to handle higher execution times than their estimated WCETs. Hence, we simply define the WCET range of τ_i by $[C_i,\ r{\cdot}C_i]$, where $r > 1$, as input WCET ranges for SAFE. This modification enables SAFE to find more relaxed safe WCET ranges. (Step 2′) Otherwise, if the system is evaluated to be unschedulable, we define the WCET range of τ_i by $[r'{\cdot}C_i,\ C_i]$, where $r' < 1$, as input for SAFE. This modification allows SAFE to find appropriate WCET estimates, ensuring the system is likely to be schedulable under the WCET ranges found by SAFE. As ICS and UAV are likely to be schedulable [64], for all task τ_i in ICS and task τ_j in UAV, we created the modified task descriptions of ICS and UAV based on Step 2. We conducted experiments using simulations to set the r values for ICS and UAV by configuring r to 1.1, 1.2, . . ., 1.5 incrementally until we could find deadline misses in each system, i.e., unsafe WCET values. Recall from Section 3.4.2 that SAFE relies on logistic regression to partition the given WCET ranges into safe and unsafe sub-ranges for a selected deadline miss probability. Hence, we modified the estimated WCET ranges of ICS and UAV to include both safe and unsafe WCET ranges.

[2]ENSMA: https://www.ensma.fr/

Algorithm 3.2: An algorithm for creating a synthetic subject while accounting for the task characteristics described in Section 3.3.

Input: - n: number of tasks
- u^t: target utilisation
- T^{min}: minimum task period
- T^{max}: maximum task period
- g: granularity of task periods
- θ: maximum offset value
- γ: ratio of aperiodic tasks
- μ: range factor to determine inter-arrival times
- ω: number of WCET ranges
- λ: range factor to determine WCET ranges

Output: - Γ: set of tasks

1: $\Gamma \leftarrow \{\}, \mathbf{C} \leftarrow \{\}$
2: //synthesise a set of periodic tasks
3: $\mathbf{U} \leftarrow$ `UUniFast_discard`(n, u^t)
4: $\mathbf{T} \leftarrow$ `generate_task_periods`(n, T^{min}, T^{max}, g) //task periods
5: **for** each $i \in [1, n]$ **do**
6: $\quad \mathbf{C} \leftarrow \mathbf{C} \cup \{U_i \cdot T_i\}$, where $U_i \in \mathbf{U}$ and $T_i \in \mathbf{T}$ //WCETs
7: **end for**
8: $\Gamma \leftarrow$ `generate_task_set`$(\mathbf{T}, \mathbf{C}, \theta, g)$
9: //convert some periodic tasks to aperiodic tasks
10: $\Gamma \leftarrow$ `convert_to_aperiodic_tasks`(Γ, γ, μ)
11: //convert some WCET point values to WCET ranges
12: $\Gamma \leftarrow$ `convert_to_WCET_ranges`$(\Gamma, \omega, \lambda)$
13: **return** Γ

Given the experiment results, we set the WCET ranges of ICS and UAV to $[C_i, 1.2 \cdot C_i]$ and $[C_j, 1.5 \cdot C_j]$, respectively. The full original and modified task descriptions of ICS and UAV are available online [112].

3.5.3 Synthetic study subjects

We evaluated the scalability of SAFE using synthetic systems, following the common scalability analysis practice applied in many real-time system studies [62, 195, 59, 82, 176, 68]. As shown in Algorithm 3.2, we synthesise a set of real-time tasks by varying key task parameters as described below. The algorithm first synthesises a set of periodic tasks (lines 2-8) and then converts some of these tasks to aperiodic tasks (lines 9-10). Last, the algorithm configures some tasks with WCET ranges (lines 11-12).

As shown on line 3 of Algorithm 3.2, the algorithm first creates a set $\mathbf{U}$ of task utilisation values by using the UUniFast-Discard algorithm [59] that is devised to give an unbiased distribution of task utilisation values. The UUniFast-Discard algorithm takes as input the number of tasks to be synthesised, n, and a target utilisation value, u^t. It then outputs n utilization values, $U_1 \ldots U_n$, where $0 < U_i < 1$ for all U_i and $\sum_{i=1}^{n} U_i = u^t$.

As for line 4 in Algorithm 3.2, the algorithm generates n task periods, $T_1 \ldots T_n$ according to a log-uniform distribution within a range $[T^{min}, T^{max}]$, i.e., given a task period (random variable) T_i, $log\ T_i$ follows a uniform distribution. For example, when a period range $[T^{min}, T^{max}]$ is [10ms, 1000ms], the algorithm generates approximately an equal number of tasks in period ranges [10ms, 100ms] and [100ms,

1000ms]. The parameter g is used to determine the granularity of period values as multiples of g. Lines 5-7 of Algorithm 3.2 describe how the algorithm synthesises tasks' WCET values. Specifically, for each task τ_i, the algorithm computes the WCET value C_i of τ_i as $C_i = U_i \cdot T_i$.

Given the task periods **T** and the WCET values **C**, line 8 of Algorithm 3.2 synthesizes a set Γ of periodic tasks accounting for offsets, priorities, and deadlines. A periodic task τ_i is characterised by a period T_i, a WCET C_i, an offset O_i, a priority P_i, and a deadline D_i (see Section 3.3). A task offset O_i is randomly selected from an input range $[0, \theta]$ of offset values. The algorithm relies on the rate-monotonic scheduling policy [118] to decide task priorities and deadlines. Specifically, tasks with shorter periods are given higher priorities and tasks' deadlines are equal to their periods.

Line 10 of Algorithm 3.2 synthesises aperiodic tasks. The algorithm converts some periodic tasks into aperiodic tasks according to a ratio γ of aperiodic tasks among all tasks. The algorithm then uses a range factor μ to determine minimum and maximum inter-arrival times of aperiodic tasks. Specifically, for a task τ_i to be converted, the algorithm computes a range $[T_i^{min}, T_i^{max}]$ of inter-arrival times as $[T_i^{min}, T_i^{max}] = [T_i \times (1 - \mu), T_i \times (1 + \mu)]$, where $\mu \in (0, 1)$. For example, if $\mu = 0.45$ and $T_i = 50$ for a task τ_i to be converted, $[T_i^{min}, T_i^{max}] = [27.5, 72.5]$.

To synthesise tasks' WCET ranges, line 12 of Algorithm 3.2 randomly selects ω tasks in Γ to convert their WCET point values to WCET ranges. For a selected task τ_i, the algorithm computes a WCET range $[C_i^{min}, C_i^{max}]$ as $[C_i^{min}, C_i^{max}] = [C_i \times (1 - \lambda), C_i \times (1 + \lambda)]$, where λ is a range factor to determine the WCET ranges and $\lambda \in (0, 1)$. For example, if $\lambda = 0.25$ and $C_i = 10$ for a task τ_i, $[C_i^{min}, C_i^{max}] = [7.5, 12.5]$.

3.5.4 Experimental setup

To answer RQ1, RQ2, and RQ3 described in Section 3.5.1, we rely on case study data pertaining to ADCS, provided by LuxSpace, as well as the ICS and UAV subjects described in Section 3.5.2. To answer RQ4, we used 800 synthetic subjects (see Section 3.5.3). We conducted four experiments, EXP1, EXP2, EXP3, and EXP4, as described below.

EXP1. To answer RQ1, we developed a baseline solution that estimates task WCETs based on random search (RS). The baseline replaces the GA in Phase 1 with RS and does not infer a safe border using logistic regression. Note that the baseline uses the same fitness function (see Section 3.4.1) and also maintains the best population during search; however, it does not employ any genetic operators, i.e., crossover and mutation. The baseline solution also produces a labeled dataset $\vec{L}$ that contains tuples (W, ℓ) where W is a set of task WCETs and ℓ is a label of W indicating either safe or unsafe (see Section 3.4.1). Given the labeled dataset, the baseline selects the best task WCETs that are safe and maximize the volume of the hyperbox they define. Specifically, the baseline finds a particular tuple (W_s, ℓ_s) in $\vec{L}$ that maximizes the volume of the hyperbox defined by W_s while satisfying the following condition: For all tuples (W_x, ℓ_x) in $\vec{L}$, the hyperbox defined by W_x is contained in the hyperbox defined by W_s, $\ell_x = safe$, and $\ell_s = safe$.

EXP1 compares the results obtained from executing SAFE and the baseline. For comparison, SAFE selects a best-size point, i.e., WCET ranges, on a safe border that maximizes the volume of the hyperbox the point defines (see Section 3.4.2). Given two solutions, i.e., estimated WCET ranges, obtained by SAFE and the baseline, EXP1 checks the schedulability of the two solutions using simulations. To do so, we ran simulations multiple times by varying task arrivals and task execution times within their estimated WCET ranges and checked whether there was a deadline miss in each simulation result.

EXP2. To answer RQ2, EXP2 compares our distance-based WCET sampling technique (described in Section 3.4.2) with the naive random WCET sampling technique, for the second phase of SAFE. To this end, EXP2 first creates an initial training dataset by running the first phase of SAFE. EXP2 then relies on this initial training data for model refinement (Section 3.4.2) by using both distance-based and naive random sampling. For comparison, EXP2 creates a test dataset by randomly sampling WCET values, which is independently created from the second phase of SAFE, and then compares the accuracy of the two sampling approaches in identifying safe WCET ranges for the test dataset.

EXP3. To answer RQ3, EXP3 computes precision values for SAFE, obtained from 10-fold cross-validation (see Section 3.4.2), over each model refinement for the ADCS subject. We note that EXP3 focuses on ADCS to evaluate the practical usefulness of SAFE as we do not have any access to the engineers who have developed the other study subjects, i.e., ICS and UAV. In our study context, i.e., developing safety-critical systems, engineers require very high precision, i.e., ideally no false positives, (see Section 3.4). Hence, EXP3 measures precision over model refinements to align with such practice. EXP3 then measures whether SAFE can compute safe WCET ranges within practical execution time and at an acceptable level of precision.

EXP4. To answer RQ4, EXP4 measures the execution time of SAFE with 800 synthetic systems. We use the task generation algorithm described in Section 3.5.3 to create synthetic systems. In order to conduct controlled experiments to study correlations between the execution time of SAFE and a particular system parameter (e.g., number of tasks), we first create a baseline synthetic system by setting the parameters of Algorithm 3.2 as follows: (1) We set the number of tasks n to 20, the ratio of aperiodic tasks γ to 0.45, and the maximum offset θ to 0. Note that these parameter values are the average values of the parameters in our industrial subjects. (2) With regard to task periods, we set the range $[T^{min}, T^{max}]$ of minimum and maximum periods to [10ms, 1s], which are common values in many real-time subjects [23]. The granularity of task periods g is set 10ms in order to increase realism as most of the task periods in our industrial subjects are multiples of 10ms. (3) For the range factor to determine inter-arrival times for aperiodic tasks μ, the number of WCET ranges ω, the range factor to determine WCET ranges λ, and the target utilisation per processing core u^t, we assign $\mu = 0.25$, ω=2, $\lambda = 0.25$, and $u^t = 0.9$, respectively. We set these parameter values based on initial experiments to ensure that the executions of the synthetic systems examined in EXP4 sometimes violate their deadlines. Recall from Section 3.4 that SAFE relies on logistic regression and a labeled dataset, containing both safe (positive) and unsafe (negative) data instances. (4) We set the number of processing cores ϵ to 1 as a baseline. (5) For the simulation time of SafeScheduler (see Section 3.4.1), we assign 30s in order to ensure that any aperiodic task arrives at least once and all possible arrivals of periodic tasks are analyzed during that time.

Given the baseline system, we create several synthetic systems to be examined in EXP4 by varying the parameters' values as follows: (1) number of tasks, $n \in \{5, 10, ..., 50\}$, (2) ratio of aperiodic tasks, $\gamma \in \{0.05, 0.1, ..., 0.5\}$, (3) range factor to determine inter-arrival times for aperiodic tasks, $\mu \in \{0.05, 0.1, ..., 0.5\}$, (4) number of WCET ranges, $\omega \in \{1, 2, ..., 10\}$, (5) range factor to determine WCET ranges, $\lambda \in \{0.05, 0.1, ..., 0.5\}$, (6) maximum offset value $\theta \in \{200ms, 400ms, ..., 2000ms\}$, (7) number of processing cores, $\epsilon \in \{1, 2, ..., 10\}$, and (8) simulation time, $\mathbf{t} \in \{30s, 1m, ..., 5m\}$. Note that we chose to study the effect of these parameters because they are controlled by engineers to design tasks in real-time systems. Simulation time $\mathbf{t}$ obviously impacts the execution time of SAFE as well. Resource dependencies are not controlled when generating synthetic systems as they do not impact SAFE's searching and learning

(see Section 3.4) but only simulations, which are investigated by varying simulation time. Due to the degree of randomness in our approach to generating synthetic systems (see Section 3.5.3), we create ten synthetic systems for each control parameter.

3.5.5 Metrics

We use the standard precision and recall metrics [187] to measure the accuracy in our experiments. To compute precision and recall in our context, for EXP2, we created a synthetic test dataset for each study subject containing tuples of WCET values and a flag indicating the presence or absence of deadline miss obtained from running SafeScheduler. Note that creating a test dataset by running an actual study subject with varying task WCETs is prohibitively expensive. We therefore used a set of task arrival sequences obtained from the first phase of SAFE for each subject as we aim at testing sequences of task arrivals which are more likely to violate their deadlines. We then ran SafeScheduler to simulate task executions for the set of task arrival sequences with randomly sampled WCET values. We note that WCET values were sampled within the restricted WCET ranges after the "handling imbalance" step in Algorithm 3.1. Parts of the WCET ranges under which tasks are unlikely to be schedulable are therefore not considered when sampling. For EXP3, we used 10-fold cross-validation based on the training dataset at each model refinement step (phase 2).

We define the precision and recall metrics as follows: (1) precision $P = TP/(TP + FP)$ and (2) recall $R = TP/(TP + FN)$, where TP, FP, and FN denote the number of true positives, false positives, and false negatives, respectively. A true positive is a test instance (a set of WCET values) labeled as safe and correctly classified as such. A false positive is a test instance labeled as unsafe but incorrectly classified as safe. A false negative is a test instance labeled as safe but incorrectly classified as unsafe. We prioritize precision over recall as practitioners require (ideally) no false positives – an unsafe instance with deadline misses is incorrectly classified as safe – in the context of mission-critical, real-time satellite systems. For EXP2, precision and recall values are measured based on a synthetic test dataset. For EXP3, precision values are computed using collective sets of true positives and false positives obtained from 10-fold cross-validation at each model refinement.

Due to the inherent degree of randomness in SAFE, we repeat our experiments 50 times. For EXP1, we ran 40000 simulations to check the schedulability of the solutions obtained by SAFE and the baseline. To statistically compare our results, we use the non-parametric Mann-Whitney U-test [125] and Vargha and Delaney's $\hat{A}_{12}$ effect size [172]. Mann-Whitney U-test determines whether two independent samples are likely or not to belong to the same distribution. We set the level of significance, α, to 0.05. Vargha and Delaney's $\hat{A}_{12}$ measures probabilistic superiority – effect size – between search algorithms. Two algorithms are considered to be equivalent when the value of $\hat{A}_{12}$ is 0.5.

3.5.6 Implementation and parameter tuning

To implement the feature reduction step of Algorithm 3.1, we used the random forest feature reduction [35] as it has been successfully applied to high-dimensional data [141, 96]. For the stepwise regression step of Algorithm 3.1, we used the stepwise AIC regression technique [191] which has been used in many applications [196, 129]. Recall from Section 3.4.2 that our distance-based sampling and best-size region recommendation require a numerical optimization technique to find the nearest WCET sample and a

maximum safe region size based on an inferred safe border. For such optimizations, we applied a standard numerical optimization method, i.e., the Nelder-Mead method [140].

To compute the GA fitness, we set the number of SafeScheduler runs (Section 3.4.1) for each solution (A in Section 3.4.1) to 20. This number was chosen based on our initial experiments. We observed that 20 runs of SafeScheduler per solution A keeps execution time under a reasonable threshold, i.e., <1.2m for all the subjects, and is sufficient to compute the fitness of SAFE. SafeScheduler schedules 34 tasks in ADCS for 1800s, 6 tasks in ICS for 150ms, and 16 tasks in UAV for 1500ms during which SafeScheduler advances its simulation clock by 0.1ms, 0.01ms, 0.01ms, respectively, for adequate precision. We chose the time periods to ensure that all the tasks in each subject can be executed at least once.

For the GA search parameters, we set the population size to 10, the crossover rate to 0.7, and the mutation rate to 0.2, which are consistent with existing guidelines [93]. We ran GA for 1000 iterations after which we observed that fitness reached a plateau in our initial experiments. Note that for the baseline comparison to be fair, we ran RS for 1500 iterations to ensure that the generated dataset $\vec{L}$ contained 30000 data instances, which are the same number of data instances obtained by SAFE.

Regarding the feature reduction step of Algorithm 3.1, we set the random forest parameters as follows: (1) the tree depth parameter is set to $\sqrt{|F|}$, where $|F|$ denotes the number of features, i.e., 26 WCET ranges in ADCS, 6 WCET ranges in ICS, and 16 WCET ranges in UAV, based on guidelines [91]. (2) The number of trees is set to 100 based on our initial experiments. We observed that learning more than 100 trees does not provide additional gains in terms of reducing the number of features.

Note that all the parameters mentioned above can probably be further tuned to improve the performance of SAFE. However, since with our current setting, we were able to convincingly and clearly support our conclusions, we do not report further experiments on tuning those parameters.

We ran our experiments over the high-performance computing cluster [173] at the University of Luxembourg. To account for randomness, we repeated each run of SAFE 50 times for all the experiments. Each run of SAFE was executed on a different node of the cluster. It took around 35h for us to create a synthetic test dataset with 50000 instances. When we set 1000 GA iterations for the first phase of SAFE and 10000 new WCET samples (100 refinements × 100 new WCET samples per refinement) for the second step of SAFE, each run of SAFE took at most 27.1h – phase 1: 16.361h and phase 2: 10.74h. The running time is acceptable as SAFE can be executed offline in practice.

3.5.7 Experiment results

RQ1. Table 3.3 compares SAFE and our baseline method (Baseline) using the following three metrics: (1) the volume of the hyperbox that is defined by the best-size point (see Sections 3.4.2 and 3.5.4) computed by each method, i.e., SAFE and Baseline, (2) the number of simulation runs, out of 40000 runs, that contain any deadline misses when tasks execute within their estimated WCET ranges defined by the best-size points, and (3) the execution time of SAFE and Baseline to estimate WCET ranges. The results presented in the table are the mean values obtained from 50 runs of SAFE and Baseline for each of the three subjects. To enable accurate comparisons, we ran 40000 simulations of each execution of SAFE and Baseline, aiming at evaluating their estimated WCET ranges as described in Section 3.5.4. Statistical comparisons of the results obtained from 50 runs of SAFE and Baseline are summarized using p-values and $\hat{A}_{12}$ values as described in Section 3.5.5.

Table 3.3: Comparing SAFE and our baseline method using (1) the volumes of the hyperboxes that are defined by the best-size points computed by each method, (2) the number of simulation runs that contain deadline misses out of total 40000 runs, and (3) the execution times of each method. The results are obtained from 50 runs of Safe and Baseline.

	SAFE			Baseline			p-value			$\hat{A}_{12}$		
	ADCS	ICS	UAV	ADCS	ICS	UAV	ADCS	ICS	UAV	ADCS	ICS	UAV
Best-size volumes	5.18e-11 ms^{26}	9.55e-01 ms^{6}	1.20e-02 ms^{16}	5.78e-12 ms^{26}	1.10e+00 ms^{6}	2.33e-04 ms^{16}	0.0000	0.0383	0.0000	**1.0000**	0.3796	**1.0000**
Deadline misses	**0.00**	**5.42**	**1.36**	114.92	13.2	32.32	0.0005	0.0023	0.0439	**0.3900**	**0.3394**	**0.04084**
Execution times	25.14 hours	1.37 hours	1.62 hours	24.29 hours	0.11 hours	0.31 hours	0.0000	0.0000	0.0000	0.8960	1.0000	1.0000

As shown in Table 3.3, compared to Baseline, SAFE provides more relaxed WCET ranges for ADCS and UAV. Note that the larger the hyperbox, the greater flexibility the engineers have in selecting appropriate WCET values. For ICS, however, Baseline finds a larger hyperbox than the best-size hyperbox produced by SAFE. This is likely due to the fact that ICS has a small number of tasks and is therefore much simpler than the other two subjects. Further, we recall that SAFE also provides engineers with a probability of deadline misses and trade-off relations between task WCETs based on an inferred logistic regression model (see Section 3.4.2). We further discuss these benefits from a practitioner's perspective in RQ3.

EXP1 evaluates the estimated WCET ranges that are defined by the best-size points obtained from 50 runs of SAFE and Baseline for each subject. The estimated WCET ranges are examined through 40000 simulation runs by varying task arrivals and their execution times within the estimated WCET ranges. The "Deadline misses" row in Table 3.3 shows the mean number of simulation runs (out of 40000 runs) containing deadline misses. Across all the subjects, the differences between SAFE and Baseline are statistically significant as the p-values are less than 0.05. The $\hat{A}_{12}$ values show that SAFE is probabilistically superior to Baseline with respect to minimizing the number of deadline misses.

Regarding the execution times of SAFE and Baseline, SAFE took more time than Baseline for all the subjects as shown in Table 3.3. Estimating safe WCET ranges for ADCS requires the largest execution time (on average, 25.14h) compared to the other subjects. We note that such execution time is acceptable as SAFE can be executed offline in practice.

Figure 3.5 shows probability distributions obtained from 50 runs of SAFE and simulations for ADCS, ICS, and UAV. As described in Section 3.4.2, SAFE partitions the given WCET ranges into safe and unsafe sub-ranges using a safe WCET border that is defined by an inferred logistic regression model and a selected probability of deadline misses. In EXP1, SAFE selects a deadline miss probability that maximizes the safe area under the safe border while ensuring that all the WCET points, i.e., sets of WCETs, classified as safe using the safe border are actually observed to be safe in the input dataset of logistic regression. The estimated WCET ranges, i.e., 50 best-size WCET points obtained by SAFE, are then evaluated through 40000 simulation runs for each WCET point by varying task arrivals and their execution time within their estimated WCET ranges. The empirical probability, i.e., relative frequency, of deadline misses is computed by the ratio of the number of simulation runs containing any deadline misses to the total number of simulation runs, i.e., 40000. The probability comparison depicted in Figure 3.5 shows that the selected probability of deadline misses by SAFE is larger than the empirical probability computed

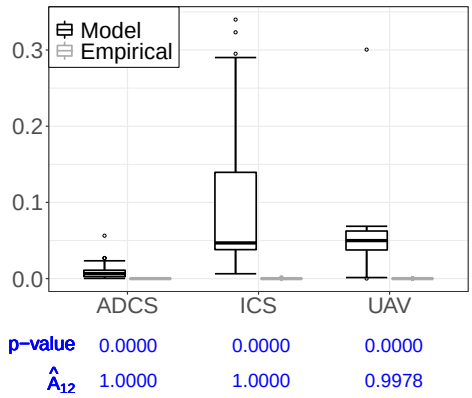


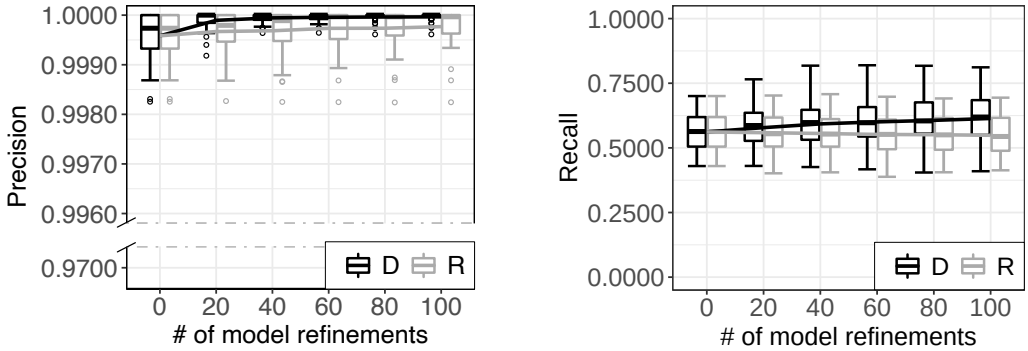
Figure 3.5: Comparing probability values of deadline misses computed by SAFE and simulations for ADCS, ICS and UAV. SAFE uses a logistic regression model and selects a deadline miss probability to find the best-size WCET point. Empirical probability values of deadline misses are estimated based on 40000 simulations for each output, i.e., WCET ranges, of SAFE. The boxplots (20%-50%-75%) show probability values obtained from 50 runs of SAFE (see Model) and 50 simulation-based evaluations (see Empirical), i.e., 50×40000 simulation runs.

by simulation-based evaluations. SAFE infers a logistic regression model, providing a probabilistic interpretation, based on a labeled dataset evaluated by worst-case task arrivals. The inferred logistic regression model, therefore, likely fits the system executions when task arrivals are worst with respect to maximizing the magnitude of deadline misses. However, the empirical probability estimated through simulations is based on system executions when tasks randomly arrive within their inter-arrival time ranges. Hence, a logistic regression model obtained by SAFE enables more conservative probabilistic interpretations of the estimated WCET ranges than simulation-based evaluations for the WCET ranges. This implies that actual deadline miss probabilities tend to be lower than SAFE probability estimates, which is in practice a desirable property.

The answer to **RQ1** *is that* SAFE significantly outperforms the baseline method with respect to minimizing the number of deadline misses when using the estimated WCET ranges. Across our experiments, SAFE takes at most 27.1h (while the baseline takes 26.4h) to estimate the best-size WCET ranges. The execution time is acceptable as SAFE can be executed offline in practice.

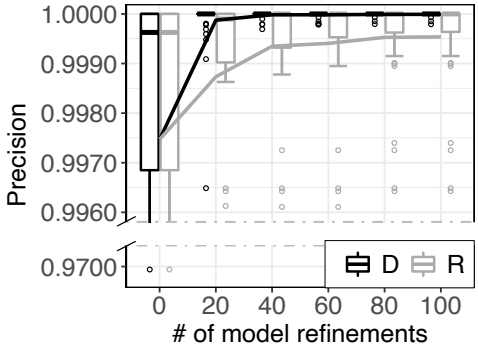
RQ2. Figure 3.6 depicts distributions of precision (Figures 3.6a, 3.6c, and 3.6e) and recall (Figures 3.6b, 3.6d, and 3.6f) obtained from EXP2 with the ADCS, ICS, and UAV subjects. The boxplots in Figures 3.6a, 3.6c, and 3.6e (resp. Figures 3.6b, 3.6d, and 3.6f) show distributions (25%-50%-75% quantiles) of precision (resp. recall) values obtained from 50 executions of SAFE with either distance-based sampling (D) or simple random sampling (R). The solid lines represent the average trends of precision and recall value changes over 100 regression model refinements.

As shown in Figures 3.6a, 3.6c, and 3.6e, across over 100 model refinements, SAFE with D achieves higher precision values than those obtained by R for the three subjects. Also, Figures 3.6a, 3.6c, and 3.6e show that the variance of precision with D tends to be smaller than that of R. On average, D's precision converges toward 1 with model refinements; however, precision with R shows a markedly different trend that is not converging to 1, an important property in our context. Based on statistical comparisons, the

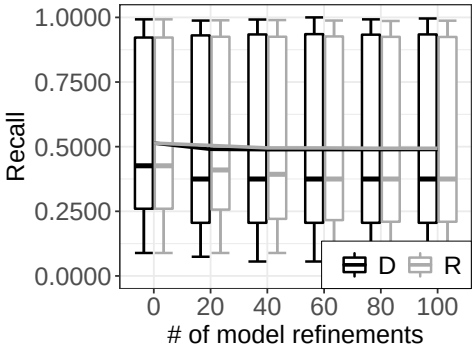


(a) ADCS: Precision

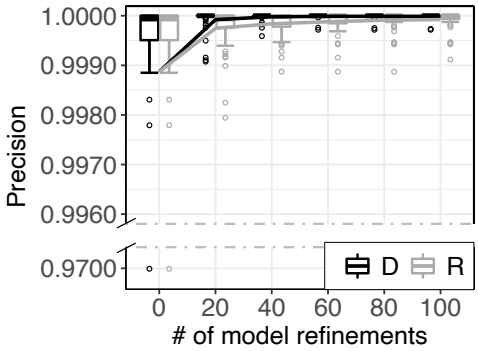
(b) ADCS: Recall



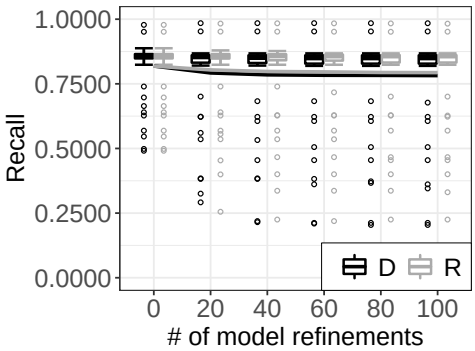
(c) ICS: Precision



(d) ICS: Recall



(e) UAV: Precision



(f) UAV: Recall

Figure 3.6: Distributions of precision and recall over 100 model refinements when SAFE employs either our distance-based sampling (D) or random sampling (R) for (a,b) ADCS, (c,d) ICS, and (e,f) UAV. The boxplots (25%-50%-75%) show precision (a,c,e) and recall (b,d,f) values obtained from 50 runs of SAFE with each sampling strategy. The lines represent average trends.

difference in precision values between D and R becomes statistically significant after only 10, 4, and 11 model refinements, respectively, for ADCS, ICS, and UAV.

Regarding recall comparisons between D and R for ADCS, as shown in Figure 3.6b, D produces higher recall values over 100 model refinements than those of R. The difference in recall values between D and R becomes statistically significant after 36 model refinements. Regarding ICS (Figure 3.6d) and UAV (Figure 3.6f), their differences in recall values for D and R are not statistically significant even after 100 model refinements. This may be explained by their much smaller number of tasks compared with

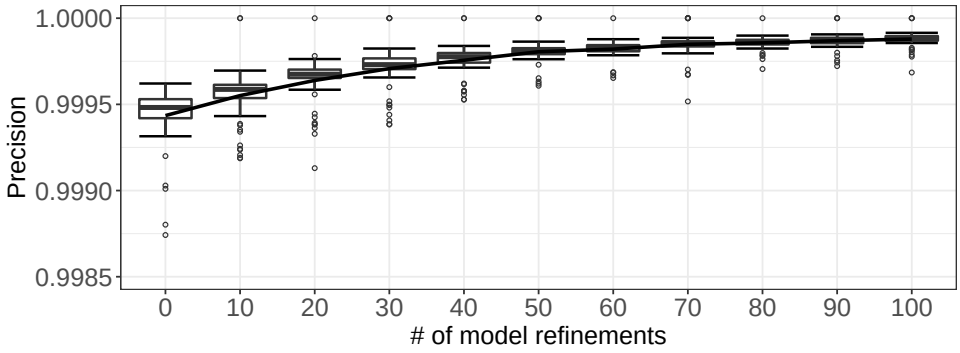


Figure 3.7: Precision values computed from 10-fold cross validation at each model refinement for ADCS. The boxplots (25%-50%-75%) show precision values obtained from 50 runs of SAFE. The line represents an average trend.

ADCS. Across our experiments, for 100 model refinements, SAFE took, at most, 10.86h and 10.54h with D and R, respectively.

The answer to **RQ2** *is that* SAFE with distance-based sampling significantly outperforms SAFE with random sampling in achieving higher precision. Only distance-based sampling can achieve a precision close to 1 within practical time, an important requirement in our context.

RQ3. Figure 3.7 shows precision values obtained from 10-fold cross-validation at each model refinement for the ADCS subject. Recall from Section 3.4.2 that SAFE stops model refinements once a precision value reaches a desired value. As shown in Figure 3.7, precision values tend to increase with additional WCET samples. Hence, practitioners are able to stop the model refinement procedure once precision reaches an acceptable level, e.g., >0.999. At 100 model refinement, SAFE reaches, on average, a precision of 0.99986. For EXP3, SAFE took, at most, 16.36h for phase 1 and 10.74h for phase 2.

As described in Section 3.4.2, SAFE reduces the dimensionality of the WCET space through a feature reduction technique based on random forest. The computed importance scores of each task's WCET in our dataset are as follows: 0.773 for T30, 0.093 for T33, 0.016 for T23, and ≤0.005 for the remaining 31 tasks. Based on a standard feature selection guideline [91], only the WCET values of two tasks, i.e., T30 and T33, are deemed to be important enough to retain as their score is higher than the average importance,

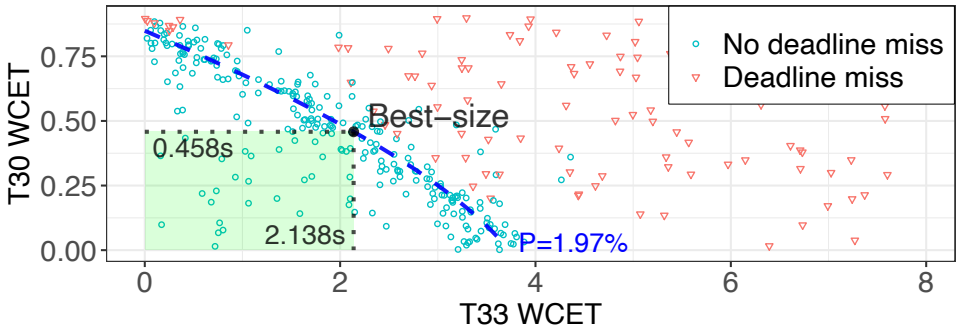


Figure 3.8: An inferred safe border and best-size WCET regions for tasks T30 and T33. The safe border determines WCET ranges within which tasks are likely to be schedulable with a deadline miss probability of 1.97%.

i.e., 0.0385. Hence, SAFE computes safe WCET ranges of these two tasks in the next steps described in Algorithm 3.1.

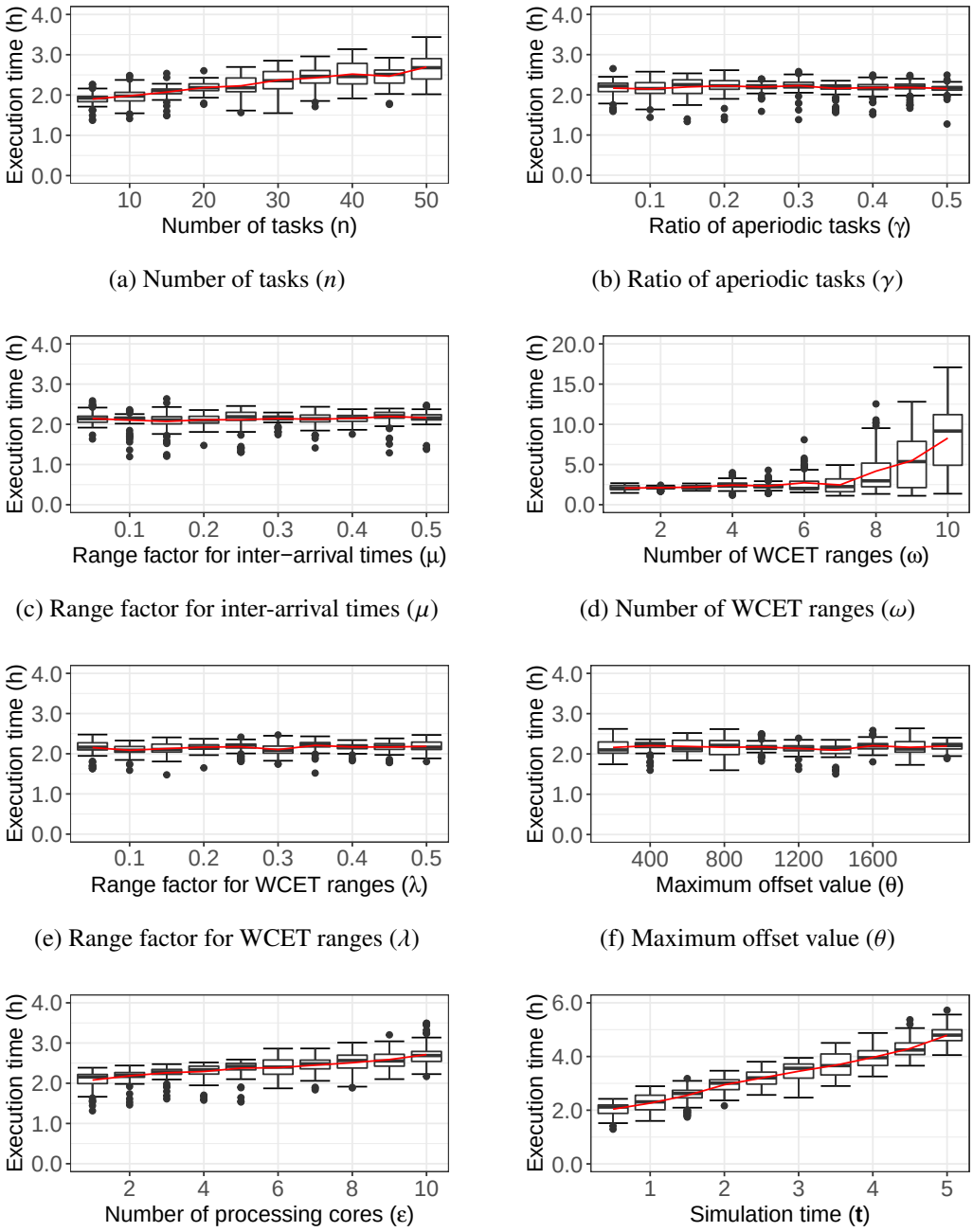
Figure 3.8 shows the inferred safe border which identifies safe WCET ranges within which all 34 tasks are schedulable with an estimated deadline miss probability of 1.97%. Given the safe border, we found a best-size point which restricts the WCET ranges of T30 and T33 as follows: T30 [0.1ms, 458.0ms] and T33 [0.1ms, 2138.1ms]. We note that the initial estimated WCET ranges of the two tasks are as follows: T30 [0.1ms, 900.0ms] and T33 [0.1ms, 20000.0ms]. SAFE therefore resulted in safe WCET ranges representing a significant decrease of 49.11% and 89.31% of initial maximum WCET estimates, respectively. This information is therefore highly important and can be used to guide design and development.

The answer to **RQ3** *is that* SAFE helps compute safe WCET ranges that have a much lower maximum than practitioners' initial WCET estimates. Our case study showed that SAFE determined safe maximum WCET values that were only 51% or less the original estimate. Further, these safe WCET ranges have a deadline miss probability of 1.97% based on the inferred logistic regression model. More restricted ranges can be selected to reduce this probability. SAFE took, on average, 25.14h to compute such safe WCET regions, which is acceptable for offline analysis in practice.

RQ4. Figure 3.9 shows the distributions (25%-50%-75%) of execution times obtained from 10×10 runs of SAFE, i.e., 10 runs for each of the 10 synthetic systems with the same experimental setting (see Section 3.5.4). The red solid lines in Figure 3.9 represent the mean value changes of the execution times of SAFE over varying values of the following control parameters: (a) number of tasks n, (b) ratio of aperiodic tasks γ, (c) range factor for inter-arrival times μ, (d) number of WCET ranges ω, (e) range factor for WCET ranges λ, (f) maximum offset value θ, (g) number of processing cores ϵ, and (h) simulation time $\mathbf{t}$. Note that all the experiments in EXP4 took at most 16.7h, which is acceptable as SAFE is an offline analysis technique.

As shown in Figures 3.9a, 3.9g, and 3.9h, the execution time of SAFE is linear in the number of tasks n, the number of processing cores ϵ, and the simulation time $\mathbf{t}$. However, regarding the ratio of aperiodic tasks γ (Figure 3.9b), the range factor for inter-arrival times μ (Figure 3.9c), the range factor for WCET ranges λ (Figure 3.9e), the maximum offset value θ (Figure 3.9f), the results indicate that they have no correlations with the execution time of SAFE. Therefore, we expect SAFE to scale well as the number of tasks, the number of processing cores, and the simulation time increase.

Figure 3.9d shows that the execution time of SAFE is quadratically correlated with the number of WCET ranges ω in a system. Recall from Section 3.4 that the number of tasks characterizing their WCETs as ranges (instead of point values) determines the size of a labeled dataset. Specifically, SAFE uses a second-order polynomial response surface model (RSM) to build a logistic regression model. RSM contains linear terms, quadratic terms, and 2-way interactions between linear terms (see Section 3.4.2). Hence, the number of coefficients in an RSM to be inferred by logistic regression is the sum of the numbers of constants, linear terms, quadratic terms, and 2-way interactions in an RSM, i.e., $1+\omega+\omega+\binom{\omega}{2}$ (see the RSM equation presented in Section 3.4.2), which impacts the execution time of logistic regression dominating the execution time of SAFE. Note that the execution time of logistic regression is linear in the number of coefficients of a regression model (e.g., RSM) [100]. Hence, the execution time of logistic regression in SAFE is quadratically correlated with the number of WCET ranges as explained above. In addition, the results presented in Figure 3.9d show that the magnitude of the execution time



(a) Number of tasks (n)

(b) Ratio of aperiodic tasks (γ)

(c) Range factor for inter-arrival times (μ)

(d) Number of WCET ranges (ω)

(e) Range factor for WCET ranges (λ)

(f) Maximum offset value (θ)

(g) Number of processing cores (ϵ)

(h) Simulation time (**t**)

Figure 3.9: Execution times of SAFE when varying the values of the following parameters: (a) number of tasks n, (b) ratio of aperiodic tasks γ, (c) range factor for inter-arrival times μ, (d) number of WCET ranges ω, (e) range factor for WCET ranges λ, (f) maximum offset value θ, (g) number of processing cores ϵ, and (h) simulation time **t**. The boxplots (20%-50%-75%) show the distributions of execution time values obtained from 100 runs of SAFE, i.e., 10 runs for each of the 10 synthetic systems with the same configuration. The red line in each figure connects the mean values of the execution times of SAFE over the parameter values.

variation increases when the number of WCET ranges in a system increases. As written in Section 3.4.2, SAFE employs a feature reduction technique and a stepwise regression technique to efficiently infer logistic regression models (see lines 1-4 of Algorithm 3.1). The outputs of the two techniques depend on

the number of tasks whose WCETs significantly impact deadline misses. Such outputs then impact the execution times of the following steps in Algorithm 3.1: imbalance handling, sampling, and regression. Note that the synthetic systems generated by setting $\omega = 10$ are more diverse in terms of WCET ranges when compared to the systems created by setting $\omega = 1$.

In EXP4, SAFE analyzes all tasks in a system. However, recall from Section 3.4 that SAFE provides the capability of selecting target tasks as engineers often need to focus on the most critical ones. In such cases, the execution time of SAFE can significantly decrease.

The answer to **RQ4** *is that* the execution time of SAFE is linear in the number of tasks, the number of processing cores, and the simulation time. However, the execution time of SAFE is quadratically correlated with the number of tasks whose WCETs are given as ranges. Across our experiments, SAFE took at most 17.1h, which is acceptable for offline analysis in practice.

Benefits from a practitioner's perspective. Investigating practitioners' perceptions of the benefits of SAFE is necessary to adopt SAFE in practice. To do so, we draw on the qualitative reflections of three software engineers at LuxSpace, with whom we have been collaborating on this research. The reflections are based on the observations that the engineers made throughout their interactions with the researchers.

SAFE produces a set of worst-case sequences of task arrivals (see Section 3.4.1). Engineers deemed them to be useful for further examinations by experts. The current practice is to use an analytical schedulability test [118] which proves whether or not a set of tasks are schedulable. Such an analytical technique typically does not provide additional information regarding possible deadline misses. In contrast, worst-case task arrivals and safe WCET ranges produced by SAFE offer insights to engineers regarding deadline miss scenarios and the conditions under which they happen.

Engineers noted that some tasks' WCET are inherently uncertain and that such uncertainty is hard to estimate based on expertise. Hence, their initial WCET estimates were very rough and conservative. Further, estimating what WCET sub-ranges are safe is even more difficult. Since SAFE estimates safe WCET ranges systematically with a probabilistic guarantee, the engineers deem SAFE to improve over existing practice. Also, SAFE allows engineers to choose system-specific safe WCET ranges from the (infinite) WCET ranges modeled by the safe border, rather than simply selecting the best-size WCET range automatically suggested by SAFE (Figure 3.8). This flexibility allows engineers to perform domain specific trade-off analysis among possible WCET ranges and is useful in practice to support decision making with respect to their task design.

Given the fact that we have not yet undertaken rigorous user studies, the benefits highlighted above are only suggestive but not conclusive. We believe the positive feedback obtained from LuxSpace and our industrial case study shows that SAFE is promising and worthy of further empirical research with human subjects.

3.5.8 Threats to validity

Internal validity. To ensure that our promising results cannot be attributed to the problem merely being simple, we compared SAFE with an alternative baseline using random search under identical parameter settings (see the RQ1 results in Section 3.5.7). Phase 1 of SAFE can indeed be replaced with a random search, as we did, or even an exhaustive technique if the targeted system is small. However, there are no alternatives for Phase 2 – our main contribution – which infers safe WCET ranges and enables trade-off analysis. We present all the underlying parameters and provide our full evaluation package [112] to

facilitate reproducibility. We mitigate potential biases and errors in our experiments by drawing on an industrial case study in collaboration with engineers at LuxSpace.

Recall from Section 3.5.7 that we compared probability values of deadline misses computed by SAFE and simulations. The results show that SAFE enables engineers to have more conservative probabilistic interpretations of the estimated WCET ranges than simulation-based evaluations for the WCET ranges. However, depending on the system's characteristics, e.g., hard real-time systems, engineers may need absolute guarantees for the WCET estimates. In such cases, once engineers find safe WCET ranges using SAFE, they can, in theory, obtain an absolute guarantee using exhaustive verification techniques, e.g., UPPAAL [131], on whether tasks always meet their deadlines for the given WCET ranges or not. Note that we performed an experiment using UPPAAL as it has often been used in the literature [132, 192, 190]. We applied UPPAAL to verify whether ADCS tasks are schedulable for the given WCET values. However, our experiment results showed that UPPAAL was not able to complete the analysis task, even after five days of execution. Hence, engineers should therefore consider such scalability issues when applying exhaustive analysis techniques to complement SAFE. Since this UPPAAL evaluation is not the main focus of this chapter, we point the reader to the UPPAAL specification of ADCS available online [112].

External validity. The main threat to external validity is that our results may not generalize to other contexts. We evaluated SAFE using early-stage WCET ranges estimated by practitioners at LuxSpace. However, SAFE can be applied at later development stages as well (1) to test the schedulability of the underlying set of tasks of a system and (2) to develop tasks under more precise constraints regarding safe WCETs. Future case studies covering the entire development process remain necessary for a more conclusive evaluation of SAFE. In addition, while motivated by ADCS (see Section 3.5.2) in the satellite domain, SAFE is designed to be generally applicable to other contexts. To evaluate the usefulness of SAFE in other contexts, in addition to our motivating case study system, we applied SAFE to two industrial systems from different domains, having very different system characteristics such as resource dependencies and multiple processing cores. As we described in Section 3.5.2, however, none of the public study subjects provide initial estimates of their task WCETs as ranges, which are required by SAFE as input. Hence, we had to modify the study subjects to include WCET ranges in their task descriptions but attempted to minimize any potential biases and errors in the experiments by converting a point WCET value to a WCET range in a systematic and straightforward way. In Section 3.5.2, we described the converting method in detail. Also, we made the modified task descriptions available online [112]. However, the general usefulness of SAFE needs to be further assessed in other contexts and domains.

3.6 Related Work

This section discusses and compares SAFE with related work in the areas of schedulability analysis, as well as testing and verification of real-time systems.

Schedulability analysis has been widely studied for real-time systems [27, 29, 126, 21, 188, 32, 47, 84, 85, 134, 128, 166, 189, 7, 87, 33, 175, 10, 143, 138]. Among them, the most related research strands study uncertain execution times [32, 188, 21, 134], probability of deadline misses [175, 128, 126], weakly hard deadlines [27, 189, 143], schedulability regions [47, 166, 10], and WCET estimation [29, 84, 85, 7, 87, 33] in the context of real-time task analysis.

Bini et al. [32] propose a theoretical sensitivity analysis method for real-time systems accounting for a set of periodic tasks and their uncertain execution times. Brüggen et al. [175] present an analytical

method to analyze a deadline miss probability of real-time tasks using probability density functions of approximated task execution times. In contrast to SAFE, most of these analytical approaches do not directly account for aperiodic tasks having variable arrival intervals; instead, they treat aperiodic tasks as periodic tasks using their minimum inter-arrival times as periods [60]. However, SAFE takes various task parameters, including irregular arrival times, into account without any unwarranted assumption. Also, our simulation-based approach enables engineers to explore different scheduling policies provided by real RTOS; however, these analytical methods are typically only valid for a specific conceptual scheduling policy model.

Bernat et al. [27] introduce the concept of weakly hard real-time systems that can tolerate occasional deadline misses. They precisely define weakly hard deadline constraints, specifying a maximum number of deadlines that can be missed during a time window, and provide the theoretical analysis of the properties and relationships between tasks with the temporal constraints. Xu et al. [189] develop an algorithm that accounts for sporadic task overload when analyzing the number of deadlines a task can miss in a given sequence of consecutive task arrivals. Pazzaglia et al. [143] present an extended weakly hard analysis method by accounting for additional uncertainties such as task offsets and release jitters. SAFE complements the above research strands on weakly hard real-time systems since, instead of analyzing the number of deadlines a task can afford to miss over a time window, SAFE provides probabilistic guarantees for deadline misses based on logistic regression models inferred from search and simulation outputs.

Cimatti et al. [47] develop an approach that computes the regions of the task parameter values guaranteeing tasks are schedulable, i.e., schedulability regions. Their approach uses parametric timed automata and an SMT solver, i.e., NuSMT [41], and is applied to a system that contains two periodic tasks. Sun et al. [166] propose a method, named IMITATOR, that aims at computing schedulability regions. IMITATOR is based on model checking of parametric timed automata with stopwatches. They evaluate the method by applying it to two test-case systems that contain at most two free parameters, i.e., task execution times, that are defined as variables. Note that the other parameters, e.g., task periods and deadlines, are defined as fixed values. The results show that IMITATOR covers the entire parameter space but does not scale well with the size of the problem. André et al. [10] developed a tool that translates a graphical specification of a real-time system to the input of IMITATOR such that it allows computation of some schedulability regions using IMITATOR. However, these methods that exhaustively search the problem space are often not amenable to analyze industrial systems in a scalable manner as they typically contain many tasks, different task types, complex task relationships, and multiple processing cores.

Hansen et al. [85] present a measurement-based approach to estimate WCET and a probability of estimation failure. The measurement-based WCET estimation technique collects actual execution time samples and estimates WCETs using linear regression and a proposed analytical model. To our knowledge, most of the research strands regarding WCET estimation are developed for later development stages at which task implementations are available. Note that relatively few prior works aim at estimating WCET at an early design stage; however, these work strands still require access to source code, hardware, compilers, and program behavior specifications [84, 7, 33]. In contrast, SAFE uses as input estimated WCET ranges and then precisely restricts the WCET ranges within which tasks are schedulable with a selected deadline miss probability, by relying on a tailored genetic algorithm, simulation, feature reduction, a dedicated sampling strategy, and logistic regression.

Testing and verification are important to successfully develop safety-critical real-time systems [131, 193, 65, 5, 106, 36]. Some prior studies employ model-based testing to generate and execute tests for real-time systems [131, 193, 65]. SAFE complements these prior studies by providing safe WCETs as objectives to engineers implementing and testing real-time tasks. Constraint programming and model checking have been applied to ensure that a system satisfies its time constraints [5, 106]. These techniques may be useful to conclusively verify whether or not a WCET value is safe. However, such exhaustive techniques are not amenable to address the analysis problem addressed in this chapter, which requires the inference of safe WCET ranges. To our knowledge, SAFE is the first attempt to accurately estimate safe WCET ranges to prevent deadline misses with a given level of confidence and offer ways to achieve different trade-offs among tasks' WCET values.

3.7 Conclusion

We developed SAFE, a two-phase approach applicable in early design stages, to precisely estimate safe WCET ranges within which real-time tasks are likely meet their deadlines with a high-level of confidence. SAFE uses a meta-heuristic search algorithm to generate worst-case sequences of task arrivals that maximize the magnitude of deadline misses, when they are possible. Based on the search results, SAFE uses a logistic regression model to infer safe WCET ranges within which tasks are highly likely to meet their deadlines, given a selected probability. SAFE is developed to be scalable by using a combination of techniques such as a genetic algorithm and simulation for the SAFE search (phase 1) and feature reduction, an effective sampling strategy, and polynomial logistic regression for the SAFE model refinement (phase 2). We evaluated SAFE on a mission-critical, real-time satellite system in collaboration with a satellite company as well as two industrial systems from different domains whose description was retrieved from the literature. The results indicate that SAFE is able to precisely compute safe WCET ranges for which deadline misses are highly unlikely, these ranges being much smaller than the WCET ranges initially estimated by engineers. Further, we evaluated the scalability of SAFE using a number of synthetic systems. The results indicate that SAFE scales to complex systems. Across the experiments on industrial and synthetic systems, SAFE took at most 27h, which is acceptable in practice as an offline analysis method.

For future work, we plan to extend SAFE in the following directions: (1) developing a real-time task modeling language to describe dependencies, constraints, behaviors of real-time tasks and to facilitate schedulability analysis and (2) building a decision support system to recommend a schedulable solution if a set of tasks are not schedulable, e.g., priority re-assignments. In the long term, we would like to more conclusively validate the usefulness of SAFE by applying it to other case studies in different domains.

Chapter 4

Estimating Probabilistic Safe WCET Ranges for Weakly Hard Real-Time Systems

4.1 Introduction

Weakly hard real-time systems can tolerate occasional deadline misses while hard real-time systems must meet their deadlines in every task activation (or task arrival) [27]. Though weakly hard systems have relaxed deadline constraints, this does not mean that schedulability in weakly hard real-time systems can be ignored [40]. For instance, telecommunications and media streaming services do not cause catastrophic events even if they miss some of the deadlines; however, they need to satisfy a desired performance requirement to provide sufficiently reliable service. Therefore, many researchers [28, 4, 79, 165, 81, 143] have investigated the schedulability of real-time systems with weakly hard constraints (thresholds of consecutive deadline misses).

Schedulability analysis determines whether or not all tasks in a real-time system satisfy their deadline constraints, even in worst-case scenarios. Schedulability is verified with abstract task properties, such as task priorities, deadlines, inter-arrival times, and worst-case execution times (WCETs) [163, 50]. Task priority, inter-arrival time, and deadline tend to be either specified or predicted in early development stages and they are relatively accurate. More specifically, task priority can be determined by the selected scheduling policy (e.g., rate monotonic [118]) or by the task criticality levels (e.g., more critical tasks are prioritized over the less critical ones). Inter-arrival time, which is the amount of time between consecutive task executions, and task deadline can be provided or analyzed by the system requirements. In contrast, WCET is influenced by more intricate variables, such as implementation choices, task duration, and hardware specifications. We rarely can determine such factors at early development stages, thus preventing accurate WCET estimation [84, 7, 33]. Therefore, schedulability analysis is typically postponed to later development stages.

Estimating WCET at early stages, however, can help practitioners make important decisions regarding their system design and implementation. When practitioners need to make decisions about data storage, for example between a relational database or an in-memory storage, WCET can be a good selection criterion. Additionally, early WCET estimates support practitioners in determining optimal configurations of hardware devices, e.g., CPUs or sensors, especially in the context of parallel development of both software and hardware, which is common in the aerospace, automotive, and healthcare domains.

The problem of estimating WCET has been widely studied based on measurements [183, 51, 153] and static analysis [73, 169, 133, 86]. The measurement-based approaches estimate WCET through multiple executions on the target hardware or an accurate simulator, using a set of inputs that are expected to be worst-case. In contrast, the static analysis-based approaches estimate WCET by investigating the longest path in the source code and the cache hit ratio based on hardware specifications. Since these approaches work based on source code, they cannot be applied at early development stages. Moreover, some approaches [84, 7, 33] aim to estimate WCET at early stages of implementation. They design a timing model that predicts execution times of machine instructions. They then translate a given source code to instructions for the timing model. The execution time of these instructions allows for approximate WCET estimation. Although several studies have been conducted on this, accurate estimation of WCET still remains a challenge due to increasing hardware and software complexities and finding worst-case input data [156]. Hence, practitioners tend to estimate WCET as a range by adding a margin (e.g., 20%) to the estimated WCET [40, 57]. Recently, SAFE [113] has been proposed to estimate WCET ranges that satisfy deadline constraints with a probabilistic guarantee. SAFE utilizes a machine learning technique to estimate WCET ranges based on the worst-case sequences of task arrivals that are found by using a meta-heuristic technique. The approach provides safe WCET ranges at early stages by analyzing schedulability from the estimated WCET ranges instead of analyzing or running source codes. This approach is effective for hard real-time systems as it maximizes the degree of deadline misses. However, in practice, many real-time systems allow weakly hard constraints that can tolerate a small number of consecutive deadline misses, unless the sequence of deadline misses is too long [54]. For weakly hard constraints, finding consecutive deadline misses with a short delay may be more important than finding a deadline miss with a long delay. Therefore, maximizing the degree of deadline miss in SAFE is incompatible with weakly hard real-time systems.

In this chapter, we propose the Safe Worst-case execution time (WCET) analysis method for wEAKly hard real-time system (SWEAK). SWEAK searches for the worst test cases that likely cause violations of weakly hard constraints based on a genetic algorithm [122] and estimates safe WCET ranges with a probabilistic guarantee by using logistic regression [100]. SWEAK evaluates the schedulability of a system with test cases by using an industrial adaptive partitioning scheduler (APS) that supports complex scheduling policies with task partitioning and core assigning (affinity) in multi-core platforms. As APS is widely used in many domains, such as automotive, transportation, medical devices, and various other embedded systems, SWEAK can also be widely applied to such domains. SWEAK infers a *safe WCET border* characterizing *safe* WCET ranges with a certain probability p of violating deadline constraints in the given multidimensional WCET space. Such border allows practitioners to investigate, for each dimension, trade-offs among WCET ranges and to select suitable (implementable) WCET ranges. In this chapter, we refer to probabilistically safe WCET ranges as safe WCET ranges for the purpose of simplicity. We evaluated SWEAK with an industrial system from a satellite domain and synthetic systems

that are generated following guidelines from our industry partner, Blackberry. The experimental results demonstrate that SWEAK can infer satisfying safe WCET ranges under both hard and weakly hard constraints in various systems. Regarding the execution time of SWEAK, it takes at most 22.1h across a large number of synthetic systems, which implies that our approach is acceptable in practice. All the details of the evaluation results are available online [111].

Organization. This chapter is organized as follows: Section 4.2 explains APS and the related issues and Section 4.3 precisely defines a real-time task model and the problem we want to solve. Section 4.4 describes our approach based on the models. Section 4.5 empirically evaluates the approach. Section 4.6 contrasts our approach against related works. Section 4.7 concludes this chapter.

4.2 Motivation

This work was inspired by the collaboration with our partner company, Blackberry. The company has developed a real-time operating system (RTOS), named QNX Neutrino[1], which satisfies ISO-26262 with the highest level of assurance (ASIL-D). Because of such high assurance, the RTOS has been used in many industrial embedded systems in critical domains such as automotive, transportation, and medical devices. To support the various system requirements, QNX Neutrino invented a sophisticated scheduler, named adaptive partitioning scheduler (APS), supporting partitioning, multi-core platforms, and multi-partitions.

Overview of APS. The main objectives of APS are to provide enough resources for important tasks and to prevent unimportant or untrusted tasks from monopolizing system resources, such as processing cores. For example, when a task is under a Denial of Service (DOS) attack, the rest of the tasks are starving since the system is too busy to provide service to the task under attack. APS solves such an issue by introducing partitions that separate tasks into virtual containers. Each partition has its own budget, which is the maximum ratio of used resources. Each task in APS needs to be assigned to a partition.

APS manages time budgets flexibly depending on system performance. When a scheduler uses fixed budget management, it may result in inefficiencies when a partition needs additional budget while the other partitions are not using their budgets. Instead, APS records the usage of processing cores for tasks in each partition over the duration of a specified time window (e.g., 100ms by default). At every time window passes, APS compares the records with the time budgets for each partition and redistributes free time to the partitions that require more time. This adaptive partitioning ensures that APS achieves efficiency in scheduling tasks not only in normal loads but also in abnormal loads.

APS basically follows a priority-driven preemptive scheduling policy. Tasks are thus allocated a processing core according to their priority order for scheduling. Such policy ensures the highest priority task can always use a processing core whenever required. When a lower priority task is using a processing core, a higher priority task can take over. In some systems, engineers may assign some tasks the same priority. For those tasks, APS provides three different scheduling policies: FIFO, round-robin, and sporadic scheduling[2]. Note that depending on the scheduling policy, additional configurations may be needed, e.g., timeslice for round-robin, which is an interval for rotating tasks.

[1] QNX Neutrino RTOS: `https://blackberry.qnx.com/en/products/foundation-software/qnx-rtos`

[2] Sporadic scheduling policy: `http://www.qnx.com/developers/docs/7.1/index.html#com.qnx.doc.neutrino.prog/topic/overview_Sporadic_scheduling.html`

APS also supports multi-core platforms since many real-time systems have relied on multi-core platforms in recent years [3]. APS schedules tasks by assigning them any idle processing cores according to their priority. This allocation method is a good way to maximize the usage of processing cores. However, it causes inefficiencies in some cases as it involves task migration across processing cores. To deal with this, APS allows practitioners to define "core affinities", which specify manual assignment to one or more processing cores for each task.

Issues on schedulability analysis. Based on the advantages of APS, QNX Neutrino can be widely used in many applications. However, practitioners typically face difficulties with schedulability analysis due to the complexity of the scheduler. Especially, a system that uses multi-partitions and core affinity can lead to priority inversion. For example, let us assume that two tasks with different priorities are assigned to different partitions. In some cases, a higher priority task may be preempted when the partition budget of the task is exhausted. Without partitions, a higher priority task cannot be preempted by a lower priority one. The issue also happens when tasks feature core affinity, i.e., a high-priority task that has core affinity, is waiting for its processing core while a low-priority task can be executed by the other processing core. Theoretical schedulability analysis methods [118, 57, 165, 81, 60] are not applicable for systems using such complex scheduling strategies.

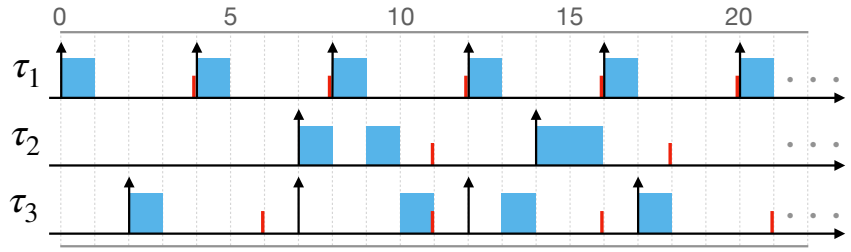
Additionally, practitioners face more schedulability analysis issues regarding uncertain WCET values and weakly hard constraints. Practitioners commonly estimate WCET values of tasks based on multiple runs of task executions or by analyzing the source code of the tasks. However, such estimation is challenging due to uncertainty in hardware and software configurations. The estimated WCET values are instead used as lower bounds, and upper bounds are determined by increasing lower bounds to some pre-defined degree [54]. These WCET range estimates make schedulability analysis even more challenging. It is, however, even harder when the source code is not fully available at early development stages. Weakly hard constraints further increase the complexity of schedulability analysis. As the provider of RTOS, to increase its usage, our partner company, Blackberry, wants to provide schedulability analysis tools that are scalable and applicable to the many complex systems requiring such flexibility.

4.3 Problem definition

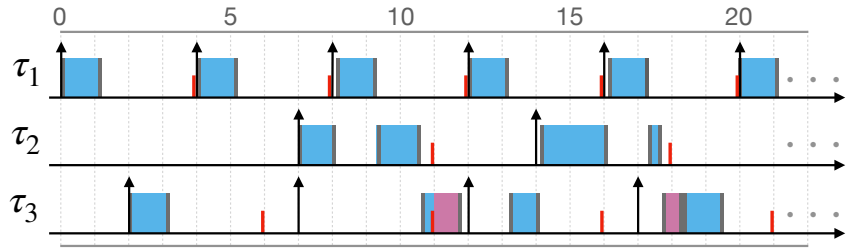
This section first introduces the task model and the notations we use in this chapter. We then, describe the problem of identifying safe WCET ranges that satisfy the deadline constraints with a certain level of confidence.

The task model. This work targets real-time systems that allow weakly hard constraints and run on a multi-core platform. We define a real-time system Γ as a set of n tasks. Since the tasks execute repetitively and should meet their deadline constraints, we define the properties of the task τ_i as a tuple: $(P_i,\ O_i,\ T_i,\ C_i,\ D_i,\ (m_i, K_i))$, where P_i is the task priority, O_i is the offset, T_i is the task period, C_i is the worst-case execution time (WCET), D_i is the relative deadline, and (m_i, K_i) is the deadline constraint. See the detailed definitions in Section 2.1.

As discussed earlier, engineers face difficulties when estimating the exact value of WCET at early stages of development because there are many uncertain factors related to hardware configuration, input data, and source code. For those tasks, we assume that engineers can provide WCET ranges instead of a single value. In such cases, we denote by $[C_i^{min}, C_i^{max}]$ the minimum and maximum WCET values for a task τ_i.



(a) Schedule scenario without context switching times



(b) Schedule scenario with context switching times

Figure 4.1: Examples of schedule scenarios that describes task executions for three tasks, τ_1, τ_2, and τ_3, running on a single core system. Under the same sequence of task arrivals, (a) has no deadline misses without considering context switching times, and (b) has deadline misses at the second and third arrivals of τ_3 with considering context switching times.

A deadline D_i is a time constraint for task τ_i. We allow the deadline to be arbitrarily greater than C_i [43], i.e., $C_i < D_i$. For a task τ_j that has a WCET range, the deadline value should be greater than the maximum WCET value C_j^{max}. Depending on the task, the time constraint should be met when it is a hard constraint and can be missed occasionally when it is a weakly hard constraint. To deal with the degree of deadline constraint violations, we introduce a constraint (m_i, K_i) for a task τ_i, which defines a tolerable number m_i of consecutive task arrivals that miss the deadline D_i within a given time window K_i (i.e., the number of consecutive arrivals) for a task τ_i [27]. Note that $m_i = 0$ if τ_i is a hard constraint while $m_i > 0$ if τ_i is a weakly hard constraint.

Context switching times. In addition to the task model, we consider context switching times during scheduling. The context switching time is the time that it takes when the state of a task changes according to the task state transition model (see Section 2.1). Depending on the current state, we define three types of context switching times as follows. Start-up time, denoted by λ_s, is the time required to change the state of a task from *ready* to *running*. Exit time, denoted by λ_x, is the time required to change the state of a task from *running* to other states, *ready* or *blocked*, or to finalize its execution. Inter-processor interrupt (IPI) time, denoted by λ_p, is the time required to propagate a task execution from one core to another in a multi-core platform. IPI can be invoked when the state of a task changes from *ready* to *running*.

Figure 4.1 compares schedule scenarios depending on the context switching times for the same sequence of task arrivals on a single-core platform. The examples contains three tasks τ_1, τ_2, and τ_3. The periodic task τ_1 is characterized by: $O_1 = 0$, $T_1 = 4$, $C_1 = 1$, and $D_1 = 4$. The aperiodic task τ_2 is characterized by: $O_2 = 0$, $[T_2^{min}, T_2^{max}] = [4, 12]$, $C_2 = 2$, and $D_2 = 4$. The periodic task τ_3 is characterized by: $O_3 = 2$, $T_3 = 5$, $C_3 = 1$, and $D_3 = 4$. Priorities of the tasks are $P_1 > P_2 > P_3$, i.e., τ_1 can preempt τ_2 and τ_3 and occupy the processing core at any time. Figure 4.1a shows an example of task

executions leaving out context switching times. The example has no deadline misses in all the task arrivals. In contrast, Figure 4.1b illustrates an example of task executions with context switching times assuming $\lambda_s = \lambda_x = 0.1$, which are the gray bars before and after each execution of tasks respectively. Note that IPI time λ_p does not appear since the scenario is running on a single-core platform. The example shows that τ_2 and τ_3 are more preempted than in the example of Figure 4.1a. This happens because all task arrivals are delayed by the context switching at starting up and exiting their execution. The task τ_3 thus misses its deadline at the second arrival $a_{3,2}$ and the third arrival $a_{3,3}$. Though context switching times are shorter than the task execution times, the more preemptions, the larger the impact of context switching.

These time delays are affected by hardware performance as well as scheduling overhead in a scheduler. Therefore, these values cannot be accurately calculated, and we define them as ranges. For example, the start-up time can be any value in the range [0.012ms, 0.022ms]. In our context, when performing schedule simulation, we select values from the specified ranges. In our case studies, we specified these ranges following guidelines from the partner company. Note that each selected context switching time is applied to all task arrivals in one simulation.

Schedulability. Schedulability determines whether all task arrivals in a system are satisfying their deadline constraints or not. Schedulability analysis is conducted with a *schedule scenario*, which describes a schedule result from a specified schedule simulator. Recall the definition of *schedule scenario* in Section 2.1. We formulate *schedule scenario* as a set of tuples: $(\tau_i, a_{i,k}, e_{i,k})$, where $a_{i,k}$ and $e_{i,k}$ are, respectively, the arrival time and the end (or completion) time of the kth task arrival of the task τ_i. For example, assuming that the scheduling period is [0, 22], S_a in Figure 4.1a becomes $\{(\tau_1, 0, 1), \cdots, (\tau_2, 7, 8), \cdots, (\tau_3, 12, 14), (\tau_3, 17, 18)\}$ and S_b in Figure 4.1b becomes $\{(\tau_1, 0, 1), \cdots, (\tau_2, 7, 8.2), \cdots, (\tau_3, 12, 18.2), (\tau_3, 17, 19.4)\}$. Due to randomness in task execution times, aperiodic task arrivals and context switching times, a scheduler may generate a different schedule scenario for different runs. The schedulability of a given a schedule scenario S can vary regarding a (m, K)-constraint. For example, Figure 4.1b has two deadline misses for task τ_3. If the deadline constraint (m_3, K_3) of τ_3 is (1,4), the scenario S is not schedulable as the deadline constraint of τ_3 only allows one deadline miss in four consecutive arrivals. However, the scenario S become schedulable when (m_3, K_3) is equal to (2,4). This scenario S can also be schedulable when the execution time of τ_3 is reduced to 1, i.e., C_2=1, even (m_3, K_3) is equal to (1,4).

Problem. The effective design and assessment of real-time systems rely on the accurate evaluation of the properties of tasks. Among the values, WCET values are estimated as ranges, which is inevitable given the high uncertainties at early stages of development. Under given WCET ranges, the upper WCET bounds are the worst-case WCET values that are likely to have deadline misses, since larger WCET values increase the probability of deadline constraint violations. Lower WCET bounds are tasks' best-case WCET values but are harder to implement in practice. We aim at determining the maximum upper bounds that allow the tasks to be schedulable under weakly hard constraints at a certain level of probability of violating deadline constraints. Practitioners can use these upper bounds as an objective when implementing the tasks. Specifically, for every task $\tau_i \in \Gamma$ to be analysed, our approach computes a new upper bound value for the WCET range of τ_i (denoted by C_i^{max*}) such that $C_i^{max*} \leq C_i^{max}$ and by restricting the WCET range of τ_i to C_i^{max*} we should, at a certain level of confidence, no longer have deadline constraint violations. That is, tasks Γ become schedulable, with a certain probability, after restricting the maximum WCET

value of τ_i to C_i^{max*}. For instance, as shown in Figure 4.1b, restricting the maximum WCET of τ_3 from $C_3^{max} = 2$ to $C_3^{max*} = 1$ enables all the three tasks to be schedulable.

4.4 Approach

Figure 4.2 shows an overview of our Safe Worst-case execution time (WCET) analysis method for wEAKly hard real-time system (SWEAK). Given the task descriptions, the approach first finds the worst test cases, which consist of sequences of task arrivals for given tasks and context switching times, using a meta-heuristic search maximizing the violation of weakly hard constraints (Section 4.4.1). During search, the approach works by relying on a simulation technique APSSimulator, a schedule simulator that mimics the behavior of APS (Section 4.4.2), to evaluate the schedulability of test cases and produces training data. The approach then builds a logistic regression model to distinguish between *safe* and *unsafe* areas in the WCET space using the training data (Section 4.4.3). The model estimates WCET ranges under which tasks are likely to be schedulable with a probabilistic guarantee. The approach refines the model with additional training data simulated by the worst test cases. In the next section, we describe each step of the approach in detail.

4.4.1 Searching for the worst test cases

The search step of our approach aims to provide the worst test cases that likely cause violations of deadline constraints by extending the Chapter 3. Since we deal with weakly hard real-time systems, we apply a multi-objective search algorithm for finding the worst test cases based on the following two objectives: (1) maximizing the magnitude of deadline misses and (2) maximizing the consecutiveness degree of deadline misses by weighing the interval of task arrivals that missed their deadlines. These two objectives lead to test cases, which are task arrival sequences and context switching times, which cause larger deadline misses and more consecutive deadline misses. To evaluate the test cases, we apply various sets of WCET values that are randomly sampled within specified ranges, since such WCET values can lead to different schedule results with the same test cases. We describe our search-based approach by defining the solution representation, the fitness functions, and the computational search algorithm following standard practice [74].

Representation. A feasible solution represents a test case for checking schedulability. Given a set Γ of tasks to be scheduled, a solution I consists of two parts, context switching times and sequences of task arrivals for all tasks in Γ. The context switching times are three scalar values, start-up λ_s, exit λ_x, and

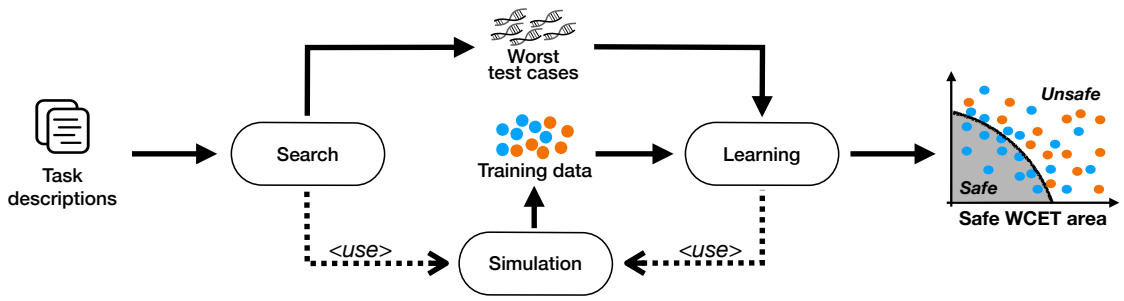


Figure 4.2: An overview of our Safe WCET analysis method for wEAKly hard real-time system (SWEAK)

IPI λ_p times, each of which is selected within their valid range (see Section 4.3). The sequences of task arrivals are denoted by a set A of tuples $(\tau_i, a_{i,k})$, where $\tau_i \in \Gamma$ and $a_{i,k}$ is the kth arrival time of a task τ_i. The number of arrivals for a task τ_i is decided by the scheduling time period $\mathbb{T} = [0, \mathbf{t}]$. For example, if a task τ_i is periodic, the number of arrivals of τ_i is fixed as $\mathbf{t}/T_i$ with the offset $O_i = 0$. In the case of aperiodic tasks, the number of arrivals varies due to its inter-arrival times (see Section 4.3). Therefore, the size of I varies across different solutions along with the size of A.

Fitness. To evaluate the fitness of each solution, we define two objective functions, which calculate the magnitude and consecutiveness degree of deadline misses in the given target tasks.

These objective functions are calculated with multiple simulation runs to account for uncertainty in WCET. Specifically, given a solution I for a set Γ of tasks, SWEAK runs APSSimulator ns times with randomly selected WCET values in the given WCET ranges and obtains schedule scenarios $\mathbf{S} = \{S_1, S_2, \cdots, S_{ns}\}$ (see Section 4.4.2). Based on the scenarios, SWEAK calculates the fitness values for a solution I according to the fitness functions.

Fitness for the magnitude of deadline misses. This fitness function, denoted by $fd(I, \Gamma^\delta, ns)$, quantifies the maximum degree of deadline misses regarding a set $\Gamma^\delta \subseteq \Gamma$ of target tasks. SWEAK provides the capability of selecting target tasks Γ^δ as practitioners often need to focus on a subset of critical tasks. To this end, we first define a distance function $dist(\tau_i, k)$ as follows:

$$dist(\tau_i, k) = e_{i,k} - a_{i,k} + D_i$$

It captures the difference between the end time and the deadline of the kth arrival of task τ_i in a schedule scenario (see Section 4.3 for each notation). If an arrival $a_{i,k}$ misses its absolute deadline $a_{i,k} + D_i$, the value of $dist(\tau_i, k)$ is larger than 0. Based on the distance function, SWEAK aims to maximize the $fd(I, \Gamma^\delta, ns)$ fitness function defined as follows:

$$fd(I, \Gamma^\delta, ns) = \sum_{h=1}^{ns} \max_{\tau_i \in \Gamma^\delta,\ k \in [1, lk(\tau_i)]} dist_h(\tau_i, k)/ns$$

where $lk(\tau_i)$ is the maximum number of arrivals of a task τ_i in I. Note that $dist_h(\tau_i, k)$ is the $dist(\tau_i, k)$ function for each schedule scenario S_h. Thus, the function $fd(I, \Gamma^\delta, ns)$ calculates the average maximum size of deadline misses for the tasks in Γ^δ in each scenario S_h.

Fitness for the consecutiveness of deadline misses. This fitness function, denoted by $fc(I, \Gamma^\delta, ns)$, quantifies the intervals between task arrivals that miss their deadlines for a set $\Gamma^\delta \subseteq \Gamma$ of target tasks. To this end, we convert a schedule scenario into μ-patterns [27] by capturing whether each task arrival meets its deadline or not. Specifically, given a schedule scenario S, a μ-pattern μ_i for a task τ_i is calculated as follows:

$$\mu_i(k) = \begin{cases} 1 & , dist(\tau_i, k) > 0 \\ 0 & , otherwise \end{cases}$$

For example, if a task τ_i has six task arrivals and the second and the fifth arrivals missed their deadlines, a μ_i is equal to {0,1,0,0,1,0}. Based on the μ-pattern, we calculate the interval, denoted by $interval(\tau_i, k)$, between task arrivals that missed their deadlines (which are 1 of the $\mu_i(k)$). In the above example, the $interval(\tau_i, 2)$ is equal to 3 because the next deadline missed arrival is the fifth arrival (i.e., 5-2=3). The $interval(\tau_i, 5)$ is equal to ∞ because the next deadline missed arrival is unknown. Note that the $interval(\tau_i, k)$ returns to 0 when the kth μ_i is 0.

Given the function $interval(\tau_i, k)$, we denote by $consec(\tau_i, k)$ the consecutiveness degree for the kth arrival of a task τ_i as follows:

$$consec(\tau_i, k) = \begin{cases} 10^{\frac{1}{interval(\tau_i, k)}} & , \mu_i(k) = 1 \\ 0 & , \mu_i(k) = 0 \end{cases}$$

To reward a small interval and penalize a large interval, we invert $interval(\tau_i, k)$ and take the exponential function, i.e., a consecutiveness degree $consec(\tau_i, k)$ exponentially decreases according to the increasing value of $interval(\tau_i, k)$. For example, with a pattern $\mu_i = \{1,1,0,0,1,0,0,0,1,0\}$, $consec(\tau_i, 1) = \sqrt[1]{10} = 10$ as the $interval(\tau_i, 1) = 1$, $consec(\tau_i, 2) = \sqrt[3]{10} = 2.15$ as the $interval(\tau_i, 2) = 3$, $consec(\tau_i, 5) = \sqrt[4]{10} = 1.58$ as the $interval(\tau_i, 5) = 4$, and $consec(\tau_i, 9) = \sqrt[\infty]{10} = 1$ as the $interval(\tau_i, 9) = \infty$.

To compute the $fc(I, \Gamma^{\delta}, ns)$ fitness value, SWEAK runs APSSimulator ns times for I and obtains ns schedule scenarios $S_1, S_2, \ldots, S_{ns}$. For each schedule scenario S_h, we denote by $consec_h(\tau_i, k)$ the consecutiveness degree for the kth arrival of a task τ_i observed in S_h. SWEAK aims to maximize the $fc(I, \Gamma^{\delta}, ns)$ fitness function defined as follows:

$$fc(I, \Gamma^{\delta}, ns) = \sum_{h=1}^{ns} \Big(\max_{\tau_i \in \Gamma^{\delta}} \sum_{k=1}^{lk(\tau_i)} consec_h(\tau_i, k) \Big) / ns$$

Computational search. SWEAK employs the NSGA-II algorithm [122]. The algorithm first generates an initial population and evaluates it with the fitness functions defined above. The fitness values determine Pareto front rankings and sparsities of the solutions in the population. The algorithm then breeds the new population to produce the next generation's population using the following genetic operators: (1) *Selection* chooses candidate solutions as parents using a tournament selection technique, with the tournament size equal to two which is the most common setting [77]. (2) *Crossover* creates offspring from the selected parents using a modified version of the one-point crossover. (3) *Mutation* makes a random change in the offspring according to a mutation rate. Our crossover and mutation operators are defined below. After evaluating the new population, the algorithm produces the next population by considering the current population and the new population by selecting superior solutions that have higher front rankings and sparsity values. The algorithm continues to evolve the population until reaching the execution budget.

Crossover. A crossover operator produces offspring from two parent solutions by inheriting their characteristics. Our crossover operator, named SWEAKCrossover, modifies the standard one-point crossover operator [122], where it selects a random crossover point among all genes and swaps them between parent solutions based on the crossover point. However, in our context, as the size of two parents can differ, the random selection may produce invalid offspring. To prevent it, SWEAKCrossover selects a crossover point among the context switching times, i.e., λ_s, λ_x, and λ_p, or the first arrival of the aperiodic tasks in Γ. As the size of Γ and context switching times are fixed for all solutions, SWEAKCrossover can crossover two solutions with different sizes.

Figure 4.3 shows an example of SWEAKCrossover operation using a system with three aperiodic tasks, τ_1, τ_2, and τ_3. Let two parent solutions I_p and I_q be as follows: $I_p = \{0.007, 0.011, 0.001, (\tau_1, 5), \ldots, (\tau_2, 10), \ldots, (\tau_3, 6), (\tau_3, 15)\}$ and $I_q = \{0.008, 0.010, 0.001, (\tau_1, 4), \ldots, (\tau_2, 8), \ldots, (\tau_3, 4), \ldots, (\tau_3, 20)\}$, where (τ_i, t) states that task τ_i arrives at time t. Given the two parents I_p and I_q, SWEAKCrossover randomly selects a point— the first arrival of τ_2 in this example—and then it swaps the context switching times and all the arrivals of τ_1 between I_p and I_q. As shown in Figure 4.3, SWEAKCrossover then

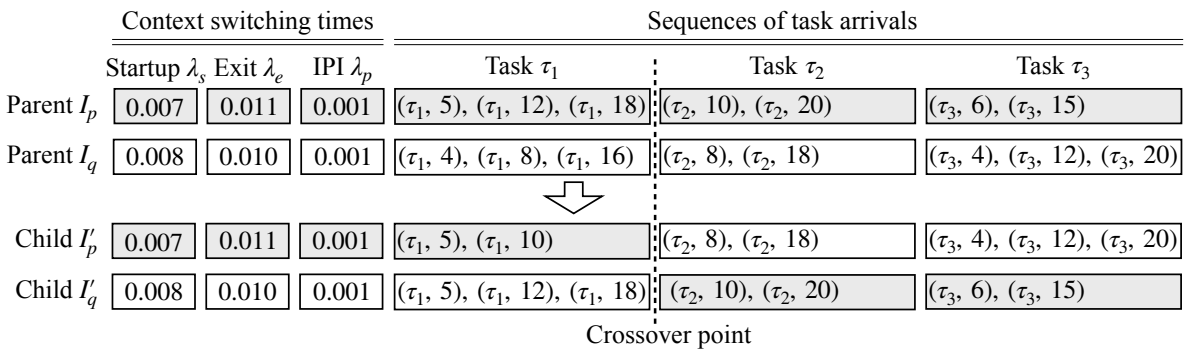


Figure 4.3: An example of a crossover operation for SWEAK. It swaps all context switching times and all task arrivals of task τ_1 between two parent solutions I_p and I_q to produce offspring I'_p and I'_q.

generates the offspring I'_p and I'_q as follows: $I'_p = \{0.007, 0.011, 0.001, (\tau_1, 5), \ldots, (\tau_2, 8), \ldots, (\tau_3, 4), \ldots, (\tau_3, 20)\}$ and $I'_q = \{0.008, 0.010, 0.001, (\tau_1, 4), \ldots, (\tau_2, 10), \ldots, (\tau_3, 6), (\tau_3, 15)\}$. The shaded (resp. unshaded) cells in Figure 4.3 indicate which task arrivals in child I'_q (resp. I'_p) come from which parent. Note that SWEAKCrossover only swaps context switching times when it selects the first aperiodic tasks as a crossover point, which are before the first arrival of τ_1 in this example.

Mutation. SWEAK uses a heuristic mutation algorithm called SWEAKMutation. For a solution I, SWEAKMutation mutates the context switching times or the kth task arrival time $a_{i,k}$ of an aperiodic task τ_i with a mutation probability. Regarding the context switching times, SWEAKMutation chooses a new time value from the range of each context switching time, λ_s, λ_x, and λ_p. Regarding arrivals of an aperiodic task τ_i, SWEAKMutation chooses a new arrival time value $a_{i,k}$ based on the $[T_i^{min}, T_i^{max}]$ inter-arrival time range of τ_i. If a mutation of the kth arrival time of τ_i does not affect the validity of the k+1th arrival time, the mutation operation ends. Specifically, let $a^*_{i,k}$ be a mutated value of $a_{i,k}$. In case $a_{i,k+1} \in [a^*_{i,k} + T_i^{min}, a^*_{i,k} + T_i^{max}]$, SWEAKMutation returns the mutated I solution.

After mutating the kth arrival time $a_{i,k}$ of a task τ_i in a solution I, if the k+1th arrival becomes invalid, SWEAKMutation corrects the remaining arrivals of τ_i. We denote by $a^*_{i,k}$ the mutated kth arrival time of τ_i. For all the arrivals of τ_i after $a^*_{i,k}$, SWEAKMutation first updates their original arrival time values by adding the difference $a^*_{i,k} - a_{i,k}$. Let $\mathbb{T} = [0, \mathbf{t}]$ be the scheduling period. SWEAKMutation then removes some arrivals of τ_i if they are mutated to arrive after $\mathbf{t}$ or adds new arrivals of τ_i while ensuring that all tasks arrive within $\mathbb{T}$.

Given the offspring presented in Figure 4.3, SWEAKMutation, for example, mutates a child solution $I'_q = \{0.008, 0.010, 0.001, (\tau_1, 4), (\tau_1, 8), (\tau_1, 16), \ldots, (\tau_3, 15)\}$. Let $[T_1^{min}, T_1^{max}] = [2, 8]$ be the inter-arrival time range of task τ_1, let $\mathbb{T} = [0, 22)$ be the time period during which APSSimulator receives task arrivals, and let us assume SWEAKMutation selects the second arrival of task τ_1, i.e., $(\tau_1, 8)$ in Figure 4.3, to mutate. Based on the inter-arrival time range of τ_1, SWEAKMutation randomly chooses a new arrival time, e.g., 6, for the second arrival of τ_1. The third arrival $(\tau_1, 16)$ of τ_1 then becomes invalid due to the mutated second arrival $(\tau_1, 6)$; i.e., τ_1 cannot arrive at time 16 because $16 \notin [6 + 2, 6 + 8]$, where $[T_1^{min}, T_1^{max}] = [2, 8]$. According to the correction procedure described above, the third arrival of τ_1 is modified to $(\tau_1, 14)$ as $14 = 16 + (6 - 8)$, where 16, 6, and 8 are, respectively, the original third arrival time of τ_1, the mutated second arrival time of τ_1, and the original second arrival time of τ_1. As APSSimulator can receive new arrivals of τ_1 after time 14, SWEAKMutation may add new arrivals of τ_1 based on its inter-arrival time range.

Note that for a system that consists of only periodic tasks, SWEAK will search for the worst test cases by varying context-switching times without changing sequences of task arrivals since the periodic tasks will have the same patterns of the task arrivals (see Section 4.3).

4.4.2 Simulation

The objective of the simulation step is to produce schedule scenarios given the input and a labeled dataset (training dataset) for the learning step. SWEAK uses a scheduling simulation technique to produce schedule scenarios since such simulation can generate a large number of tests for a lower cost than the actual scheduler an it can enable testing without the actual code for the tasks. Based on the simulation results, we generate a labeled data set for the learning step.

APSSimulator. A schedule simulator, named APSSimulator, simulates the behavior of APS according to the characteristics described in Section 4.2. As an input, the simulator takes a feasible solution I, which contains sequences A of task arrivals for a set Γ of tasks, context switching times ($\lambda_s,\ \lambda_x,\ \lambda_p$), and a set W of WCET values for the tasks. Given the sequences A of task arrivals, the simulator calculates when each task arrival will be completed with the given context switching times in a solution I and a set W of WCET values, as well as APS configurations, e.g., the window size for partitioning and the timeslice for round-robin (see Section 4.2). We set the APS configuration values following the guideline from the partner company. The simulation results are processed into a schedule scenario S.

Generating a labeled dataset. SWEAK requires a labeled dataset as it uses a supervised learning technique [152] to find a model to predict safe WCET ranges. Importantly, engineers want to have a certain level of confidence about safety. To achieve that, SWEAK applies logistic regression to obtain such interpretable probability. In our context, based on a given labeled dataset, the model reveals the relationship between WCET for all tasks and the schedulability of these tasks. The detailed learning step is explained in Section 4.4.3.

The labeled dataset, denoted by $\vec{L}$, is a list of tuples (W, ℓ) where W is a set of WCET values and ℓ is the label indicating the schedulability of a schedule scenario resulting from W. SWEAK generates a tuple (W, ℓ) for each APSSimulator run. For example, in the case of SWEAK calculates a fitness value for a feasible solution I, it appends ns tuples of (W, ℓ) to the labeled dataset $\vec{L}$ (see Section 4.4.1). To evaluate a feasible solution I, SWEAK runs APSSimulator ns times with the sampled sets $\{W_1, W_2, ...W_{ns}\}$. Each set W_h consists of a set of tuples (τ_i, C_i), where C_i is a randomly selected WCET value within a range $[C_i^{min}, C_i^{max}]$ for all tasks $\tau_i \in \Gamma$. Note that W_h has the same WCET value for the tasks that have a fixed WCET value. Given a feasible solution I and a set W_h of WCET values, APSSimulator produces a schedule scenario S_h. SWEAK then labels ℓ as *safe* when the schedule scenario S_h satisfies all the deadline constraints for the target tasks in Γ^{δ}; otherwise it labels ℓ as *unsafe*. Since schedule scenarios vary across the test cases, the labeled dataset $\vec{L}$ can contain instances that have different labels for the same set W of WCET values.

4.4.3 Learning logistic regression model

The objective of the learning step is to estimate safe ranges of WCET values under which target tasks are likely to be schedulable. To achieve the objective, SWEAK builds a model to predict safe WCET ranges using logistic regression [100]. This technique provides probabilistic interpretation and flexibility in selecting alternatives to predict safe WCET ranges. Figure 4.4 shows the overall process of the learning

step. We summarize each procedure in the following order: feature reduction, imbalance handling, model refinements (including sampling and simulation), and selecting WCET ranges. For more details on the learning step, see Chapter 3.

Feature reduction. Given the training data $\vec{L}$ during the search step, this procedure generates an equation f for the logistic regression. Logistic regression builds a model by inferring coefficients from a given equation. The equation f is formulated with the input variables such as WCET variables for the tasks in Γ of our dataset $\vec{L}$. Some variables have significant effects on predicting whether the label is *safe* or *unsafe*, while other variables do not. Since computational complexity increases when a large number of variables in the dataset are given, we remove insignificant variables.

SWEAK applies a feature reduction technique, random forest, widely used for dimensionality reduction [96, 141]. Given the labeled dataset $\vec{L}$, random forest builds a large number of decision trees to predict the label, i.e., *safe* and *unsafe* in our case, using a randomly selected subset of variables. The technique then derives the importance of each variable based on Gini impurity [35]. SWEAK selects important variables V that are above a particular threshold. Parameter values for feature reduction are explained in Section 4.5.5. Given the important variables V, SWEAK formulates an equation f for the logistic regression model using a second-order polynomial response surface model (RSM) [101] as follows:

$$\log \frac{p}{1-p} = c_0 + \sum_{i=1}^{|V|} c_i v_i + \sum_{i=1}^{|V|} c_{ii} v_i^2 + \sum_{i=1}^{|V|-1} \sum_{j=i+1}^{|V|} c_{ij} v_i v_j$$

where $v_i, v_j \in V$, p is the probability of violating deadline constraints, and c_0, c_i, c_{ii}, and c_{ij} are the coefficients that will be inferred by logistic regression. SWEAK also applies stepwise AIC (Akaike Information Criterion) [191] to the equation f, as the equation may contain redundant terms that are not related to predicting the label.

Imbalance handling. Supervised machine learning is highly dependent on the training dataset $\vec{L}$. As $\vec{L}$ is generated by search that aims to find the worst test cases (see Section 4.4.1), it tends to be imbalanced with more sets of WCET values violating deadline constraints. Imbalanced data may lead to unsatisfactory results for supervised machine learning. SWEAK handles the imbalance using the logistic regression model based on $\vec{L}$.

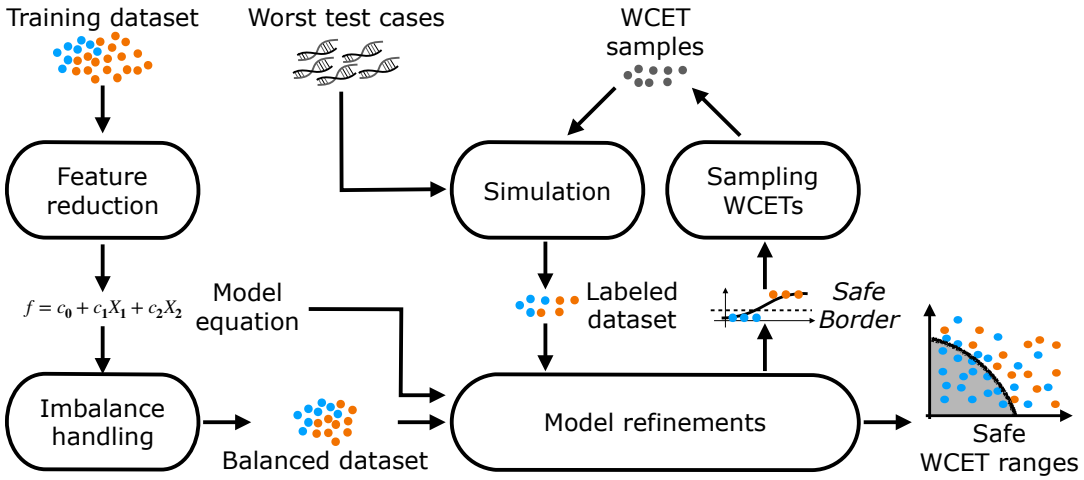


Figure 4.4: An overview of the learning step

SWEAK builds an initial model m from the training dataset $\vec{L}$ and the equation f. Figure 4.5 shows a model m on the WCET space for two tasks τ_1 and τ_2. The logistic regression technique estimates probabilities for *safe* and *unsafe*. For example, the gray area in Figure 4.5 represents the model area with the probability of violating deadline constraints in the range [0.0001, 0.9999]. SWEAK selects a probability p^u which is the minimum probability that classifies data instances as safe and which are labeled as safe in the dataset $\vec{L}$ (no false unsafe). SWEAK then calculates reduced WCET ranges [C_i^{min}, C_i'], where C_i' is the intercept between the WCET axis for τ_i and the initial model. The balanced dataset $\vec{L}^b$ is produced by pruning the data instances out of the reduced WCET ranges. Note that C_i' is equal to C_i^{max} when there is no intercept for a task τ_i.

Model refinements. Given the balanced dataset $\vec{L}^b$ and the equation f, SWEAK rebuilds a logistic regression model m. Then SWEAK finds a probability p^s that maximizes the *safe* area, while ensuring that all the data instances below the *safe* area are classified as *safe*, i.e., no false safe. We call this area, which is defined by the model m and the probability p^s, as *safe border*, e.g., p^s=0.01 in Figure 4.5. The *safe border* may over-fit the current dataset $\vec{L}^b$, which may lead to mis-classifications in different datasets. To alleviate over-fitting, SWEAK refines the *safe border* using a distance-based sampling method (see Section 3.4.2). The sampled WCET values are evaluated and labeled by the simulation step with the worst test cases given from the search step, which results in the new labeled dataset $\vec{L}^{new}$. SWEAK then rebuilds the *safe border* after merging $\vec{L}^b$ with $\vec{L}^{new}$. This refinement is repeated until either reaching the specified number of refinements (assigned analysis budget) or reaching an acceptable level of precision of the *safe border* according to standard precision metric [187].

Selecting WCET ranges. Given the *safe border*, safe WCET ranges are determined by selecting one point on that border. The *safe border* is a set of points that represents the upper bounds of safe WCET ranges. Engineers thus can find safe WCET ranges by choosing one point on the *safe border* depending on their system requirements. For example, assuming a black dot on the safe border in Figure 4.5 is the selected point, i.e., $[C_1, C_2]$, the safe WCET ranges become $[C_1^{min}, C_1]$ and $[C_2^{min}, C_2]$. However, engineers may not have such contextual information at early design stages. As the result, SWEAK suggests a point, named *best-size point*, on a *safe border* by maximizing the volume of the WCET ranges using Nelder-Mead algorithm [140]. We assume that a larger volume of the WCET ranges can provide safer WCET ranges as they provide wider space for tasks' WCET values. Note that SWEAK also provides the flexibility to select another point through trade-off analysis between tasks' WCET values.

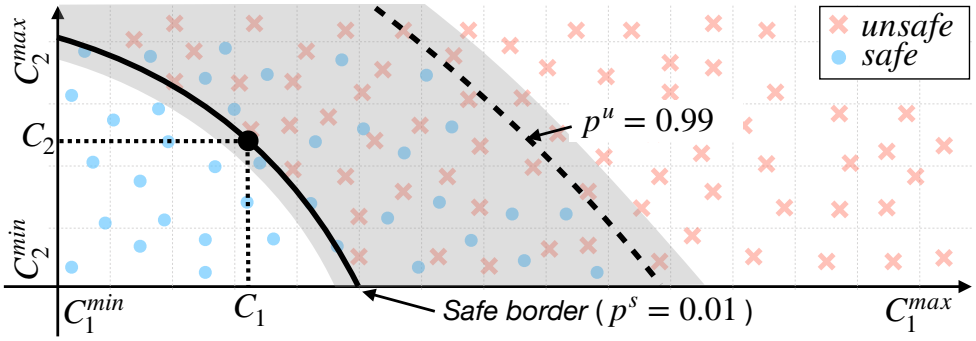


Figure 4.5: A logistic regression model on the WCET space for two tasks τ_1 and τ_2

4.5 Evaluation

In this section, we evaluate SWEAK by answering three research questions below. In our experiments, we apply SWEAK to an industrial study subject from the satellite domain and synthetic study subjects.

4.5.1 Research questions

RQ1. (baseline comparison): *Can SWEAK provide better options for WCET ranges than a baseline?* It is a general concept that comparing against a baseline, such as a random search-based approach, is an important for the sanity check [89, 12]. Hence, we compare SWEAK and a baseline to see if SWEAK can infer better WCET options than the baseline. We believe that SWEAK can outperform the baseline with a higher degree of confidence in estimating safe WCET ranges.
RQ2. (accuracy of probability): *How accurately does SWEAK infer a probability of violating deadline constraints?* Accurate estimation of probability ensures the reliability of the estimated WCET ranges. We compare probabilities computed by SWEAK with simulation-based ones that are calculated from a large number of simulations with varying WCET values within the estimated WCET ranges. Our conjecture is that SWEAK can infer conservative or similar probabilities to simulation-based probabilities, as SWEAK is relying on fine-tuning of the logistic regression step.
RQ3. (scalability): *Can SWEAK find safe WCET ranges for large-scale systems within a practical time budget?* It is challenging to estimate acceptable WCET ranges in large-scale systems because there exist complex task interactions caused by combinations of arrival sequences, priorities, context switching times, and WCET values. To assess the scalability of SWEAK in terms of execution time, we use a large number of synthetic systems that are generated with various characteristics. We expect that the execution time of SWEAK can be acceptable in practice.

4.5.2 Synthetic systems

A synthetic system is an artificially generated system accounting for the characteristics of real-time tasks which reflect actual systems in the real world. Such systems are used in many studies [62, 195, 59, 82, 176, 68] to evaluate their approaches as they can be used to complement a number of evaluation subjects. Algorithm 4.1 describes a procedure for generating a synthetic system by varying the key task parameters. The algorithm synthesizes a set of periodic tasks (lines 18-23) and sets a certain number of tasks of them to have weakly hard constraints (lines 24-25). The algorithm selects a set of tasks that have fixed periods and single WCET values in a single partition. The algorithm then modifies the system using the following functions: (1) converting some tasks to aperiodic tasks (lines 26-27), (2) transforming some tasks to have WCET ranges (lines 28-29), and (3) configuring partitions and assigning tasks to each partition (lines 30-31).

As shown in line 18 of Algorithm 4.1, the algorithm first creates a set $\mathbf{U}$ of task utilization values using the UUniFast-Discard algorithm [59], which is devised to give an unbiased distribution of task utilization values. The UUniFast-Discard algorithm takes as input the number of tasks to be synthesized, n, and a target utilization value, u^t. It then outputs n utilization values, $\{U_1, \ldots, U_n\}$, where $0 < U_i < 1$ for all U_i and $\sum_{i=1}^{n} U_i = u^t$. The maximum of target utilization u^t relies on the number of processing cores, i.e., the maximum target utilization is equal to the number of processing cores. For example, if a system uses two processing cores, the maximum value of u^t is 2.

Algorithm 4.1: An algorithm for generating synthetic systems subjected to weakly hard constraints, including partitioning.

```
1  Input n: number of tasks
2  Input u^t: target utilization per processing core
3  Input T^min: minimum task period
4  Input T^max: maximum task period
5  Input g: granularity of task periods
6  Input θ: maximum offset value
7  Input γ: ratio of aperiodic tasks
8  Input μ: range factor to determine inter-arrival times
9  Input ω: number of WCET ranges
10 Input λ: range factor to determine WCET ranges
11 Input ρ: number of partitions
12 Input (m, K): weakly hard constraint
13 Input nw: number of tasks that are subject to the weakly hard constraint
14 Output- Γ: set of tasks
15
16 Γ ← {}, C ← {}
17 // synthesize a set of periodic tasks
18 U ← UUniFast_discard(n, u^t) // task utilizations
19 T ← generate_task_set(n, T^min, T^max, g) //task periods
20 for each i ∈ [1, n] do
21     |  C ← C ∪ {U_i·T_i}, where U_i ∈ U and T_i ∈ T // WCETs
22 end for
23 Γ ← generate_task_periods(T, C, θ, g)
24 // select weakly hard real-time tasks
25 Γ ← set_weakly_hard_constraint(Γ, nw, (m, K))
26 // convert some periodic tasks to aperiodic tasks
27 Γ ← convert_to_aperiodic_tasks(Γ, γ, μ)
28 // convert some WCET point values to WCET ranges
29 Γ ← convert_to_WCET_ranges(Γ, ω, λ)
30 // assign partitions and partition budgets
31 Γ ← assign_partitions(Γ, ρ)
32 return Γ
```

As for line 19 of Algorithm 4.1, the algorithm generates n task periods, $T_1 \ldots T_n$ according to a log-uniform distribution within a range $[T^{min}, T^{max}]$, i.e., given a task period (random variable) T_i, $log\ T_i$ follows a uniform distribution. For example, when a period range $[T^{min}, T^{max}]$ is [10ms, 1000ms], the algorithm generates approximately an equal number of tasks in the period ranges [10ms, 100ms] and [100ms, 1000ms]. The parameter g is used to determine the granularity of period values as multiples of g. Lines 20-22 of Algorithm 4.1 describe how the algorithm synthesizes tasks' WCET values. Specifically, for each task τ_i, the algorithm computes the WCET value C_i of τ_i as $C_i = U_i \cdot T_i$.

Given the task periods $\mathbf{T}$ and the WCET values $\mathbf{C}$, line 23 of Algorithm 4.1 synthesizes a set Γ of periodic tasks accounting for offsets, priorities, and deadlines. A periodic task τ_i is characterized by a period T_i, a WCET C_i, an offset O_i, a priority P_i, and a deadline D_i (see Section 4.3). A task offset O_i is randomly selected from an input range $[0, \theta]$ of offset values. The algorithm applies a rate-monotonic scheduling policy [118] to assign task priorities, in which tasks that have longer periods are given lower priorities. This policy assumes that task deadlines are equal to their periods.

Given the system Γ, line 25 of Algorithm 4.1 assigns the specified weakly hard constraints (m, K) to nw tasks. Real-time tasks in a system can have different weakly hard constraints. However, for controlled experiments, we assume that tasks subjected to weakly hard constraints have the same constraints (see

Section 4.5.4). We select *nw* tasks from the lower priority tasks to associate them with the weakly hard constraint. Since lower priority tasks have higher chances of missing deadlines, this assumption allows us to know the effect of a weakly hard constraint.

Line 27 of Algorithm 4.1 selects some periodic tasks and converts them into aperiodic tasks according to the ratio γ of aperiodic tasks. The algorithm then uses a range factor μ to determine the minimum and maximum inter-arrival times of the aperiodic tasks. Specifically, for a task τ_i to be converted, the algorithm computes a range $[T_i^{min}, T_i^{max}]$ of inter-arrival times as $[T_i^{min}, T_i^{max}] = [T_i \times (1-\mu), T_i \times (1+\mu)]$, where $\mu \in (0, 1)$. For example, if $\mu = 0.45$ and $T_i = 50$ for a task τ_i to be converted, $[T_i^{min}, T_i^{max}] = [27.5, 72.5]$.

To synthesize tasks' WCET ranges, line 29 of Algorithm 4.1 randomly selects ω tasks in Γ to convert their WCET point values into WCET ranges. For a selected task τ_i, the algorithm computes a WCET range $[C_i^{min}, C_i^{max}]$ as $[C_i^{min}, C_i^{max}] = [C_i \times (1-\lambda), C_i \times (1+\lambda)]$, where λ is a range factor to determine the WCET ranges and $\lambda \in (0, 1)$. For example, if $\lambda = 0.25$ and $C_i = 10$ for a task τ_i, $[C_i^{min}, C_i^{max}] = [7.5, 12.5]$.

Regarding APS partitioning, line 31 of Algorithm 4.1 assigns tasks to partitions. The algorithm generates ρ partitions and assigns evenly distributed partition budgets. For example, when ρ=2, the budget distribution is [50%, 50%]. If ρ=3, the budget distribution is [34%, 33%, 33%]. The algorithm then randomly selects a partition to which a task will belong. Each partition has at least one task, and a task can be assigned to only one partition.

4.5.3 Study subjects

To evaluate our approach through RQ1 and RQ2, we use four case study subjects: ESAIL [113] (an industrial real-time system), and three synthetic systems that contain the APS characteristics described in Section 4.2. ESAIL does not have APS characteristics such as multi-partition, multi-policy, and multi-core. We thus generate a synthesized system (base) to build three different systems having the APS characteristics. We further describe the details of the systems below.

ESAIL is a commercial microsatellite developed by LuxSpace that tracks ships by broadcasting radio signals worldwide. The ESAIL management system is made up of 12 periodic tasks and 13 aperiodic tasks on a single core platform. During the design stages, 23 tasks were analyzed to estimate their WCET values as ranges that are varied from 0.1ms to 20s due to uncertain factors. Regarding deadline constraints, five aperiodic tasks have weakly hard constraints while the other tasks have hard constraints.

A base system Γ is generated using Algorithm 4.1 by modifying the function that converts WCET values to ranges. To synthesize the base system, we first generate a system with the following parameter values: (1) the number of tasks $n = 25$, the ratio of aperiodic tasks $\gamma = 0.5$, the range factor to determine inter-arrival times $\mu = 0.25$, and the maximum offset $\theta = 0$. These settings are decided based on the characteristics of the industrial system. (2) We set the minimum task period T^{min} = 10ms, the maximum task period T^{max} = 1s, and the granularity g = 10ms; these are commonly used in real-time systems [23]. (3) We set the target utilization $u^t = 0.9$ for a single-core system. This parameter is decided to make the system sometimes miss any deadline constraints [70]. (4) Regarding the number of APS partitions, we set $\rho = 1$. This parameter is decided for the base system to be simple so that it can easily be converted to other synthetic systems. (5) Regarding weakly hard constraints, we associate 10 low priority tasks with

Table 4.1: An example of creating WCET ranges from a WCET value according to a randomly selected ratio r from the log-uniform distribution in the range [0, 0.5]. Each WCET range $[C_i^{min}, C_i^{max}]$ is calculated by $C_i \times (1 \pm r_i)$.

	Task τ_1	Task τ_2	Task τ_3	Task τ_4	Task τ_5
WCET value (C_i)	1.5	2	5	10.5	3
Range ratio (r_i)	0.03	0.15	0.01	0.45	0.02
WCET range ($[C_i^{min}, C_i^{max}]$)	[1.5, 1.5]	[1.7, 2.3]	[5.0, 5.1]	[5.8, 15.2]	[2.9, 3.1]

with a weakly hard constraint ($nw = 10$). The weakly hard constraint is set to $(0, 10)$, i.e., hard deadline constraint, but it will be varied in our experiments.

With regard to converting WCET ranges, we use a different method of Algorithm 4.1 to make the system have different WCET ranges. This method uses random ratios selected from the log-uniform distribution for each WCET value of tasks in Γ. Table 4.1 shows an example of converting WCET values to WCET ranges for five tasks. A range ratio r_i for each task τ_i is randomly selected from log-uniform distribution in the range [0, 0.5], which means that the probability of r_i selected in the range [0, 0.1] is twice that of r_i selected in the range (0.1, 0.5]. Each WCET range $[C_i^{min}, C_i^{max}]$ is determined by $C_i \times (1 \pm r_i)$. This method makes a system have a small number of tasks with large WCET ranges and a large number of tasks with small WCET ranges, e.g., τ_2, τ_3, and τ_5 have small WCET ranges while τ_4 has a large WCET range as shown in Table 4.1. Note that some tasks can have no WCET ranges, such as τ_1 in Table 4.1. For those tasks, we assign a single WCET value instead of a WCET range.

If a system is composed of only periodic tasks, the minimum simulation time can be the LCM of the period values for all tasks [177]. However, the base system contains aperiodic tasks as well. To deal with such a system, we calculate the LCM of the period values of the periodic tasks and find the maximum value among the maximum inter-arrival times of the aperiodic tasks. We then set the minimum simulation time as the maximum value between those two values. The simulation time allows us to simulate all possible patterns of arrivals of periodic tasks including at least one arrival of aperiodic tasks. To reduce the execution time of the experiment, we generate a base system Γ that requires a minimum simulation time less than 5s.

Given the base system Γ, we synthesize the three systems described below by modifying Γ to account for the characteristics of APS following the guidelines from the partner company, Blackberry.

- *PARTITION*: This system has two APS partitions with 60% and 40% budgets for each partition. For efficient scheduling, a partition budget should be enough to execute all tasks in the partition and should not have much wastage. We assign tasks to each partition so that each partition budget is close to the total utilization of the tasks in the partition. In our evaluation, 19 tasks with higher priorities in Γ are assigned to the first partition. The remaining six tasks are assigned to the second partition.
- *POLICY*: This system contains two pairs of duplicate priorities. To make a pair, we randomly select two tasks from the given system Γ and assign the same priority and scheduling policy to the selected tasks. Each pair applies a different scheduling policy, e.g., the first pair uses FIFO, the second pair uses round-robin.
- *MULTICORE*: This system works on a two-core platform and assigns core affinities to some tasks. To make this system, we multiply WCET values (as well as the WCET ranges) by two for all tasks in Γ to

make the total utilization about 1.8. Recall that the maximum total utilization of a system is equal to the number of cores. We then assign core affinity to tasks using a random selection. For this system, we assign core 1 affinity to eight tasks, core 2 affinity to the other eight tasks, and no core affinity to the remaining nine tasks.

4.5.4 Experimental design

To answer the three RQs described in Section 4.5.1, we design three experiments EXP1, EXP2, and EXP3, respectively. We conduct EXP1 and EXP2 with the four case study subjects described in Section 4.5.3: ESAIL, PARTITION, POLICY, and MULTICORE. For EXP3, we experiment with 600 synthetic systems with different parameter settings. We describe each experiment in detail below.

EXP1. To answer RQ1, we implement a baseline that uses a random search (RS) without the learning step in SWEAK. The RS is a variant of the search step in SWEAK that does not use genetic operators, such as selection, crossover, and mutation, to breed offspring (see Section 4.4.1). Instead, the RS generates offspring by randomly creating new solutions. The same fitness functions evaluate the solutions in SWEAK. As the RS also uses the multi-objective functions, it employs the dominance and crowding distances for selecting Pareto fronts as well. During the search, a labeled dataset $\vec{L}$ is produced by our baseline, which contains tuples (W, ℓ) where W is a set of WCET values of tasks and ℓ is the label that classifies the simulation result with W as safe or unsafe. Once our baseline has the labeled dataset, it retrieves all tuples from the labeled dataset $\vec{L}$ to select a specific tuple (W_s, ℓ_s) that is labeled safe, $\ell_x = \mathit{safe}$, and maximizes the volume of the hyperbox defined by W_s. Note that W_s should satisfy the condition that any tuple (W_x, ℓ_x) contained in the hyperbox defined by W_s be safe, $\ell_x = \mathit{safe}$.

EXP1 compares the results obtained from SWEAK against the baseline. Recall Section 4.4.3 that SWEAK suggests the safe WCET ranges on the safe border by selecting a best-size point that maximizes its volume of the hyperbox. EXP1 compares the size of the volumes from the safe WCET ranges obtained by both approaches, SWEAK and the baseline. To analyze the weakly hard constraints, we apply both approaches to the subjects with different weakly hard constraints (m_i, K_i), where m_i is the tolerable number of deadline misses and K_i is the window size to check the deadline constraint (see Section 4.3). To do this, we vary m_i with a fixed K_i by assuming that all tasks having a weakly hard constraint are subjected to the same constraint. For example, if we set (m_i, K_i) = (2,10), the deadline constraints of every task τ_i in a subject Γ become (2,10). Note that we do not vary K_i because it does not affect the results since we are dealing with consecutive deadline misses.

EXP2. To answer RQ2, EXP2 calculates the empirical probability of the safe WCET ranges obtained from SWEAK. To this end, we first randomly sample multiple test cases including task arrivals and context switching times as well as the sample execution times within the estimated safe WCET ranges from each approach. We then simulate all combinations of the test cases and WCET values using APSSimulator and check for the presence of violating deadline constraints in each simulation result. The empirical probability is calculated as the number of simulations that violate deadline constraints over all combinations. We simulate 40000 times to obtain the probability of safe WCET ranges obtained by SAFE and the baseline. We also conduct EXP2 varying the tolerable number m_i of deadline misses to account for the effects of weakly hard constraints.

EXP3. To answer RQ3, we conduct EXP3 to assess the execution time of SWEAK with parameters that can affect the system characteristics. EXP3 is conducted by controlled experiments that vary each

parameter value while fixing the values of the other parameters so that we can reveal the correlations between the execution time and the parameters. We generate 600 synthetic systems based on Algorithm 4.1 with the six parameters and 10 variants for each as follows: (a) number of all tasks, $n \in \{5, 10, \cdots, 50\}$, (b) ratio of aperiodic tasks, $\gamma \in \{0.05, 0.10, \cdots, 0.50\}$, (c) number of WCET ranges, $\omega \in \{1, 2, \cdots, 10\}$, (d) number of processing cores, $\epsilon \in \{1, 2, \cdots, 10\}$, (e) number of APS partitions, $\rho \in \{1, 2, \cdots, 10\}$, and (f) simulation time, $\mathbf{t} \in \{5\text{s}, 10\text{s}, 15\text{s}, \cdots, 50\text{s}\}$. The number of all tasks n, the ratio of aperiodic tasks γ, and the number of WCET ranges ω are selected because they are necessary factors for executing SWEAK. The number of processing cores and the number of APS partitions are selected as they are adjustable by APS. We also include the simulation time as SWEAK relies on simulation.

To generate each synthetic system, we modify Algorithm 4.1 to account for APS partitions and partition budgets. After generating a set of tasks, we select the number ρ of APS partitions by evenly distributing a partition budget. For example, the partition budgets for each partition would be 25% when $\rho = 4$. We then randomly assign tasks to each partition, ensuring that one partition holds at least one task. Note that the total utilization of a generated synthetic system changes according to the number of processing cores. For example, when the input $u^t = 0.9$ and $\epsilon = 2$, the total utilization of the system becomes 1.8 (see Section 4.5.2).

While varying a parameter's value, we use fixed values for the other parameters including the uncontrolled parameters below. (1) We set the number of all tasks $n = 25$, the ratio of aperiodic tasks $\gamma = 0.50$, and the maximum offset $\theta = 0$. We set these values according to our industrial subject. (2) Regarding the task periods, we set the range $[T^{min}, T^{max}]$ of minimum and maximum periods to [10ms, 1s] with granularity $g = 10$ms. These values are commonly used in many real-time subjects [23]. (3) We set the range factor to determine inter-arrival times of aperiodic tasks $\mu = 0.25$, the number of WCET ranges ω=2, the range factor to determine WCET range $\lambda = 0.25$, and the target utilization per processing core $u^t = 0.9$. The values of each parameter are based on our preliminary experiments. They ensure that the executions of the synthetic systems examined in EXP3 can sometimes violate their deadline constraints, i.e., they contain both safe (positive) and unsafe (negative) data instances (see Section 4.4.3). (4) We set the number of processing cores ϵ and the number of APS partitions $\rho = 1$. These values are determined as a baseline to minimize their impact on the other parameters. (5) For the simulation time of APSSimulator (see Section 4.4.2), we assign a minimum simulation time of 5s to guarantee that any aperiodic task arrives at least once and that all possible arrivals of periodic tasks can be analyzed during that period.

Due to the randomness of our approach, we conduct our experiments multiple times, 50 times for EXP1 and EXP2 and 10 times for each parameter configuration of EXP3. To compare the multiple results, we perform a statistical comparison using the non-parametric Mann-Whitney U-test [125] and Vargha and Delaney's $\hat{A}_{12}$ [172]. The Mann-Whitney U-test tests the equality of two independent samples without assuming a distribution. If the test result is less than 0.05, we consider the two samples are significantly different. Vargha and Delaney's $\hat{A}_{12}$ is a measure of effect size that calculates the degree of difference between two samples, regardless of the size of the samples. If $\hat{A}_{12}$ is equal to 0.5, we consider the two samples are equivalent.

4.5.5 Implementation and parameter tuning

We set the following parameter values for running SWEAK and the baseline. For the NSGA-II search in SWEAK, we set the population size to 10, the crossover rate to 0.7, and the mutation rate to 0.2 based on

existing guidelines [93]. We set the number of iterations to 1000 since we observed that the fitness values reached a plateau after the iterations in our preliminary experiment. To calculate the fitness values, we ran APSSimulator 20 times (determined by the preliminary experiment) for each solution (I in Section 4.4.1). We found that this number was cost-effective in computing reasonable fitness values. For a random search in the baseline, we set the same population size and the number of APSSimulator runs but the number of iterations to 1500 to ensure that the baseline produces the same size of dataset $\vec{L}$ with SWEAK. Recall that our baseline only searches for the worst WCET ranges without the learning step (see Section 4.5.4).

To simulate study subjects, we set the simulation time to 60s for the ESAIL subject and 5s for other synthetic subjects. Such values are determined by the rule we defined in Section 4.5.3. To execute APSSimulator, we set the window size for partitioning to 100ms, which is the default value of APS. The timeslice for the round-robin is set to 4ms assuming that the processor tick interval is 1ms. According to the guidelines from our partner company, the timeslice is usually set to 4 times a processor tick interval.

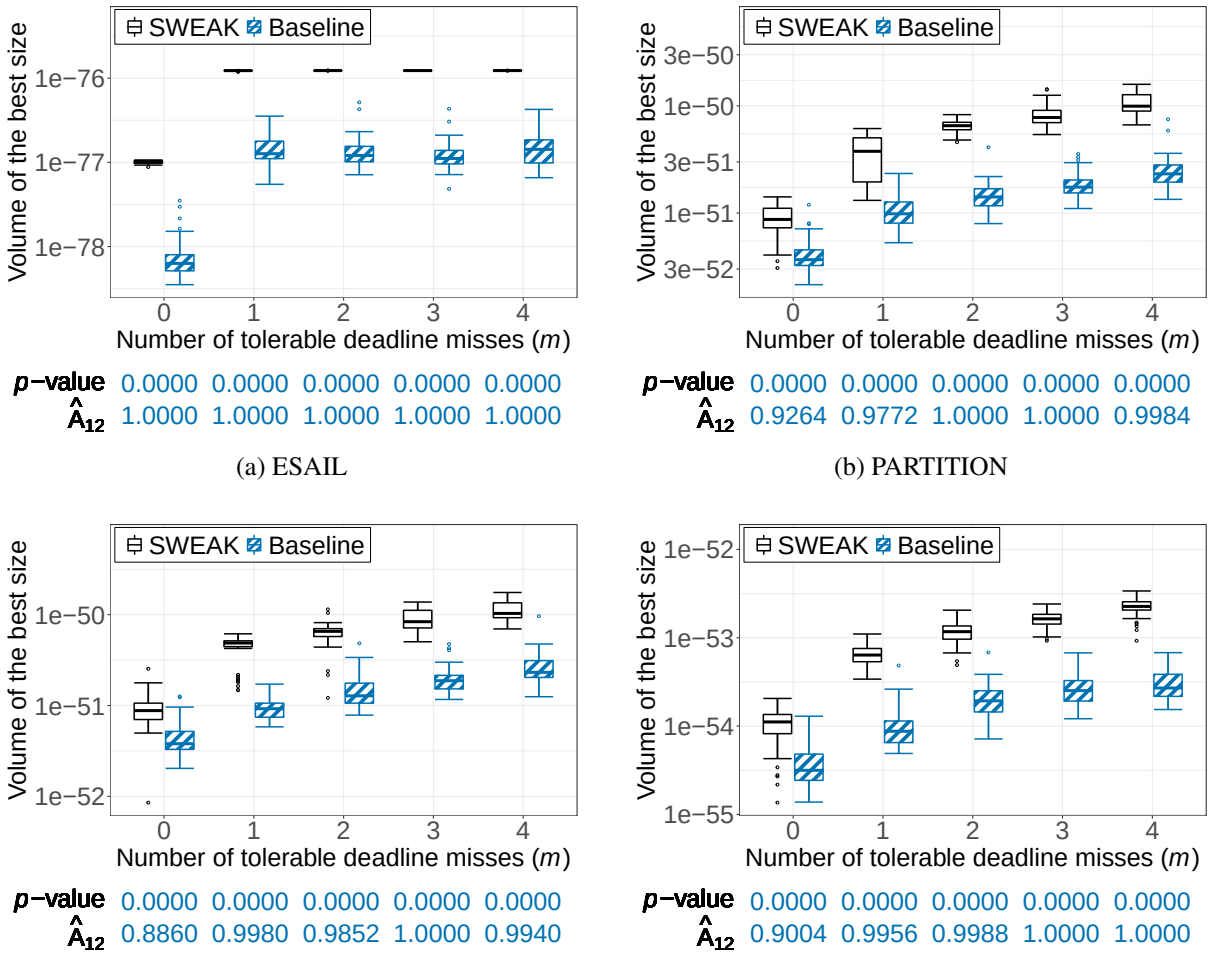
SWEAK has some parameters in the learning step for tuning the feature reduction and the model refinement. To reduce the features, we employed the random forest algorithm that includes the following parameters: (1) The tree depth was set to $\sqrt{|F|}$, where $|F|$ is the number of features, following the guidelines [91]. For example, since the ESAIL subject contains 23 features (i.e., WCET ranges), we assigned $\sqrt{|23|}$ to the tree depth of the subject (see Section 4.4.3). (2) The number of trees was set to 100 as we found that learning more than 100 trees did not provide further benefits for reducing the number of features in our preliminary experiments. Regarding the model refinement, we set the number of solutions to 10 and the number of model updates to 100. We observed that the precision of the model reaches an acceptable level with these parameters in our preliminary experiments.

Although the current parameter settings effectively support our conclusions, all the parameters can be further tuned to improve the approach. Note that we do not report further experiments on varying the parameters' values.

4.5.6 Results

RQ1. Figure 4.6 shows the results of EXP1, which compare the volumes of the hyperboxes defined by the WCET ranges estimated by SWEAK and Baseline. The comparisons are conducted with five different numbers of tolerable deadline misses, given by m, in weakly hard constraints for the following four study subjects: ESAIL (Figure 4.6a), PARTITION (Figure 4.6b), POLICY (Figure 4.6c), and MULTICORE (Figure 4.6d). Each boxplot in the figures shows a distribution (25%-50%-75% quartiles) obtained from 50 runs of SWEAK (or Baseline). The figure also reports p-values and $\hat{A}_{12}$ values of the results from 50 runs of SWEAK and Baseline.

As shown in Figure 4.6, SWEAK produces larger volumes of hyperboxes compared to Baseline for all the subjects. A larger volume provides greater flexibility in selecting appropriate WCET values, as such a hyperbox has wide WCET ranges. Regarding weakly hard constraints, SWEAK produces a larger volume of the hyperbox than Baseline when more relaxed weakly hard constraints are applied to the following subjects: PARTITION, POLICY, and MULTIFORE. Specifically, when the number of tolerable deadline misses is 1, the hyperbox volume is significantly larger than the hyperbox volume obtained when m is 0, while the degree of increasing volume is reduced when m is 2, 3, or 4. From this trend, it can be implied that when a deadline miss occurs, the subjects are likely to have more deadline misses, i.e., the probability of consecutive deadline misses is increased. Unlike the subjects discussed above, the hyperbox volumes



(a) ESAIL (b) PARTITION

(c) POLICY (d) MULTICORE

Figure 4.6: Distributions of the volumes of the hyperboxes that are defined by the safe WCET ranges obtained from SWEAK and Baseline for (a) ESAIL, (b) PARTITION, (c) POLICY, and (d) MULTICORE. The boxplots (25%-50%-75%) show the volumes of the hyperboxes obtained from 50 runs of SWEAK and Baseline.

of ESAIL are similar when the system is subjected to weakly hard constraints ($m \geq 1$), but not when the hard constraints ($m = 0$) is applied. This trend is caused by the characteristics of ESAIL that the lowest priority task in the system can easily starve to the worst test cases. Across all the subjects and weakly hard constraints, the experiment results obtained from SWEAK are statistically significant and superior to those obtained from Baseline (i.e., p-value < 0.05 and large effect sizes of $\hat{A}_{12} \approx 1.0$) with regard to the recommendation of safe WCET ranges. The average execution times of SWEAK and Baseline are 9.37h and 7.55h, respectively.

EXP1 also evaluates the estimated WCET ranges obtained from 50 runs of SWEAK and Baseline. We investigated each safe WCET range using 40000 simulation runs by varying test cases and WCET values within the estimated WCET ranges. Table 4.2 shows the number of simulation runs (out of 40000 runs) where any violation of deadline constraints occurred for the following subjects: ESAIL (4.2a), PARTITION (4.2b), POLICY (4.2c), and MULTICORE (4.2d). The tables present the maximum, median, minimum, or average values of the results obtained from 50 runs of SWEAK and Baseline. We vary the number of tolerable deadline misses m in the experiments. The p-values and $\hat{A}_{12}$ values show the differences between the results obtained from 50 runs of both approaches.

Table 4.2: Statistics of the number of simulation runs that violate any deadline constraints among the 40000 simulations with varying test cases and WCET values within the estimated WCET ranges obtained from SWEAK and Baseline. Each table compares the statistic values under different numbers of tolerable deadline misses in deadline constraints for (a) ESAIL (b) PARTITION, (c) POLICY, and (d) MULTICORE.

		Number of tolerable deadline misses (m)				
		0	1	2	3	4
SWEAK	Max	19.00	0.00	0.00	0.00	0.00
	Median	2.00	0.00	0.00	0.00	0.00
	Min	0.00	0.00	0.00	0.00	0.00
	Average	3.80	0.00	0.00	0.00	0.00
Baseline	Max	574.00	0.00	0.00	0.00	0.00
	Median	0.00	0.00	0.00	0.00	0.00
	Min	0.00	0.00	0.00	0.00	0.00
	Average	19.24	0.00	0.00	0.00	0.00
	p-value	0.1269	1.0000	1.0000	1.0000	1.0000
	$\hat{A}_{12}$	0.5830	0.5000	0.5000	0.5000	0.5000

(a) ESAIL

		Number of tolerable deadline misses (m)				
		0	1	2	3	4
SWEAK	Max	17.00	13.00	2.00	2.00	3.00
	Median	0.00	0.00	0.00	0.00	0.00
	Min	0.00	0.00	0.00	0.00	0.00
	Average	2.60	0.90	0.06	0.08	0.10
Baseline	Max	409.00	333.00	128.00	180.00	345.00
	Median	76.00	10.50	7.00	2.00	1.50
	Min	0.00	0.00	0.00	0.00	0.00
	Average	111.36	42.00	15.68	13.68	15.94
	p-value	0.0000	0.0000	0.0000	0.0000	0.0000
	$\hat{A}_{12}$	0.0196	0.1604	0.1280	0.1840	0.2378

(b) PARTITION

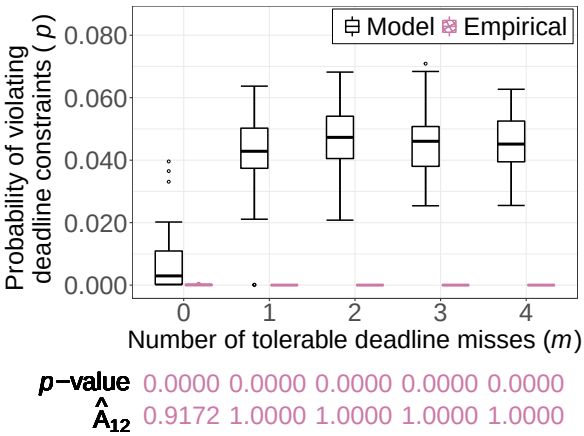
		Number of tolerable deadline misses (m)				
		0	1	2	3	4
SWEAK	Max	29.00	20.00	1.00	1.00	0.00
	Median	1.50	0.00	0.00	0.00	0.00
	Min	0.00	0.00	0.00	0.00	0.00
	Average	3.48	1.14	0.04	0.08	0.00
Baseline	Max	705.00	199.00	137.00	46.00	216.00
	Median	56.00	7.00	2.50	1.00	0.00
	Min	0.00	0.00	0.00	0.00	0.00
	Average	125.70	18.46	15.44	7.18	15.16
	p-value	0.0000	0.0000	0.0000	0.0000	0.0000
	$\hat{A}_{12}$	0.1024	0.2130	0.1988	0.2316	0.3000

(c) POLICY

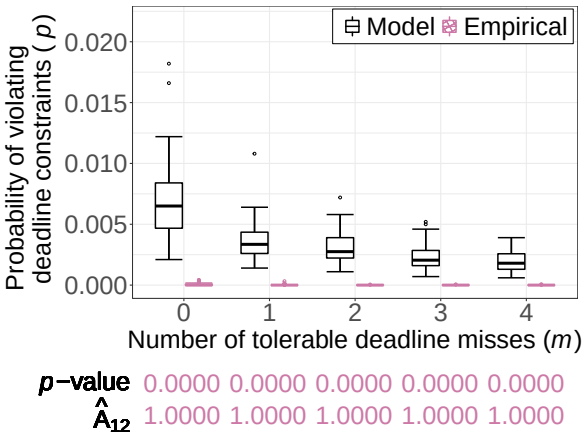
		Number of tolerable deadline misses (m)				
		0	1	2	3	4
SWEAK	Max	90.00	2.00	0.00	0.00	0.00
	Median	0.00	0.00	0.00	0.00	0.00
	Min	0.00	0.00	0.00	0.00	0.00
	Average	2.60	0.08	0.00	0.00	0.00
Baseline	Max	720.00	385.00	216.00	108.00	411.00
	Median	37.00	4.00	6.00	2.00	0.00
	Min	0.00	0.00	0.00	0.00	0.00
	Average	104.58	27.26	18.98	11.02	13.70
	p-value	0.0000	0.0000	0.0000	0.0000	0.0000
	$\hat{A}_{12}$	0.0934	0.1516	0.1500	0.2000	0.2600

(d) MULTICORE

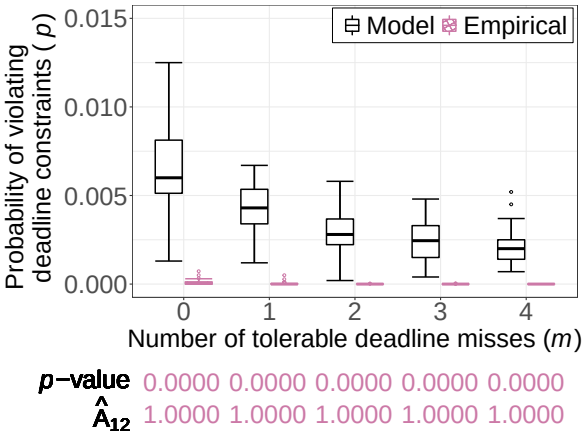
As shown in Table 4.2, SWEAK is statistically better (p-values are less than 0.05) than Baseline across all the numbers of tolerable deadline misses m in the PARTITION, POLICY, and MULTICORE subjects. The $\hat{A}_{12}$ values are also much lower than 0.5. Specifically, SWEAK has small variations in the number of simulation runs that violate deadline constraints (i.e., the differences between maximum and minimum values), while Baseline has large variations. Regarding the ESAIL subject, both SWEAK and Baseline show similar results, with p-values are greater than 0.05. However, Baseline has a large variation in the number of simulation runs that violate deadline constraints when $m = 0$. Note that the p-values in both SWEAK and Baseline with weakly hard constraints (i.e., $m \geq 1$) are 1.0 because both approaches



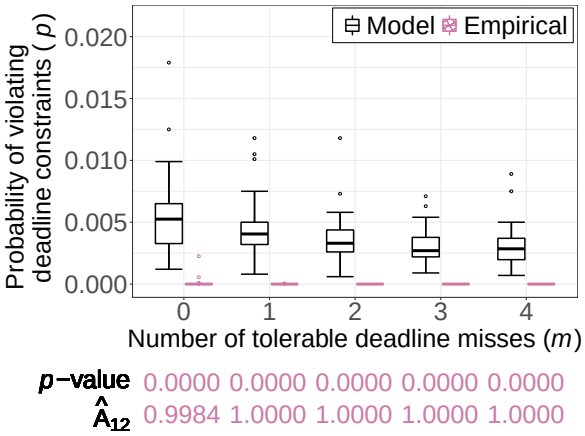
(a) ESAIL



(b) PARTITION



(c) POLICY



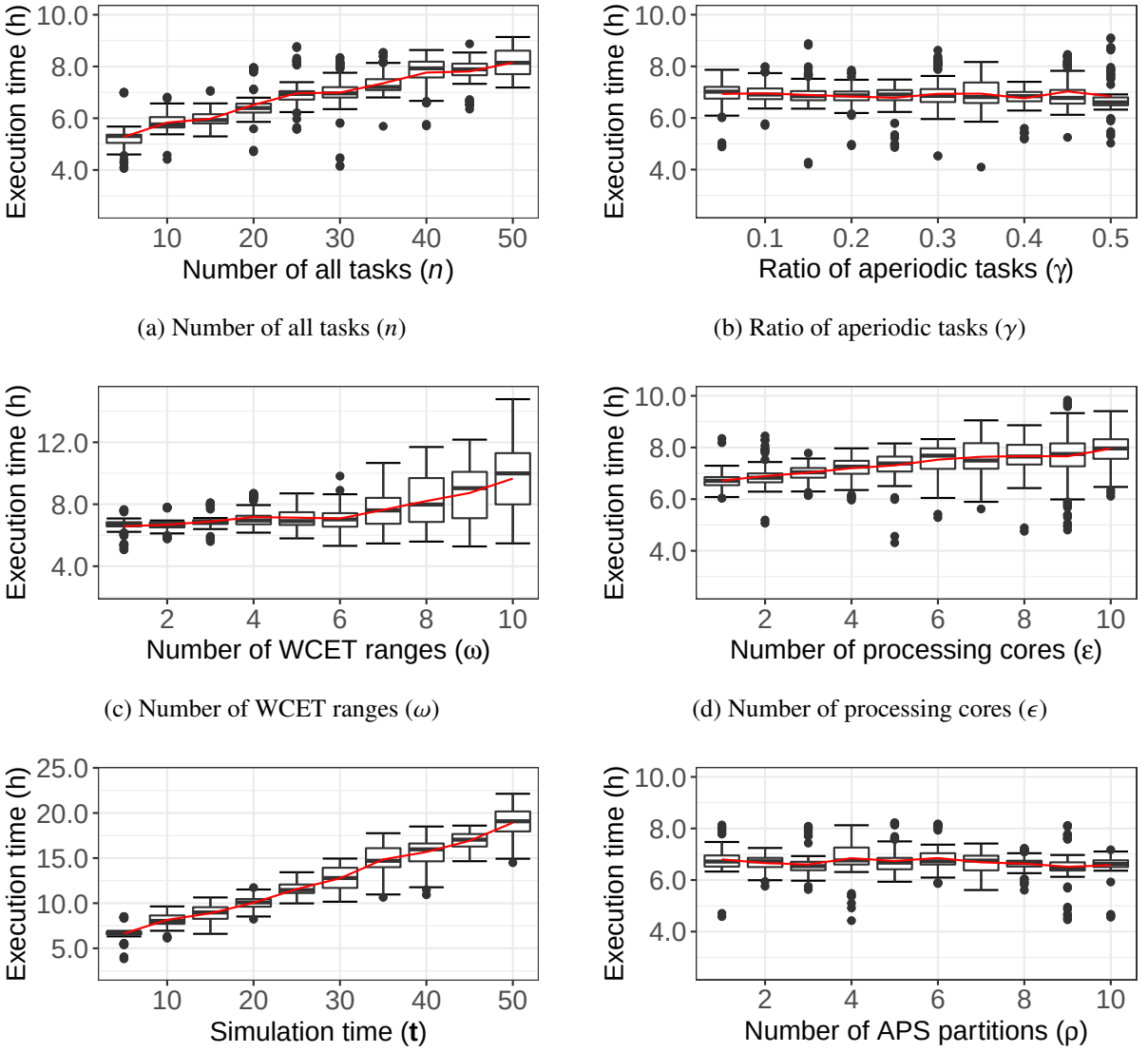
(d) MULTICORE

Figure 4.7: Distributions of empirical probabilities and model probabilities across different numbers of tolerable deadline misses m in (a) ESAIL, (b) PARTITION, (c) POLICY, and (d) MULTICORE. The boxplots (25%-50%-75%) show distributions of probabilities obtained from 50 runs of SWEAK and Baseline.

produce similar WCET ranges (see Figure 4.6). The results indicate that the estimated WCET ranges computed by SWEAK are more reliable than those computed by Baseline.

The answer to **RQ1** *is that* SWEAK significantly outperforms the baseline approach with respect to maximizing the hyperbox volume of WCET ranges under weakly hard constraints. The estimated WCET ranges that use SWEAK have a smaller chance of violating deadline constraints on average than the baseline approach. SWEAK takes 9.37h on average to estimate the WCET ranges, while the baseline takes 7.53h, which indicates that SWEAK is acceptable for use in practice as an offline analysis tool.

RQ2. Figure 4.7 depicts the results of EXP2 for the following subjects: ESAIL (Figure 4.7a), PARTITION (Figure 4.7b), POLICY (Figure 4.7c), and MULTICORE (Figure 4.7d). Each sub-figure compares the model probability and empirical probability obtained from the 50 runs of SWEAK by varying the number of tolerable deadline misses m. Each boxplot in the figures shows the distributions (25%-50%-75% quartiles) of model probabilities or empirical probabilities. As shown in Figure 4.7, the empirical probabilities for all values of m and for all the subjects, are significantly smaller than the model



(a) Number of all tasks (n)

(b) Ratio of aperiodic tasks (γ)

(c) Number of WCET ranges (ω)

(d) Number of processing cores (ϵ)

(e) Simulation time (**t**)

(f) Number of APS partitions (ρ)

Figure 4.8: Execution times of SWEAK when varying the values of the following parameters: (a) number of all tasks n, (b) ratio of aperiodic tasks γ, (c) number of WCET ranges ω, (d) number of processing cores ϵ, (e) simulation time **t**, and (f) number of APS partitions ρ. Each boxplot (25%-50%-75%) shows the distribution of 100 execution time values measured from 10 runs of SWEAK for 10 synthetic systems with the same configuration. The red line in each figure represents the trend of mean values of the execution times over each parameter value change.

probabilities, as all the p-values are 0 and all the $\hat{A}_{12}$ values are approximately 1. SWEAK infers a logistic regression model with a probability of violating deadline constraints based on the labeled dataset evaluated by the worst test cases. Therefore, the inferred model can be sensitive to the worst test cases when the model is too fit to it. However, the model probabilities are more conservative (higher) than the empirical probabilities that are evaluated by random test cases and random WCET values within the best-point WCET ranges from the models. The results indicate that we can expect the actual probability of violating deadline constraints to be lower than the model probability determined by SWEAK.

The answer to **RQ2** *is that* SWEAK estimates significantly conservative probabilities of violating deadline constraints compared to the empirical probabilities computed using simulations.

RQ3. Figure 4.8 shows the results of EXP3 that describe the distributions (boxplots) of execution times obtained from 10×10 runs of SWEAK, i.e., 10 runs for each synthetic system with the same experimental setting (see Section 4.5.4). The red solid lines represent the changes in mean values of the execution times while varying the following control parameters: (a) number of all tasks n, (b) ratio of aperiodic tasks γ, (c) number of WCET ranges ω, (d) number of processing cores ϵ, (e) simulation time $\mathbf{t}$, and (f) number of APS partitions ρ. The experiments in EXP3 took 22.1 hours at most, which is acceptable as an offline analysis technique. As shown in Figures 4.8a, 4.8d, and 4.8e, there is a positive linear relationship between its execution times and the parameters, n, ϵ, and $\mathbf{t}$, in SWEAK. Thus, we expect SWEAK to scale well as the number of tasks, the number of processing cores, and the simulation time increase. However, the parameters such as γ (Figure 4.8b) and rho (Figure 4.8f), they have no correlation with execution time. We note that the parameter ω (Figure 4.8c) is quadratically correlated with the execution time of SWEAK.

The quadratic correlation is mainly attributed to the number of terms in the model equation used in logistic regression (see Section 4.4.3). To build the equation, we leverage the second-order polynomial RSM that contains linear, quadratic, and two-way interaction terms. Given the number of WCET ranges ω, the number of RSM terms is determined by $1 + \omega + \omega + \binom{\omega}{2}$ including constants. The logistic regression algorithm infers the coefficients for each term of RSM, and the algorithm is used during the execution of SWEAK. As the execution time of logistic regression is linearly related to the number of coefficients [100], the execution time of SWEAK is quadratically correlated with the number of WCET ranges. Moreover, as the number of WCET ranges ω increases, the distribution of execution time becomes wider, as can be seen in Figure 4.8c. We found that this phenomenon occurs due to feature reduction and stepwise regression in the learning step (see Section 4.4.3). These techniques output an equation consisting of terms that are considered highly related to the violation of deadline constraints. This output affects the execution time for sampling WCET values (distance-based) and building logistic regression models. Depending on the system characteristics, the synthetic systems generated by setting $\omega = 10$ have a more diverse equations than those generated by setting $\omega = 1$.

The answer to **RQ3** *is that* SWEAK's execution time is linearly related to the number of tasks, the number of processing cores, and the simulation time. However, the execution time has a quadratic correlation with the number of tasks when the WCETs are defined as ranges. Overall, SWEAK is applicable in practice since it took 22.1h at most, in our experiments, which is generally acceptable.

Usefulness of SWEAK from the perspective of practitioners. To understand the practical usefulness of SWEAK, we discussed it with two actual practitioners in the institution (Blackberry) with whom we are collaborating. The feedback from the practitioners is as follows: (1) practitioners perceived that the worst test cases SWEAK produces help them conduct further analysis of their systems' schedulability, (2) practitioners agreed that SWEAK enables trade-off analysis by training a machine learning model, and (3) practitioners observed that SWEAK can be applied to various systems since it utilizes a simulator that mimics the behavior of an industrial scheduler and supports weakly hard constraints.

Existing studies on subjects such as analytical schedulability analysis techniques [118, 31, 128] have focused only on determining whether a system is schedulable or not. In addition, they do not support additional information about occurring deadline misses, such as test cases. SWEAK, on the other hand, yields worst-case (e.g., unschedulable) test cases that help practitioners investigate possible ways to improve their systems. Moreover, SWEAK's safe WCET ranges can offer some insights into deadline miss scenarios and the corresponding conditions.

The practitioners explained that the WCET values of some tasks are inherently unpredictable and are usually estimated in a rough and conservative way. They also mentioned that estimating safe WCET ranges is very difficult. As SWEAK systematically estimates safe WCET ranges with a probabilistic guarantee, the practitioners agreed that SWEAK can provide guidelines for improving their systems. Practitioners are able to select safe system-specific WCET ranges from the (infinite) WCET ranges which are modeled by the safe border, rather than the simple selection of the best-size WCET range which is automatically suggested by SWEAK. This flexibility allows them to perform domain-specific trade-off analysis among possible WCET ranges and helps them in decision-making for their task design.

As a company that develops an ROTS, Blackberry has a long-term goal of providing its customers with a schedulability analysis tool for systems with heterogeneous characteristics. SWEAK can be selected as one candidate solution, as it is not only an industrial simulation-based approach but it also deals with tasks that have hard and weakly hard constraints. These characteristics allow engineers to adopt SWEAK to analyze various systems. Although we have not conducted user studies, given the positive feedback from Blackberry, we believe SWEAK can be practically applicable and is worthy of further research.

4.5.7 Threats to validity

Internal validity. The internal validity of our experiments can be raised from the randomness of our approach. To mitigate this threat, we compared SWEAK against a baseline under identical parameter settings (e.g., the same number of WCET samples). We also ran both approaches 50 times for each experiment setting and provided the results with the statistical comparisons using the Mann-Whitney U-test and Vargha and Delaney's $\hat{A}_{12}$.

Another internal validity that can be raised is related to the configuration of the experiment. As SWEAK uses a multi-objective search algorithm, there are many parameters that need to be optimized to find proper solutions. However, these values could be applied differently depending on the subjects. This could also be a research topic for future study. With this in mind, we applied these parameters following the values that are commonly used in software engineering [93] and conducted preliminary experiments for some parameters, such as the number of iterations.

External validity. The main threat to external validity is that our results may not be generalizable to other contexts. SWEAK is applied to the systems that the WCET ranges are estimated by practitioners at early stages. However, as we discussed in Section 4.1, estimating WCET values is a hard-problem, even in the later stage of development, and the values are resulted in ranges. Thus, SWEAK is also available for use at later stage of development for testing schedulability of systems and for providing more precise WCET ranges. In addition, SWEAK was evaluated with an industrial system from the satellite domain. To support the lack of industrial subjects, we evaluated SWEAK by applying it to a large number of synthetic systems that vary their characteristics, including those of the industrial scheduler according to the industrial guidelines. We believe that these synthetic systems reflect real industrial systems. We have made the synthetic systems available online [111].

4.6 Related works

This section discusses previous studies related to WCET estimation in real-time systems and schedulability analysis with weakly hard constraints and systems under APS. To the best of our knowledge, there is no

previous work that probabilistically estimates WCET ranges accounting for weakly hard constraints by using industrial simulations.

WCET estimation in real-time systems. Measurement-based approaches are commonly used in practice [3]. The basic concept of this method is to run a large number of executions on the targeted hardware or an accurate simulator based on some inputs [183]. To obtain the input data, the method analyzes source code execution paths including sub-tasks and partition the execution paths. Owing to the difficulty of finding the worst-case inputs and the WCET value, Burns and Edgar [38] proposed a probabilistic WCET estimation approach that applies statistical analysis. Since after the initial study, measurement-based approaches have shown significant progress [85, 51, 26, 2]. However, as they need an executable source code and the target hardware, these approaches are only available for systems at later stages of development. To remove the hardware dependencies, static analysis approaches [73, 169, 133, 86] have been proposed. These approaches commonly estimate WCET values based on the abstract timing model of the target hardware and the software structure analysis. The main research strands are modeling cache behaviors [169, 133, 86] and building a timing model for hardware instructions [84, 7, 33]. However, these approaches still requires source code implementation. By contrast, SWEAK is an approach for the early stages of development. In contrast to the approaches reported in previous studies, SWEAK uses roughly estimated WCET ranges as inputs. SWEAK restricts the WCET ranges, i.e., whether a task is schedulable or not, with a selected probability of violating deadline constraints. This approach relies on an adapted genetic algorithm, a simulation, feature reduction, a dedicated sampling strategy, and logistic regression. Additionally, SWEAK supports the trade-off analysis of the WCET values of tasks and also allows practitioners to choose their desired probability of violating deadline constraints.

Schedulability analysis with weakly hard constraints. To ensure the quality of service (QoS), schedulability should be analyzed with weakly hard constraints [27]. There are approaches [189, 143] that analyze schedulability of weakly hard real-time systems. Xu et al. [189] proposed a deadline miss model for analyzing the number of deadline misses within consecutive task arrivals when a task is under unexpected overload. Pazzaglia et al. [143] presented a generalized schedulability analysis tool for weakly hard real-time systems by accounting for free offsets and release jitters based on a mixed integer linear programming formulation. Instead of mathematical schedulability analysis, SWEAK provides probabilistic guarantees of satisfying deadline constraints by inferring logistic regression models based on search and simulation outputs; thus, it complements the aforementioned research strands on weakly hard real-time systems.

Schedulability analysis of systems under APS. As the usage of QNX Neutrino has increased, a few APS-related analysis have recently been started [53, 52]. Dasari et al. [53] found that APS has drawbacks in partition configurations. Then, they provided some guidelines to help engineers properly configure partitions. Dasari et al. [52] investigated the APS behavior in practice and proposed a technique that verifies the end-to-end delay (i.e., schedulability) of event chains, which are the sequence of tasks activated by an event, on a real-time system using APS. The model employs a response time analysis of the chains. SWEAK complements APS-related research strand by providing analysis tools for schedulability at early stages so that engineers can design their systems with safe WCET ranges under a certain level of probabilistic guarantee.

4.7 Conclusion

This chapter introduced SWEAK, which estimates safe WCET ranges of tasks in a weakly hard real-time system at early design stages. SWEAK employs a genetic algorithm to search for the worst test cases that maximize the magnitude of deadline misses and the degree of consecutive deadline misses. Based on the search results, SWEAK infers safe WCET ranges using a logistic regression model. The inferred WCET ranges maximize the volume of the ranges with a conservative probability of violating deadline constraints. Consequently, we evaluated SWEAK on a mission-critical real-time satellite system and synthetic systems generated by following the guidelines provided by our industrial partner. The results indicate that SWEAK infers conservative WCET ranges that provide high flexibility in selecting WCET ranges for practitioners. We further evaluated SWEAK through 600 synthetic systems, varying their degree of complexity. The results show that SWEAK scales to complex systems. SWEAK completed all experiments within at most 22.1h. Hence, SWEAK is acceptable as an offline analysis technique in practice for estimating WCET ranges with a conservative probability of violating deadline constraints.

For future directions of this research, we plan to develop a real-time task modeling language that facilitates schedulability analysis and represents dependencies, constraints, and system behaviors. In addition, the usefulness of SWEAK should be further validated with other case studies from different domains.

Chapter 5

Optimal Priority Assignment for Real-Time Systems: A Coevolution-Based Approach

5.1 Introduction

Mission-critical systems are found in many different application domains, such as aerospace, automotive, and healthcare domains. The success of such systems depends on both functional and temporal correctness. For functional correctness, systems are required to provide appropriate outputs in response to the corresponding stimuli. Regarding temporal correctness, systems are supposed to generate outputs within specified time constraints, often referred to as deadlines. The systems that have to comply with such deadlines are known as real-time systems [119]. Real-time systems typically run multiple tasks in parallel and rely on a real-time scheduling policy to decide which tasks should have access to processing cores, i.e., CPUs, at any given time.

While developing a real-time system, one of the most common problems that engineers face is the assignment of priorities to real-time tasks in order for the system to meet its deadlines. Based on priorities of real-time tasks, the system's task scheduler determines a particular order for allocating real-time tasks to processing cores. Hence, a priority assignment that is poorly designed by engineers makes the system scheduler execute tasks in an order that is far from optimal. In addition, the system will likely violate its performance and time constraints, i.e., deadlines, if a poor priority assignment is used.

In real-time systems, the problem of optimally assigning priorities to tasks is important not only to avoid deadline misses but also to maximize *safety margins* from task deadlines and is subject to *engineering constraints*. Tasks may exceed their expected execution times due to unexpected interrupts. For example, it is infeasible to test an aerospace system exhaustively on the ground such that potential environmental uncertainties, e.g., those related to space radiations, are accounted for. Hence, engineers assign optimal priorities to tasks such that the remaining times from tasks' completion times to their deadlines, i.e., safety margins, are maximized to cope with potential uncertainties. Furthermore, engineers

typically have to account for additional engineering constraints, e.g., they assign higher priorities to critical tasks that must always meet their deadlines compared to the tasks that are less critical or non-critical.

A brute force approach to find an optimal priority assignment would have to examine all $n!$ distinct priority assignments, where n denotes the number of tasks. Furthermore, for a given priority assignment, schedulability analysis is, in general, known as a hard problem [20], which determines whether or not tasks will always complete their executions within their specified deadlines. Thus, optimizing priority assignments is also a hard problem because the space of all possible system states to explore in order to find optimal priority assignments is very large. Most of the prior works on optimizing priority assignments provide analytical methods [76, 115, 19, 56, 46, 58, 55], which rely on well-defined system models and are very restrictive. For example, they assume that tasks are independent, i.e., tasks do not share resources [61, 197]. Industrial systems, however, are typically not compatible with such (simple) system models. In addition, none of the existing work addresses the problem of optimizing priority assignments by simultaneously accounting for multiple objectives, such as safety margins and engineering constraints, as discussed above.

Search-based software engineering (SBSE) has been successfully applied in many application domains, including software testing [180, 179, 117, 14, 158], program repair [181, 168, 1], and self-adaptation [9, 44, 157], where the search spaces are very large. Despite the success of SBSE, engineering problems in real-time systems have received much less attention in the SBSE community. In the context of real-time systems, there exists limited work on finding stress test scenarios [36] and predicting worst-case execution times [113], which complements our work.

In practice, priority assignments result from an interactive process between the development and testing teams. While developing a real-time system, developers assign priorities to real-time tasks in the system and then testers stress the system to check whether or not the system meets its specified deadlines. If testers find a problematic condition under which any of the tasks violates its deadline, developers have to modify the priority assignment to address the problem. The back-and-forth between the development and testing teams continues until a priority assignment that does not lead to any deadline miss is found or the one that yields the least critical deadline misses is identified. The process is, however, not automated.

In this chapter, we use metaheuristic search algorithms to automate the process of assigning priorities to real-time tasks. To mimic the interactive back-and-forth between the development and testing teams, we use competitive coevolutionary algorithms [122]. Coevolutionary algorithms are a specialized class of evolutionary search algorithms. They simultaneously coevolve two populations (also called species) of (candidate) solutions for a given problem. They can be cooperative or competitive. Such competitive coevolution is similar to what happens in nature between predators and preys. For example, faster preys escape predators more easily, and hence they have a higher probability of generating offspring. This impacts the predators, because they need to evolve as well to become faster if they want to feed and survive [130]. Hence, the two species, i.e., predators and preys, have coevolved competitively. We note that no species has the competing traits of predators and preys simultaneously as such species could not evolve to survive. In our context, priority assignments defined by developers can be seen as preys and stress test scenarios as predators. The priority assignments need to evolve so that stress testing is not able to push the system into breaking its real-time constraints. Dually, stress test scenarios should evolve to be able to break the system when there is a chance to do so.

Contributions. We propose an Optimal Priority Assignment Method for real-time systems (OPAM). Specifically, we apply multi-objective, two-population competitive coevolution [146] to address the problem of finding near-optimal priority assignments, aiming at maximizing the magnitude of safety margins from deadlines and constraint satisfaction. In OPAM, two species relate to priority assignment and stress testing coevolve synchronously, and compete against each other to find the best possible solutions. We evaluated OPAM by applying it to six complex, industrial systems from different domains, including the aerospace, automotive, and avionics domains, and several synthetic systems. Our results show that: (1) OPAM finds significantly better priority assignments compared to our baselines, i.e., random search and sequential search, (2) the execution time of OPAM scales linearly with the number of tasks in a system and the time required to simulate task executions, and (3) OPAM priority assignments significantly outperform those manually defined by engineers based on domain expertise.

We note that OPAM is the first attempt to apply coevolutionary algorithms to address the problem of priority assignment. Further, it enables engineers to explore trade-offs among different priority assignments with respect to two objectives: maximizing safety margins and satisfying engineering constraints. Our full evaluation package is available online [109].

Organization. The remainder of this chapter is structured as follows: Section 5.2 motivates our work. Section 5.3 defines our specific problem of priority assignment in practical terms. Section 5.4 discusses related work. Sections 5.5 and 5.6 describe OPAM. Section 5.7 evaluates OPAM. Section 5.8 concludes this chapter.

5.2 Motivating case study

We motivate our work using an industrial case study from the satellite domain. Our case study concerns a mission-critical real-time satellite, named ESAIL [123], which has been developed by LuxSpace – a leading system integrator for microsatellites and aerospace system. ESAIL tracks vessels' movements over the entire globe as the satellite orbits the earth. The vessel-tracking service provided by ESAIL requires real-time processing of messages received from vessels in order to ensure that their voyages are safe with the assistance of accurate, prompt route provisions. Also, as ESAIL orbits the planet, it must be oriented in the proper position on time in order to provide services correctly. Hence, ESAIL's key operations, implemented as real-time tasks, need to be completed within acceptable times, i.e., deadlines.

Engineers at LuxSpace analyze the schedulability of ESAIL across different development stages. At an early design stage, the engineers use a priority assignment method that extends the rate monotonic scheduling policy [76], which is a theoretical priory assignment algorithm used in real-time systems. At a later development stage, if the engineers found that any real-time task of ESAIL cannot complete its execution within its deadline, the engineers, in our study context, reassign priorities to tasks in order to address the problem of deadline violations.

The rate monotonic policy assigns priorities to tasks that arrive to be executed periodically and must be completed within a certain amount of time, i.e., periodic tasks with hard deadlines. According to the policy, periodic tasks that arrive frequently have higher priorities than those of other tasks that arrive rarely. In ESAIL, for example, if the vessel-tracking task arrives every 100ms and the satellite-position control task arrives every 150ms, the former has a higher priority than the latter. However, the rate monotonic policy does not account for tasks that arrive irregularly and should be completed within a reasonable amount of time, i.e., aperiodic tasks with soft deadlines. ESAIL contains aperiodic tasks with soft deadlines as

well, such as a task for updating software. Hence, the engineers extend the rate monotonic policy to assign priorities to all tasks of ESAIL. The extensions are as follows: First, the engineers assign priorities to periodic tasks based on the rate monotonic policy. Second, the engineers assign lower priorities to aperiodic tasks than those of periodic tasks. As aperiodic tasks with soft deadlines are typically considered less critical than periodic tasks with hard deadlines, the engineers aim to ensure that periodic tasks complete their executions within their deadlines by assigning lower priorities to aperiodic tasks while periodic tasks have higher priority. Engineers use a heuristic to assign priorities to aperiodic tasks. They treat aperiodic tasks as (pseudo-)periodic tasks by setting aperiodic tasks' (expected) minimum arrival rates as their fixed arrival periods, making the tasks frequently arrive. The engineers then apply the rate monotonic policy for the aperiodic tasks with the synthetic periods while ensuring that aperiodic tasks have lower priorities than those of periodic tasks.

A priority assignment made at an early design stage keeps changing while developing ESAIL due to various reasons, such as changes in requirements and implementation constraints. At a development stage, instead of relying on the extended rate monotonic policy, the engineers assign priorities based on their domain expertise, manually inspecting schedulability analysis results. Hence, a priority assignment at later development stages often does not follow the extended rate monotonic policy. For example, as aperiodic tasks are also expected to be completed within a reasonable amount of time, some aperiodic tasks may have higher priorities than some periodic tasks as long as they are schedulable.

Engineers at LuxSpace, however, are still faced with the following issues: (1) Their priority assignment method, which extends the rate monotonic scheduling policy, assigns priorities to tasks in order to ensure only that tasks are to be schedulable. However, engineers have a pressing need to understand the quality of priority assignments in detail as they impact ESAIL operations differently. For example, once ESAIL is launched into orbit, the satellite operates in the space environment, which is inherently impossible to be fully tested on the ground. Unexpected space radiations may trigger unusual system interrupts, which hasn't been observed on the ground, resulting in overruns of ESAIL tasks' executions. In such cases, a priority assignment assessed on the ground may not be able to tolerate such unexpected uncertainties. Hence, engineers need a priority assignment that enables ESAIL tasks to tolerate unpredictable uncertainties as much as possible and to be schedulable. (2) Engineers at LuxSpace assign priorities to tasks without any systematic assistance. Instead, they rely on their expertise and the current practices described above to manually assign priorities to ensure that tasks are to be schedulable. To this end, we are collaborating with LuxSpace to develop a solution for addressing these issues in assigning task priority.

5.3 Problem description

This section defines the task, scheduler, and schedulability concepts, which extend the concepts defined in Section 2.1 by augmenting our previous definitions with the notions of safety margins, constraints in assigning priorities, and relationships between real-time tasks. We then describe the problem of optimizing priority assignments such that we maximize the magnitude of safety margins and the degree of constraint satisfaction. Figure 5.1 shows an overview of the conceptual model that represents the key abstractions required to analyze optimal priority assignments for real-time systems. The entities in the conceptual model are described below.

Task. We denote by τ_i a real-time task that should complete its execution within a specified deadline after it is activated (or arrived). Every real-time task τ_i has the following properties: priority denoted

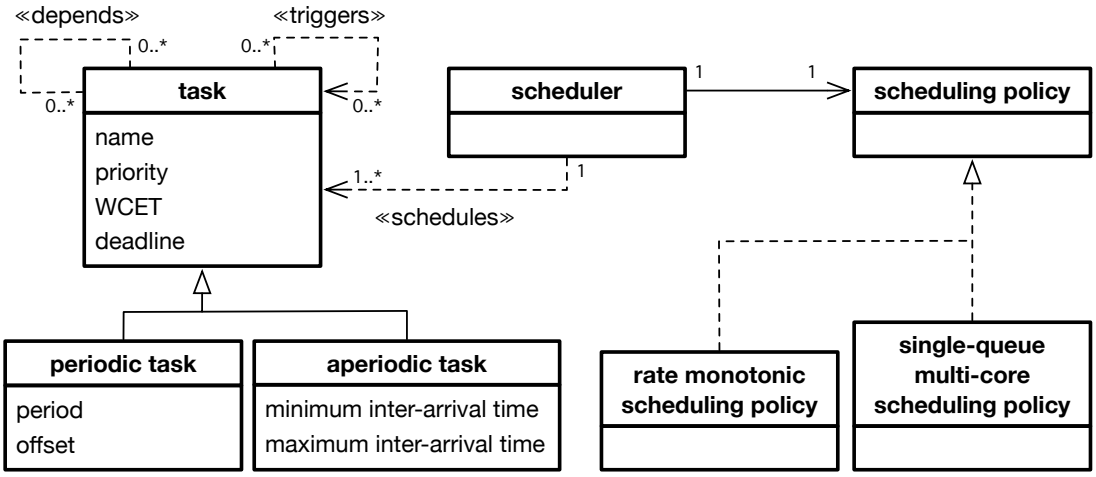


Figure 5.1: A conceptual model representing the key abstractions to analyze optimal priority assignments.

by P_i, deadline denoted by D_i, and worst-case execution time (WCET) denoted by C_i. Task priority P_i determines if an execution of a task τ_i is preempted by another task. Typically, a task τ_i preempts the execution of a task τ_j if the priority of τ_i is higher than the priority of τ_j, i.e., $P_i > P_j$. The P_i priority is a fixed value assigned to task τ_i. Such fixed priorities are determined offline; hence, they are not changed online for any reason. Note that a real-time task scheduler that relies on fixed priorities is applied in all the study subjects in this chapter (see Section 5.7.2) and is commonly used in industrial systems [36, 83, 117, 11, 194, 64, 68, 108].

The D_i function determines the deadline of a task τ_i relative to its arrival time. A task deadline can be either *hard* or *soft*. A hard deadline of a task τ_i constrains that τ_i *must* complete its execution within a deadline D_i after τ_i is activated. While violations of hard deadlines are not acceptable, depending on the operating context of a system, violating soft deadlines may be to some extent tolerated. Note that we use a metaheuristic search relying on fitness functions quantifying the degrees of deadline misses, safety margins, and constraint satisfaction. Such functions do not depend on the nature of the deadlines. Our approach outputs a set of priority assignments that are Pareto optimal with respect to safety margins and constraint satisfaction. Engineers then perform domain-specific trade-off analysis among Pareto solutions. Hence, in this chapter, we handle hard and soft deadline tasks in the same manner.

Real-time tasks are either *periodic* or *aperiodic*. Periodic tasks, which are typically triggered by timed events, are invoked at regular intervals specified by their *period*. We denote by T_i the period of a periodic task τ_i, i.e., a fixed time interval between subsequent activations (or arrivals) of τ_i. Any task that is not periodic is called aperiodic. Aperiodic tasks have irregular arrival times and are activated by external stimuli which occur irregularly. In real-time analysis, based on domain knowledge, we typically specify a minimum inter-arrival time denoted by T_i^{min} and a maximum inter-arrival time denoted by T_i^{max} indicating the minimum and maximum time intervals between two consecutive arrivals of an aperiodic task τ_i. In real-time analysis, sporadic tasks are often separately defined as having irregular arrival intervals and hard deadlines [119]. In our conceptual definitions, however, we do not introduce new notations for sporadic tasks because the deadline and period concepts defined above sufficiently characterize sporadic tasks. Note that for periodic tasks τ_i, we have $T_i^{min} = T_i^{max} = T_i$. Otherwise, for aperiodic tasks τ_i, we have $T_i^{max} > T_i^{min}$.

Task relationships. The execution of a task τ_i depends not only on its own parameters described above, e.g., priority P_i and period T_i, but also on its relationships with other tasks. Relationships between tasks are typically determined by task interactions related to accessing shared resources and triggering arrivals of other tasks [5]. Specifically, if two tasks τ_i and τ_j access a shared resource r in a mutually exclusive way, τ_i may be blocked from executing for the period during which τ_j accesses r. We denote by $dp(\tau_i, \tau_j)$ the resource-dependency relation between tasks τ_i and τ_j that holds if τ_i and τ_j have mutually exclusive access to a shared resource r such that they cannot be executed in parallel or preempt each other, but one can execute only after the other has completed accessing r.

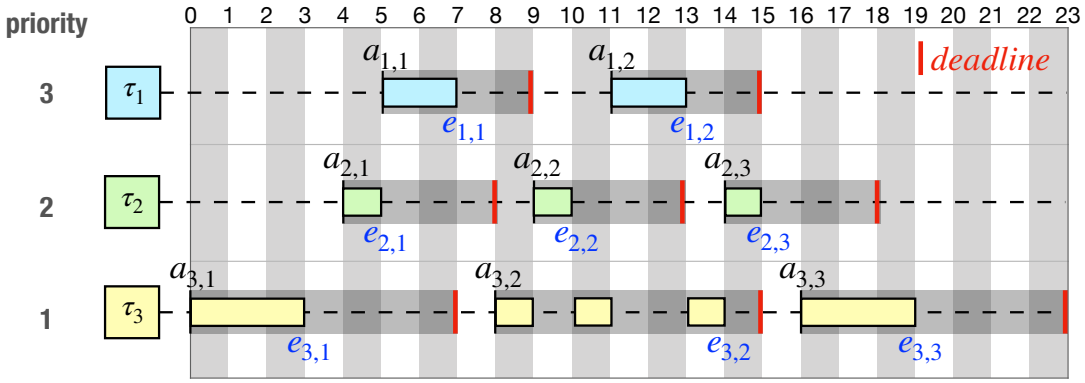
The other type of relationship between tasks is related to a task τ_i triggering the arrival of another task τ_j. This is a common interaction between tasks [120, 11, 64]. For example, τ_i may hand over some of its workload to τ_i due to performance or reliability reasons. We denote by $tr(\tau_i, \tau_j)$ the triggering relation between tasks τ_i and τ_j that holds if τ_i triggers the arrival of τ_j. We note that both relationships are defined at the level of tasks, following prior works [120, 11, 64] describing the five industrial case study systems used in our experiments (see Section 5.7.2).

Scheduler. Let Γ be a set of tasks to be scheduled by a real-time scheduler. A scheduler then dynamically schedules executions of tasks in Γ according to the tasks' arrivals and the scheduler's scheduling policy over the scheduling period $\mathbb{T} = [0, \mathbf{t}]$. We denote by $a_{i,k}$ the kth arrival time of a task $\tau_i \in \Gamma$. The first arrival of a periodic task τ_i does not always occur immediately at the system start time (0). Such offset time from the system start time to the first arrival time $a_{i,1}$ of τ_i is denoted by $\mathit{offset}(j)$. For a periodic task τ_i, the kth arrival of τ_i within $\mathbb{T}$ is $a_{i,k} \leq \mathbf{t}$ and is computed by $a_{i,k} = O_i + (k-1) \cdot T_i$. For an aperiodic task τ_j, $a_{j,k}$ is determined based on the $k-1$th arrival time of τ_j and its minimum and maximum arrival times. Specifically, for $k > 1$, $a_{j,k} \in [a_{j,k-1} + T_j^{min}, a_{j,k-1} + T_j^{max}]$ and, for $k = 1$, $a_{j,1} \in [T_j^{min}, T_j^{max}]$, where $a_{j,k} < \mathbf{t}$.

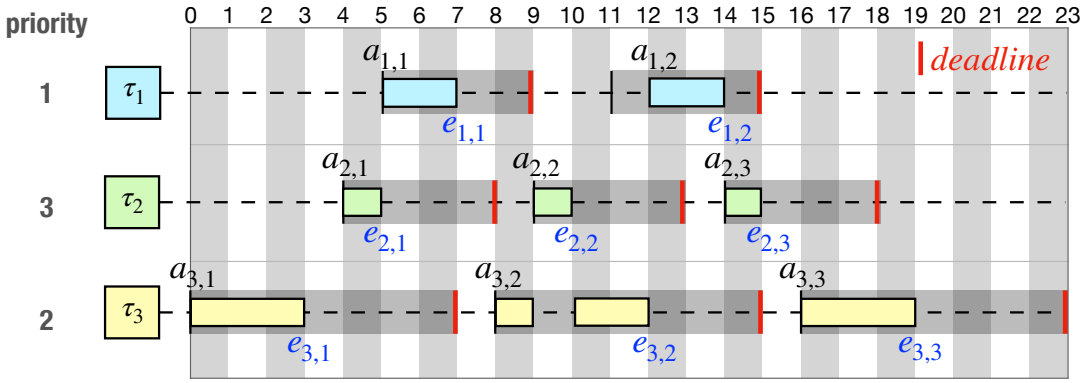
A scheduler reacts to a task arrival at $a_{i,k}$ by scheduling the execution of τ_i. Depending on a scheduling policy (e.g., rate monotonic scheduling policy for single-core systems [76] and single-queue multi-core scheduling policy [17]), an arrived task τ_i may not start its execution at the same time as it arrives when higher priority tasks are executing on all processing cores. Also, task executions may be interrupted due to preemption. We denote by $e_{i,k}$ the completion time for the kth arrival of a task τ_i. According to the worst-case execution time of a task τ_i, we have: $e_{i,k} \geq a_{i,k} + C_i$.

During system operation, a scheduler generates a *schedule scenario* which describes a sequence of task arrivals and their completion time values. We define a schedule scenario as a set S of tuples $(\tau_i, a_{i,k}, e_{i,k})$ indicating that a task τ_i has arrived at $a_{i,k}$ and completed its execution at $e_{i,k}$. Due to a degree of randomness in task execution times and aperiodic task arrivals, a scheduler may generate a different schedule scenario for different runs of a system.

Figure 5.2 shows two schedule scenarios S (Figure 5.2a) and S' (Figure 5.2b) produced by a scheduler over the $[0, 23]$ time period of a system run. Both S and S' describe executions of three tasks, τ_1, τ_2, and τ_3 arrived at the same time stamps (see at_i in the figures). In both scenarios, the aperiodic task τ_1 is characterized by: $T_1^{min} = 5$, $T_1^{max} = 13$, $D_1) = 4$, and $C_1 = 2$. The aperiodic task τ_2 is characterized by: $T_2^{min}) = 3$, $T_2^{max}) = 10$, $D_2 = 4$, and $C_2 = 1$. The periodic task τ_3 is characterised by: $T_3 = 8$, $D_3 = 7$, and $C_3 = 3$. The priorities of the three tasks in S (resp. S') satisfy the following: $P_1 > P_2 > P_3$ (resp. $P_2 > P_3 > P_1$). In both scenarios, task executions can be preempted depending on their priorities.



(a) Schedule scenario S



(b) Schedule scenario S'

Figure 5.2: Example schedule scenarios S and S' of three tasks: τ_1, τ_2, and τ_3. (a) The S schedule scenario is produced when $P_1 = 3$, $P_2 = 2$, and $P_3 = 1$. (b) The S' schedule scenario is produced when $P_1 = 1$, $P_2 = 3$, and $P_3 = 3$.

Then, S is defined by $S = \{(\tau_1, 5, 7), \ldots, (\tau_2, 4, 5), \ldots, (\tau_3, 8, 14), (\tau_3, 16, 19))\}$; and S' is defined by $S' = \{(\tau_1, 5, 7), \ldots, (\tau_2, 4, 5), \ldots, (\tau_3, 8, 12), (\tau_3, 16, 19))\}$.

Schedulability. Given a schedule scenario S, a task τ_i is *schedulable* if τ_i completes its execution before its deadline, i.e., for all $e_{i,k}$ observed in S, $e_{i,k} \leq a_{i,k} + D_i$). Let Γ be a set of tasks to be scheduled by a scheduler. A set Γ of tasks is then schedulable if for every schedule S of Γ, we have no task $\tau_i \in \Gamma$ that misses its deadline.

As shown in schedule scenarios S and S' presented in Figures 5.2a and 5.2b, respectively, all three tasks, τ_1, τ_2, and τ_3, are schedulable. However, we note that the overall amounts of remaining time, i.e., safety margins, from the tasks' completions to their deadlines observed in S and S' are different (see the second completion times and deadlines of τ_1, τ_2, and τ_3 in S and S') because S and S' are produced by using different priority assignments. Engineers typically desire to assign optimal priorities to real-time tasks that aim at maximizing such safety margins, as discussed below.

Problem. In real-time systems, fixed priorities are typically assigned to tasks [61, 108]. Finding an appropriate priority assignment is important not only for ensuring the schedulability of a system but also for maximizing the safety margins within which a system can tolerate unexpected execution time overheads. For example, if an unpredictable error occurs and triggers check-point mechanisms [56], which re-execute part or all of a task τ_i, then the execution time of τ_i unexpectedly overruns. Hence, engineers

need an optimal priority assignment that maximizes the overall remaining times from task completion times to task deadlines, i.e., safety margins.

While assigning priorities to tasks, engineers also account for constraints, that are often but not always domain-specific. For example, aperiodic tasks' priorities should be lower than those of periodic tasks because periodic tasks are often more critical than aperiodic tasks. Hence, engineers develop a system that prioritizes executions of periodic tasks over aperiodic tasks. Recall from Section 5.2, this constraint is desirable by engineers. When needed, however, engineers can violate the constraint to some extent in order to ensure that aperiodic tasks complete within a reasonable amount of time while periodic tasks meet their deadlines. Constraints can be either *hard* constraints, which must be satisfied, or *soft* constraints, which are desired to be satisfied. In our study, hard constraints need to be assured while scheduling tasks, e.g., a running task's priority must be higher than a ready task's priority, which are enforced by a scheduler. In the context of optimizing priority assignments, we focus on maximizing the extent of satisfying soft constraints. We refer to a soft constraint as a constraint in this chapter.

Our work aims at optimizing priority assignments that maximize the safety margins while satisfying such constraints. Specifically, for a set Γ of tasks to be analyzed, we define three concepts as follows: (1) a priority assignment for Γ denoted by $\vec{P}$, (2) the magnitude of safety margins for a priority assignment $\vec{P}$ denoted by $fs(\vec{P})$, and (3) the degree of constraint satisfaction denoted by $fc(\vec{P})$. We note that Section 5.6.3 describes how we optimize $\vec{P}$, and compute $fs(\vec{P})$ and $fc(\vec{P})$ in detail. Our study aims at finding a set **B** of best possible priory assignments that are Pareto optimal [102] such that a priority assignment $\vec{P} \in \mathbf{B}$ maximizes both $fs(\vec{P})$ and $fc(\vec{P})$, and any other priority assignments in **B** are equally viable.

5.4 Related work

This section discusses related research strands in the areas of priority assignments, real-time analysis using exhaustive techniques, search-based analysis in real-time systems, and coevolutionary analysis in software engineering.

Priority assignment. The problem of optimally assigning priorities to real-time tasks has been widely studied [76, 118, 115, 19, 170, 78, 20, 56, 46, 58, 59, 55, 61, 197, 92]. Fineberg and Serlin [76] reported early work that relies on a simple system model, assuming, for example, that all tasks arrive periodically, tasks run on a single processing core, tasks' deadlines are equal to their periods, and task executions are independent from one another. They proposed a priority assignment method, named rate-monotonic priority ordering (RMPO), that assigns higher priorities to the tasks with shorter periods. RMPO can find a feasible priority assignment that guarantees periodic tasks to be schedulable when such priority assignments exist [118]. Leung and Whitehead [115] extended RMPO to relax one of the underlying assumptions made in RMPO. Specifically, their priority assignment approach, known as deadline-monotonic priority ordering (DMPO), accounts for task deadlines that can be less than or equal to their periods. In contrast to our work, however, these methods are often not applicable to industrial systems that are not compatible with their simplified system models. Recall from Section 5.3 that a realistic system typically consists of both periodic and aperiodic tasks. Task executions depend on their relationships, i.e., resource dependencies and triggering relationships, with other tasks.

Audsley [20] designed a priority assignment method, named optimal priority assignment (OPA), that relies on an existing schedulability analysis method M. OPA guarantees to find a feasible priority

Table 5.1: Comparing our work, OPAM, with existing priority assignment techniques with respect to the properties captured in their underlying system models.

Properties	OPAM	RMPO	DMPO	OPA	OPA-MLD	RPA	FNR-PA	PRPA	OPTA	EPAF
Periodic task	○	○	○	○	○	○	○	○	○	○
Aperiodic task	○			○	○	○	○	○		
Resource dependency	○									
Triggering relationship	○									
Multi-core system	○			○	○	○	○	○		
Safety margin	○					○	○	○		
Engineering constraint	○				○					

assignment that is schedulable according to M if such priority assignments exist. OPA is applicable to more complex systems than those supported by the methods mentioned above, i.e., RMPO and DMPO. Specifically, OPA can find a feasible priority assignment even in the following situations: (1) First arrivals of periodic tasks occur after some offset time [19]. (2) Aperiodic tasks have arbitrary deadlines [170]. (3) Task executions are scheduled based on a non-preemptive scheduling policy [78]. (4) Tasks run on multiple processing cores [59]. Unlike our approach that accounts for two objectives, safety margins and engineering constraints (see Section 5.3), OPA attempts to find a feasible priority assignment whose only objective is to make all tasks schedulable. Note that such a feasible priority assignment does not necessarily maximize safety margins as discussed in Section 5.3. Hence, a feasible priority assignment obtained by OPA is often fragile and sensitive any changes in task executions and unable to accommodate unexpected overheads in task execution times, which are commonly observed in industrial systems [56].

OPA has been extended by several works [56, 46, 58, 55]. Davis and Burns [56] presented a robust priority assignment method (RPA) with a degree of tolerance for unexpected overruns of task execution times. Chu and Burns [46] introduced an extended OPA algorithm (OPA-MLD) that minimizes the lexicographical distance between the desired priority assignment and the one obtained by the algorithm. OPA-MLD enables important tasks to have higher priorities. Davis and Bertogna [55] proposed an RPA extension (FNR-PA) to make RPA work when a system allows task preemption to be deferred for some interval of time. Davis and Burns [58] developed a probabilistic robust priority assignment method (PRPA) for a real-time system to be less likely to violate its deadlines. Even though the prior works mentioned above improve OPA to some extent, they assume that task executions are independent of one another. In contrast to these existing approaches, OPAM accounts for dependencies among task executions, i.e., resource dependencies and triggering relationships (see our problem description in Section 5.3).

Some recent priority assignment techniques address scalability. Hatvani et al. [92] presented an optimal priority and preemption-threshold assignment algorithm (OPTA) that attempts to decrease the

computation time for finding a feasible priority assignment. OPTA uses a heuristic to traverse a problem space while pruning infeasible paths to efficiently and effectively explore the problem space. Zhao and Zeng [197] introduced an effective priority assignment framework (EPAF) that combines a commercial solver for integer linear programs and their problem-specific optimization algorithm. However, these methods rely on simple system models that assume, for example, task executions to be independent and running on a single processing core. Therefore, the applicability of these techniques is limited. In contrast, recall from Sections 5.2 and 5.3 that our approach aims at scaling to complex industrial systems while accounting for realistic system characteristics regarding task periods, inter-arrival times, resource dependencies, triggering relationships, and multiple processing cores.

Table 5.1 compares our work, OPAM, with the other priority assignment techniques mentioned above. As shown in the table, we note that prior works rely on system models that are very restrictive. In particular, existing work assumes that task executions are independent of one another. However, task dependencies such as resource dependencies and triggering relationships are commonly observed in industrial systems. In addition, we note that no existing solution simultaneously accounts for safety margins and engineering constraints. Hence, to our knowledge, OPAM is the first attempt to provide engineers with a set of equally viable priority assignments, allowing trade-off analysis with respect to the two objectives: maximizing safety margins and satisfying engineering constraints.

Real-time analysis using exhaustive techniques. Constraint programming and model checking have been applied to conclusively and exhaustively verify whether or not a system meets its deadlines [106, 5, 138, 6]. Existing research on priority assignment based on OPA rely on such exhaustive techniques to prove the schedulability of a set of tasks for a given priority assignment. We note that schedulability analysis is, in general, an NP-hard problem [61] that cannot be solved in polynomial time. As a result, exhaustive techniques based on model checking and constraint solving are often not amenable to analyze large industrial systems such as ESAIL – our motivating case study system – described in Section 5.2. To assess if exhaustive techniques could scale to ESAIL, as discussed in Section 5.7.8, we performed a preliminary experiment using UPPAAL [25], a model checker for real-time systems. We observed that UPPAAL was not able to verify schedulability of ESAIL tasks for a fixed priority assignment even after letting it run for several days (see Section 5.7.8 for more details).

Search-based analysis in real-time systems. In real-time systems, most of the existing works that use search-based techniques focus on testing [180, 179, 36, 117, 14]. Wegener et al. [180, 179] introduced a testing approach based on a genetic algorithm that aims to check computation time, memory usage, and task synchronization by analyzing the control flow of a program. Briand et al. [36] applied a genetic algorithm to find stress test scenarios for real-time systems. Lin et al. [117] proposed a search-based approach to check whether a real-time system meets its timing and security constraints. Arcuri et al. [14] presented a black-box system testing approach based on a genetic algorithm. Beyond testing real-time systems, Nejati et al. [137, 139] developed a search-based trade-off analysis technique that helps engineers balance the satisfaction of temporal constraints and keeping the CPU time usage at an acceptable level. Lee et al. [113] combined a search algorithm and machine learning to estimate safe ranges of worst-case task execution times within which tasks likely meet their deadlines. In contrast to these prior works, OPAM addresses the problem of optimally assigning priorities to real-time tasks while accounting for multiple objectives regarding safety margins and engineering constraints, thus enabling Pareto (trade-off)

analysis. Further, OPAM uses a multi-objective, competitive coevolutionary search algorithm, which has been rarely applied to date in prior studies of real-time systems, as discussed next.

Coevolutionary analysis in software engineering. Despite the success of search-based software engineering (SBSE) in many application domains including software testing [180, 179, 117, 14, 158], program repair [181, 168, 1], and self-adaptation [9, 44, 157], coevolutionary algorithms have been applied in only a few prior studies [185, 186, 34]. Wilkerson et al. [185, 186] present a coevolution-based approach to automatically correct software. Their work introduced a program representation language to facilitate their automated corrections. Boussaa et al. [34] developed a code-smells detection approach. The main idea is to evolve two competing populations of code-smell detection rules and artificial code-smells. Unlike these prior works, we study the problem of optimally assigning priorities to tasks in real-time systems. To our knowledge, we are the first to address the priority assignment problem using a multi-objective, competitive coevolutionary search algorithm.

5.5 Approach overview

Finding an optimal priority assignment is an inherently interactive process. In practice, once engineers assign priorities to the real-time tasks in a system, testers then stress the system to find a condition, i.e., a particular sequence of task arrivals, in which a task execution violates its deadline. Testers typically use a simulator or hardware equipment to stress the system by triggering plausible worst-case arrivals of tasks that maximize the likelihood of deadline misses. If testers find task arrivals that induce deadline misses, the task arrivals are reported to engineers in order to fix the problem by reassigning priorities. This interactive process of assigning priorities and testing schedulability continues until both engineers and testers ensure that the tasks meet their deadlines.

For such intrinsically interactive problem-solving domains, we conjecture that coevolutionary algorithms are potentially suitable solutions. A coevolutionary algorithm is a search algorithm that mutually adapts one of different species, e.g., in our study, two populations of priority assignments and task-arrival sequences, acting as foils against one another. Specifically, we apply multi-objective, two-population competitive coevolution [122] to address our problem of finding optimal priority assignments (see Section 5.3). In our approach, the two populations of priority assignments and stress test scenarios, i.e., task-arrival sequences, evolve synchronously, competing with each other in order to search for optimal priority assignments that maximize the magnitude of safety margins from deadlines and the extent of constraint satisfaction. Note that better priority assignments enable a system to achieve larger safety margins. Hence, those priority assignments have a higher chance to pass stress test scenarios. This impacts the stress test scenarios because they need to evolve as well, aiming at inducing deadline misses in the system.

Recall from Section 5.4 that most of the existing SBSE research relies on search algorithms using a single population [44, 1, 157]. However, such algorithms do not fit the problem of priority assignments targeted here. When (1) two competing traits between task arrivals and priority assignments are encoded together in an individual of a single population and (2) two contradicting fitness functions regarding safety margins and deadline misses, which are exact opposites, assess such individuals, the notion of Pareto optimality is not applicable. In that case, maximizing the magnitude of safety margins necessarily entails minimizing the magnitude of deadline misses. Hence, a single population-based search algorithm cannot make Pareto improvements that maximize safety margins (resp. deadline misses) while not minimizing

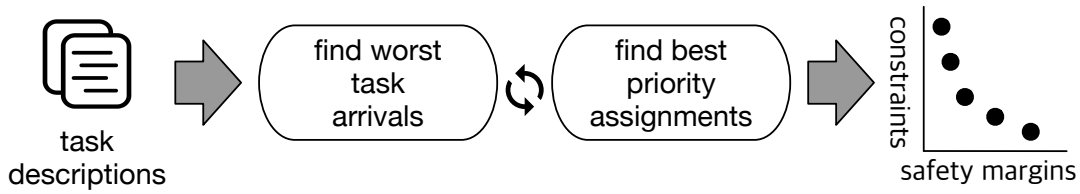


Figure 5.3: An overview of our Optimal Priority Assignment Method for real-time systems (OPAM).

deadline misses (resp. safety margins). Specifically, the dominance relation over such individuals does not exist because if an individual I is strictly better than another individual I' in one fitness value, I is always worse than I' in the other fitness value. Hence, we are not able to obtain equally viable solutions with respect to the contradicting objectives using such a method.

Figure 5.3 shows an overview of our proposed solution: Optimal Priority Assignment Method for real-time tasks (OPAM). OPAM requires as input task descriptions defined by engineers, which specify task characteristics and their relationships (see Section 5.3). Given such input task descriptions, the "find worst task arrivals' and "find best priority assignments" steps aim at generating worst-case sequences of task arrivals and best-case priority assignments, respectively. A worst-case sequence of task arrivals means that the magnitude of deadline misses, i.e., the amounts of time from task deadlines to task completion times, is maximized when tasks arrive as defined in the sequence. Note that if there is no deadline miss, a task-arrival sequence is considered worst-case if tasks complete their executions as close to their deadlines as possible. In contrast, a priority assignment is best-case when the magnitude of safety margins is maximized. Beyond maximizing safety margins, the "find best priority assignments" step accounts for satisfying engineering constraints in assigning priorities to tasks. OPAM evolves two competing populations of task-arrival sequences and priority assignments synchronously generated from the two steps. OPAM then outputs a set of priority assignments that are Pareto optimal with regards to the magnitude of safety margins and the extent of satisfying constraints. Hence, OPAM allows engineers to perform domain-specific trade-off analysis among Pareto solutions and is useful in practice to support decision making with respect to their task design. For example, suppose engineers develop a weakly hard real-time systems [27] that can tolerate occasional deadline misses. In that case, engineers may consider a few deadline misses as less important (as long as their consequences are negligible) than the overall magnitude of safety margins in their trade-off analysis. Section 5.6 describes OPAM in detail.

5.6 Competitive coevolution

Figure 5.4 describes the OPAM algorithm for finding optimal priority assignments, which employs multi-objective, two-population competitive coevolution. The algorithm first randomly initializes two populations **A** and **P** for task-arrival sequences and priority assignments, respectively (lines 13–15). For **A**, OPAM randomly varies task arrivals of aperiodic tasks to create ps_a task-arrival sequences, according to the input task descriptions Γ. Regarding **P**, OPAM randomly creates ps_p priority assignments that may include one defined by engineers if available.

The two populations sequentially evolve during the allotted analysis budget (see line 17 in Figure 5.4). The best priority assignment is the one that makes tasks schedulable and maximizes the magnitude of safety margins, while satisfying engineering constraints for a given worst sequence of task arrivals. Hence, searching for the best priority assignments involves searching for the worst sequences of task

```
1  Algorithm Search optimal priority assignments
2  Input Γ: task descriptions
3  Input n_c: number of coevolution cycles //budget
4  Input ps_a: population size //task-arrival sequences
5  Input ps_p: population size //priority assignments
6  Input cp_a: crossover probability //task-arrival sequences
7  Input cp_p: crossover probability //priority assignments
8  Input mp_a: mutation probability //task-arrival sequences
9  Input mp_p: mutation probability //priority assignments
10 Input E: set of task-arrival sequences //external evaluation
11 Output B: best Pareto front
12
13 //initialize populations
14 A ← randomize_arrivals(Γ, ps_a)
15 P ← randomize_priorities(Γ, ps_p)
16
17 for n_c times do
18   //evolution: find worst-case sequences of task arrivals
19   //objective: deadline misses
20   evaluate_internal_fitness_arrivals(A, P)
21   A ← breed_arrivals(A, P, cp_a, mp_a) //GA
22
23   //evolution: find best-case priority assignments
24   //objectives: safety margins and constraints
25   evaluate_internal_fitness_priorities(P, A)
26   P ← breed_priorities(P, A, cp_p, mp_p) //NSGAII
27
28   //external fitness evaluation
29   //objectives: safety margins and constraints
30   evaluate_external_fitness(P, E)
31   B ← select_best(P ∪ B)
32
33 return B
```

Figure 5.4: Multi-objective two-population competitive coevolution for finding optimal priority assignments.

arrivals. We create two populations **A** and **P** searching for the worst arrival sequences and the best priority assignments, respectively. The fitness values of task-arrival sequences in **A** are computed based on how well they challenge the priority assignments in **P**, i.e., maximizing the magnitude of deadline misses (line 20). Likewise, the priority assignments in **P** are evaluated based on how well they perform against the task-arrival sequences in **A**, i.e., maximizing the magnitude of safety margins while satisfying constraints (line 25). Once the two populations are assessed against each other, OPAM generates the next populations based on the computed fitness values (lines 21 and 26). OPAM tailors the breading mechanisms of steady-state genetic algorithms (GA) [184] for **A** and NSGAII [63] for **P**.

OPAM uses two types of fitness functions, namely internal and external fitness evaluations, which play a different and complementary role as described below. The two internal fitness evaluations in lines 20 and 25 of the listing in Figure 5.4 aim at selecting individuals – task-arrival sequences and priority assignments – for breeding the next **A** and **P** populations. OPAM evaluates the external fitness for the **P** population of priority assignments to find a best Pareto front (lines 28–31). As shown in lines 20 and 25, the internal fitness values of individuals in **A** (resp. **P**) are computed based on how they perform with respect to individuals in **P** (resp. **A**). Hence, an individual's internal fitness is assessed through interactions with competing individuals. For example, a priority assignment in the first generation may

have acceptable fitness values regarding safety margins and constraint satisfaction with respect to the first generation of task-arrival sequences, which are likely far from worst-case sequences. However, priority assignment fitness may get worse in later generations as the task-arrival sequences evolve towards larger deadline misses. Thus, if OPAM simply monitors internal fitness, it cannot reliably detect coevolutionary progress as an individual's internal fitness changes according to competing individuals. The problem of monitoring progress in coevolution has been observed in many studies [75, 146]. To address it, OPAM computes external fitness values of priority assignments in **P** based on a set **E** of task-arrival sequences generated independently from the coevolution process. By doing so, OPAM can observe the monotonic improvement of external fitness for priority assignments. We note that, in general, if interactions between two competing populations are finite and any interaction can be examined with non-zero probability at any time, monotonicity guarantees that a coevolutionary algorithm converges to a solution [146].

We note that our approach for evolving task-arrival sequences is based on past work [36], where a specific genetic algorithm configuration was proposed to find worst-case task-arrival sequences. One significant modification is that OPAM accounts for task relationships – resource-dependency and task triggering relationships – and a multi-core scheduling policy based on simulations to evaluate the magnitude of deadline misses.

Following standard practice [149], the next sections describe OPAM in detail by defining the representations, the scheduler, the fitness functions, and the evolutionary algorithms for coevolving the task-arrival sequences and priority assignments. We then describe the external fitness evaluation of OPAM.

5.6.1 Representations

OPAM coevolves two populations of task-arrival sequences and priority assignments. A task-arrival sequence is defined by their inter-arrival time characteristics (see Section 5.3). A priority assignment is defined by a function that maps priorities to tasks.

Task-arrival sequences. Given a set Γ of tasks to be scheduled, a feasible sequence of task arrivals is a set A of tuples $(\tau_i, a_{i,k})$ where $\tau_i \in \Gamma$ and $a_{i,k}$ is the kth arrival time of a task τ_i. Thus, a solution A represents a valid sequence of task arrivals of Γ (see valid $a_{i,k}$ computation in Section 5.3). Let $\mathbb{T} = [0, \mathbf{t}]$ be the time period during which a scheduler receives task arrivals. The size of A is equal to the number of task arrivals over the $\mathbb{T}$ time period. Due to the varying inter-arrival times of aperiodic tasks (Section 5.3), the size of A will vary across different sequences.

Priority assignments. Given a set Γ of tasks to be scheduled, a feasible priority assignment is a list $\vec{P}$ of priority P_i for each task $\tau_i \in \Gamma$. OPAM assigns a non-negative integer to a priority P_i of τ_i such that priorities are comparable to one another. The size of $\vec{P}$ is equal to the number of tasks in Γ. Each task in Γ has a unique priority. Hence, a priority assignment $\vec{P}$ is a permutation of all tasks' priorities. We note that these characteristics of priority assignments are common in many real-time analysis methods [20, 56, 197] and industrial systems (e.g., see our six industrial case study systems described in Section 5.7.2).

5.6.2 Simulation

OPAM relies on simulation for analyzing the schedulability of tasks in a scalable way. For instance, an inter-arrival time of a software update task in a satellite system is approximately at most three months. In such cases, conducting an analysis based on an actual scheduler is prohibitively expensive. Also, applying an exhaustive technique for schedulability analysis typically doesn't scale to an industrial system (e.g., see

our experiment results using a model checker described in Section 5.7.8). Instead, OPAM uses a real-time task scheduling simulator, named OPAMScheduler, which applies a scheduling policy, i.e., single-queue multi-core scheduling policy [17], based on discrete simulation time events. Note that we chose the single-queue multi-core scheduling policy for OPAMScheduler since our case study systems (described in Section 5.7.2) rely on this policy.

OPAMScheduler takes as input a feasible task-arrival sequence A and a priority assignment $\vec{P}$ for scheduling a set Γ of tasks. It then outputs a schedule scenario as a set S of tuples $(\tau_i, a_{i,k}, e_{i,k})$ where $a_{i,k}$ and $e_{i,k}$ are the kth arrival and end time values of a task τ_i, respectively (see Section 5.3). For each task τ_i, OPAMScheduler computes $e_{i,k}$ based on its WCET and scheduling policy while accounting for task relationships (see the $dp(\tau_i, \tau_j)$ resource-dependency relationship and the $tr(\tau_i, \tau_j)$ task triggering relationship in Section 5.3). To simulate the worst-case executions of tasks, OPAMScheduler assigns tasks' WCETs to their execution times.

OPAMScheduler implements a single-queue multi-core scheduling policy [17], which schedules a task τ_i with explicit priority P_i and deadline D_i. When tasks arrive, OPAMScheduler puts them into a single queue that contains tasks to be scheduled. At any simulation time, if there are tasks in the queue and multiple cores are available to execute tasks, OPAMScheduler first fetches a task τ_i from the queue in which τ_i has the highest priority P_i. OPAMScheduler then allocates task τ_i to any available core. Note that if task τ_i shares a resource with a running task τ_j in another core, i.e., the $dp(\tau_i, \tau_j)$ resource-dependency relationship holds, τ_i will be blocked until τ_j releases the shared resource.

OPAMScheduler works under the assumption that context switching time is negligible, which is also a working assumption in many scheduling analysis methods [118, 20, 64]. Note that the assumption is practically valid and useful at an early development step in the context of real-time analysis. For instance, our collaborating partner, LuxSpace, accounts for the waiting time of tasks due to context switching between tasks through adding some extra time to WCET estimates at the task design stage. Note that OPAM can be applied with any scheduling policy, including those that account for context switching time and multiple queues.

5.6.3 Fitness functions

Internal fitness: deadline misses. Given a feasible task-arrival sequence A and a priority assignment $\vec{P}$, we formulate a function, $fd(A, \vec{P})$, to quantify the degree of deadline misses regarding a set J of tasks to be scheduled. To compute $fd(A, \vec{P})$, OPAM runs OPAMScheduler for A and $\vec{P}$ and obtains a schedule scenario S. We denote by $dist(\tau_i, k)$ the distance between the end time and the deadline of the kth arrival of task τ_i observed in S and define $dist(\tau_i, k) = e_{i,k} - a_{i,k} + D_i$ (see Section 5.3 for the notation end time $e_{i,k}$, arrival time $a_{i,k}$, and deadline D_i). We denote by $lk(\tau_i)$ the last arrival index of a task τ_i in A. Given a set Γ of tasks to be scheduled, the $fd(A, \vec{P})$ function is defined as follows:

$$fd(A, \vec{P}) = \sum_{\tau_i \in \Gamma, k \in [1, lk(\tau_i)]} 2^{dist(\tau_i, k)}$$

Note that $fd(A, \vec{P})$ is defined as an exponential equation. Hence, when all task executions observed in a schedule scenario S meet their deadlines, $fd(A, \vec{P})$ is a small value as any distance $dist(\tau_i, k)$ between the task end time and the deadline of the kth arrival of task τ_i is a negative value. In contrast, deadline misses result in positive values for $dist(\tau_i, k)$. In such cases, $fd(A, \vec{P})$ is a large value. The exponential

form of $fd(A, \vec{P})$ was precisely selected for this reason, to assign large values for deadline misses but small values when deadlines are met. By doing so, $fd(A, \vec{P})$ prevents an undesirable solution that would result into many task executions meeting deadlines obfuscating a smaller number of deadline misses.

Following the principles of competitive coevolution, individuals in a population **A** of task-arrival sequences need to be assessed by pitting them against individuals in the other population **P** of priority assignments. We denote by $fd(A, \mathbf{P})$ the internal fitness function that quantifies the overall magnitude of deadline misses across all priority assignment $\vec{P} \in \mathbf{P}$, regarding a set Γ of tasks to be scheduled. The $fd(A, \mathbf{P})$ fitness is used for breeding the next population of task-arrival sequences. OPAM aims to maximize $fd(A, \mathbf{P})$, defined as follows:

$$fd(A, \mathbf{P}) = \sum_{\vec{P} \in \mathbf{P}} fd(A, \vec{P}) / |\mathbf{P}|$$

Internal fitness: safety margins. Given a feasible priority assignment $\vec{P}$ and a task-arrival sequence A, we denote by $fs(\vec{P}, A)$ the magnitude of safety margins regarding a set J of tasks to be scheduled. The computation of $fs(\vec{P}, A)$ is similar to the computation of $fd(A, \vec{P})$ regarding the use of OPAMScheduler, which outputs a schedule scenario S. The difference is that OPAM reverses the sign of $fd(A, \vec{P})$ as OPAM aims at maximizing the magnitude of safety margins. Given a set J of tasks to be scheduled, the $fs(\vec{P}, A)$ function is defined as follows:

$$fs(\vec{P}, A) = \sum_{\tau_i \in \Gamma, k \in [1, lk(\tau_i)]} -2^{dist(\tau_i, k)} \quad \text{(i.e,} -fd(A, \vec{P}))$$

Given two populations **P** and **A** of priority assignments and task-arrival sequences, similar to internal fitness $fd(A, \mathbf{P})$, priority assignments in **P** need to be assessed against task-arrival sequences in **A**. We formulate an internal fitness function, $fs(\vec{P}, \mathbf{A})$, to quantify the overall magnitude of safety margins across all task-arrival sequences $A \in \mathbf{A}$, regarding a set Γ of tasks to be scheduled and a priority assignment $\vec{P}$. OPAM relies on the $fs(\vec{P}, \mathbf{A})$ function to breed the next population of priority assignments. OPAM aims to maximize $fs(\vec{P}, \mathbf{A})$, which is defined as follows:

$$fs(\vec{P}, \mathbf{A}) = \sum_{A \in \mathbf{A}} fs(\vec{P}, A) / |\mathbf{A}|$$

Internal fitness: constraints. Given a priority assignment $\vec{P}$, we formulate an internal fitness function, $fc(\vec{P})$, to quantify the degree of satisfaction of soft constraints set by engineers. Such function is required as we recast the satisfaction of such constraints into an optimization problem, in order to minimize constraint violations. Specifically, OPAM accounts for the following constraint: aperiodic tasks should have lower priorities than those of periodic tasks. Recall from Section 5.2 that engineers consider this constraint to be desirable. We denote by $lp(\vec{P})$ the lowest priority of periodic tasks in $\vec{P}$. For a set J of tasks to be scheduled, OPAM aims to maximize $fc(\vec{P})$, which is defined as follows:

$$fc(\vec{P}) = \sum_{j \in J} \begin{cases} lp(\vec{P}) - P_i, \text{ if } \tau_i \text{ is an aperiodic task} \\ 0, \text{ otherwise} \end{cases}$$

Greater P_i values denote higher priorities. Given a priority assignment $\vec{P}$, if P_i for an aperiodic task τ_i is lower than the priority of any of the periodic tasks, $lp(\vec{P}) - P_i$ is a positive value. OPAM measures the difference between priorities of aperiodic and periodic tasks. By doing so, $fc(\vec{P})$ rewards aperiodic

tasks that satisfy the above constraint and consistently penalizes those that violate it. Hence, OPAM aims at maximizing $fc(\vec{P})$.

External fitness: safety margins and constraints. To examine the quality of priority assignments and monitor the progress of coevolution, OPAM takes as input a set **E** of task-arrival sequences created independently from the coevolution process. Given a set **E** of task-arrival sequences and a priority assignment $\vec{P}$, OPAM utilizes $fs(\vec{P}, \mathbf{E})$ and $fc(\vec{P})$ described above as external fitness functions for quantifying the magnitude of safety margins and the extent of constraint satisfaction, respectively. As **E** does not change over the coevolution process, $fs(\vec{P}, \mathbf{E})$ is used for evaluating a priority assignment $\vec{P}$ since it is not impacted by the evolution of task-arrival sequences. Hence, external fitness functions ensure that OPAM monitors the progress of coevolution in a stable manner. Given two populations **P** and **A** of priority assignments and task-arrival sequences, we recall that the $fd(A, \mathbf{P})$ internal fitness function quantifies the overall magnitude of deadline misses across all priority assignments in **P** for the given sequence of task arrivals A. The $fs(\vec{P}, \mathbf{A})$ internal fitness function quantifies the overall magnitude of safety margins across all sequences of task arrivals in **A** for the given priority assignments $\vec{P}$. Hence, the internal fitness of A (resp. $\vec{P}$) is assessed through interactions with competing individuals in **P** (resp. **A**). Therefore, if OPAM relies only on the internal fitness functions, it cannot gauge the progress of coevolution in a stable manner as an individual's internal fitness depends on competing individuals.

We note that soft deadline tasks also require to execute within reasonable execution time, i.e., (soft) deadline. As the above fitness functions return quantified degrees of deadline misses and safety margins, OPAM uses the same fitness functions for both soft and hard deadline tasks.

5.6.4 Evolution: worst-case task arrivals

The algorithm in Figure 5.5 describes in detail the evolution of task-arrival sequences in lines 18–21 of the listing in Figure 5.4. OPAM adapts a steady-state Genetic Algorithm (GA) [122] for evolving task-arrival sequences. As shown in lines 8–14, OPAM first evaluates each task-arrival sequence in the **A** population against the **P** population of priority assignments. OPAM executes OPAMScheduler to obtain a schedule scenario S for a task-arrival sequence $A_x \in \mathbf{A}$ and a priority assignment $\vec{P}_y \in \mathbf{P}$ (line 11). OPAM then computes the internal fitness $fd(A_x, \mathbf{P})$ capturing the magnitude of deadline misses (lines 12–14). We note that a steady-state GA iteratively breeds offspring, assess their fitness, and then reintroduce them into a population. However, OPAM computes internal fitness of all task-arrival sequences in **A** at every generation. This is because internal fitness is computed in relation to **P**, which is coevolving with **A**.

Breeding the next population is done by using the following genetic operators: (1) *Selection:* OPAM selects candidate task-arrival sequences using a tournament selection technique, with the tournament size equal to two which is the most common setting [77] (line 17 in Figure 5.5). (2) *Crossover:* Selected candidate task-arrival sequences serve as parents to create offspring using a crossover operation (line 18). (3) *Mutation:* The offspring are then mutated (line 19). Below, we describe our crossover and mutation operators.

Crossover. A crossover operator is used to produce offspring by mixing traits of parent solutions. OPAM modifies the standard one-point crossover operator [122] as two parent task-arrival sequences A_p and A_q may have different sizes, i.e., $|A_p| \neq |A_q|$. Let $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of tasks to be scheduled. Our crossover operator first randomly selects an aperiodic task $\tau_r \in \Gamma$. For all $i \in [1, r]$ and

```
1  Algorithm Task-arrival sequences evolution
2  Input A: population of task-arrival sequences
3  Input P: population of priority assignments
4  Input cp_a: crossover probability //task-arrival sequences
5  Input mp_a: mutation probability //task-arrival sequences
6  Output A: population of task-arrival sequences
7
8  //evaluate internal fitness values for A
9  for each A_x ∈ A
10   for each P⃗_y ∈ P
11     S ← simulate(A_x, P⃗_y) //OPAMScheduler
12     //dist(τ_i, k) is computed based on S
13     fd(A_x, P⃗_y) = Σ_{τ_i∈Γ, k∈[1, lk(τ_i)]} 2^{dist(τ_i,k)}
14   fd(A_x, P) = Σ_{P⃗_y∈P} fd(A_x, P⃗_y)/|P|
15
16 //breed task-arrival sequences
17 parents ← select_arrivals(A)
18 offspring ← crossover_arrivals(parents, cp_a)
19 offspring ← mutate_arrivals(offspring, mp_a)
20 //evaluate internal fitness values for offspring
21 for each A_x ∈ offspring
22   for each P⃗_y ∈ P
23     S ← simulate(A_x, P⃗_y) //OPAMScheduler
24     //dist(τ_i, k) is computed based on S
25     fd(A_x, P⃗_y) = Σ_{τ_i∈Γ, k∈[1, lk(τ_i)]} 2^{dist(τ_i,k)}
26   fd(A_x, P) = Σ_{P⃗_y∈P} fd(A_x, P⃗_y)/|P|
27 A ← replace_arrivals(A, offspring)
28
29 return A
```

Figure 5.5: A steady-state GA-based algorithm for evolving task-arrival sequences.

$\tau_i \in \Gamma$, OPAM then swaps all τ_i arrivals between the two task-arrival sequences A_p and A_q. Since Γ is fixed for all solutions, OPAM can cross over two solutions that may have different sizes.

Mutation operator OPAM uses a heuristic mutation algorithm. For a task-arrival sequence A, OPAM mutates the kth task arrival time $a_{i,k}$ of an aperiodic task τ_i with a mutation probability. OPAM chooses a new arrival time value of $a_{i,k}$ based on the $[P_i^{min}, P_i^{max}]$ inter-arrival time range of τ_i. If such a mutation of the kth arrival time of τ_i does not affect the validity of the k+1th arrival time of τ_i, the mutation operation ends. Specifically, let d be a mutated value of $a_{i,k}$. In case $a_{i,k+1} \in [d + P_i^{min}, d + P_i^{max}]$, OPAM returns the mutated A task-arrival sequence.

After mutating the kth arrival time $a_{i,k}$ of a task τ_i in a solution A, if the k+1th arrival becomes invalid, OPAM corrects the remaining arrivals of τ_i. Let o and d be, respectively, the original and mutated kth arrival time of τ_i. For all the arrivals of τ_i after d, OPAM first updates their original arrival time values by adding the difference $d - o$. Let $\mathbb{T} = [0, \mathbf{t}]$ be the scheduling period. OPAM then removes some arrivals of τ_i if they are mutated to arrive after $\mathbf{t}$ or adds new arrivals of τ_i while ensuring that all tasks arrive within $\mathbb{T}$.

As shown in lines 20–26 in Figure 5.5, the internal fitness of the generated offspring is computed based on the **P** population. OPAM then updates the **A** population of task-arrival sequences by comparing the offspring and individuals in **A** (line 27).

We note that when a system is only composed of periodic tasks, OPAM will skip evolving for worst-case arrival sequences as arrivals of periodic tasks are deterministic (see Section 5.3). Nevertheless,

```
1  Algorithm Priority assignments evolution
2  Input A: population of task-arrival sequences
3  Input P: population of priority assignments
4  Input ps_p: population size //priority assignments
5  Input cp_p: crossover probability //priority assignments
6  Input mp_p: mutation probability //priority assignments
7  Output P: population of priority assignments
8
9  //evaluate internal fitness values for P
10 for each P⃗_x ∈ P
11   for each A_y ∈ A
12     S ← simulate(A_y, P⃗_x) //OPAMScheduler
13     //dist(τ_i, k) is computed based on S
14     fs(P⃗_x, A_y) = Σ_{τ_i∈Γ, k∈[1,lk(τ_i)]} −2^{dist(τ_i,k)}
15   fs(P⃗_x, A) = Σ_{A_y∈A} fs(P⃗_x, A_y)/|A|
16   fc(P⃗_x) = Σ_{τ_i∈Γ} { lp(P⃗_x) − P_i, if τ_i is an aperiodic task
                           { 0, otherwise
17
18 //breed priority assignments
19 R⃗ ← sort_non_dominated_fronts(P)
20 assign_crowding_distance(R⃗)
21 P_α ← NSGAII_breed(R⃗, ps_p, cp_p, mp_p)
22 //evaluate internal fitness values for P_α
23 for each P⃗_x ∈ P_α
24   for each A_y ∈ A
25     S ← simulate(A_y, P⃗_x) //OPAMScheduler
26     //dist(τ_i, k) is computed based on S
27     fs(P⃗_x, A_y) = Σ_{τ_i∈Γ, k∈[1,lk(τ_i)]} −2^{dist(τ_i,k)}
28   fs(P⃗_x, A) = Σ_{A_y∈A} fs(P⃗_x, A_y)/|A|
29   fc(P⃗_x) = Σ_{τ_i∈Γ} { lp(P⃗_x) − P_i, if τ_i is an aperiodic task
                           { 0, otherwise
30 R⃗ ← sort_non_dominated_fronts(P ∪ P_α)
31 assign_crowding_distance(R⃗)
32 P ← select_archive(R⃗, ps_p)
33
34 return P
```

Figure 5.6: An NSGAII-based algorithm for evolving priority assignments.

OPAM will optimize priority assignments based on given arrivals of periodic tasks. When needed, OPAM can be easily extended to manipulate offset and period values for periodic tasks, in a way identical to how we currently handle inter-arrival times for aperiodic tasks.

5.6.5 Evolution: best-case priority assignments

Figure 5.6 shows the evolution procedure of priority assignments, which refines lines 23–26 in Figure 5.4. OPAM tailors the Non-dominated Sorting Genetic Algorithm version 2 (NSGAII) [63] to generate a non-dominating (equally viable) set of priority assignments, representing the best trade-offs found among the given internal fitness functions. This is referred to as a Pareto nondominated front [102], where the dominance relation over priority assignments is defined as follows: A priority assignment $\vec{P}$ dominates another priority assignment $\vec{P}'$ if $\vec{P}$ is not worse than $\vec{P}'$ in all fitness values, and $\vec{P}$ is strictly better than $\vec{P}'$ in at least one fitness value. NSGAII has been applied to many multi-objective optimization problems [107, 158, 178].

```
1  Algorithm Priority assignments evolution
2  Input E: set of task-arrival sequences //external evaluation
3  Input P: population of priority assignments
4  Input ps_p: population size //priority assignments
5  Input cp_p: crossover probability //priority assignments
6  Input mp_p: mutation probability //priority assignments
7  Output P: population of priority assignments
8
9  //evaluate external fitness values for P
10 for each P⃗_x ∈ P
11 |  for each E_y ∈ E
12 |  |  S ← simulate(E_y, P⃗_x) //OPAMScheduler
13 |  |  //dist(τ_i, k) is computed based on S
14 |  |  fs(P⃗_x, E_y) = Σ_{τ_i∈Γ, k∈[1,lk(τ_i)]} −2^{dist(τ_i,k)}
15 |  fs(P⃗_x, E) = Σ_{E_y∈E} fs(P⃗_x, E_y)/|E|
16 |  fc(P⃗_x) = Σ_{τ_i∈Γ} { lp(P⃗_x) − P_i, if τ_i is an aperiodic task
   |                      { 0, otherwise
17 R⃗ ← sort_non_dominated_fronts(P ∪ B)
18 assign_crowding_distance(R⃗)
19 B ← select_best_front(R⃗) //|B| ≤ |P|
20
21 return B
```

Figure 5.7: An algorithm for evaluating external fitness and finding the best Pareto front.

OPAM maintains a population **P** of priority assignments as an archive that contains the best priority assignments discovered during coevolution. Unlike a standard application of NSGAII, in our study, we need to reevaluate the internal fitness values for priority assignments in **P** at every generation as the internal fitness values are computed based on the **A** population of task-arrival sequences, which coevolves. As shown in lines 9–16 in Figure 5.6, OPAM first computes the internal fitness functions that measure the magnitude of safety margins and the extent of constraint satisfaction. OPAM then sorts non-dominated Pareto fronts (line 19) and assigns crowding distance (line 20) to introduce diversity among non-dominated priority assignments [63].

For breeding the next population of priority assignments (line 21 in Figure 5.6, OPAM applies the following standard genetic operators [159] that have been applied to many similar problems [99, 127, 158]: (1) *Selection.* OPAM uses a binary tournament selection based on non-domination ranking and crowding distance. The binary tournament selection has been used in the original implementation of NSGAII [63]. (2) *Crossover.* OPAM applies a partially mapped crossover (PMX) [80]. PMX ensures that the generated offspring are valid permutations of priorities. (3) *Mutation.* OPAM uses a permutation swap method for mutating a priority assignment. This mutation method interchanges two randomly-selected priorities in a priority assignment according to a given mutation probability.

For the generated population $\mathbf{P}_\alpha$ of priority assignments, OPAM computes the two internal fitness functions (lines 22–29 in Figure 5.6). OPAM then sorts non-dominated Pareto fronts for the union of the current **P** and next $\mathbf{P}_\alpha$ populations (line 30), assign crowding distance (line 31), and select the best archive by accounting for the computed non-domination ranking and crowding distance (line 32).

5.6.6 External fitness evaluation

Figure 5.7 shows an algorithm that computes the external fitness functions and finds the best Pareto front, which refines lines 28–31 in Figure 5.4. To monitor the coevolution progress in a stable manner,

OPAM takes as input a set **E** of task-arrival sequences that are generated independently from the coevolution process. We use an adaptive random search technique [45] to sample task-arrival sequences in order to create **E**. The adaptive random search extends the naive random search by maximizing the Euclidean distance between the sampled points such that it maximizes the diversity of task-arrival sequences in **E**.

As shown in lines 9–16 in Figure 5.7, OPAM computes the two external fitness values for each priority assignment in the **P** population based on a given set **E** of task-arrival sequences. OPAM then sorts non-dominated Pareto fronts for the union of the **P** population and the current best Pareto front (line 17), assigns crowding distance (line 18), and selects the best Pareto front by accounting for the computed non-domination ranking and crowding distance (line 32). OPAM adopts NSGAII in order to maximize the diversity of priority assignments in the best Pareto front.

5.7 Evaluation

This section describes our evaluation of OPAM through six industrial case studies from different domains and several synthetic subjects. Our full evaluation package is available online [109].

5.7.1 Research questions

RQ1 (sanity check): *How does OPAM perform compared with Random Search?* For search-based solutions, this RQ is an important *sanity check* to ensure that success is not due to the search problem being easy [12]. Our conjecture is that a search-based algorithm, although expensive, will significantly outperform naive random search (RS).

RQ2 (coevolution): *Is competitive coevolution suitable to find best-case priority assignments?* We conjecture that a coevolutionary algorithm is a suitable solution to address the priority assignment problem since it is solved, in practice, through a competing interactive process between the development and testing teams. To answer this RQ, we compare OPAM with a sequential approach that first looks for worst-case sequences of task arrivals and then tries to find best-case priority assignments.

RQ3 (scalability): *Can OPAM find (near-)optimal solutions for large-scale systems in a reasonable time budget?* In this RQ, we investigate the scalability of OPAM by conducting some experiments with systems of various sizes, including six industrial and several synthetic subjects. We study the relationship between OPAM's performance measures and the characteristics of study subjects.

RQ4 (usefulness): *How do priority assignments generated by OPAM compare with priority assignments defined by engineers?* OPAM can be considered useful only when it finds priority assignments that show benefits over those defined (manually) by engineers with domain expertise. This RQ therefore compares the quality of priority assignments generated by OPAM with those defined by engineers. We further discuss the usefulness of OPAM from a practical perspective, based on the feedback received from engineers in LuxSpace.

5.7.2 Industrial study subjects

To evaluate RQs in realistic and diverse settings, we apply OPAM to six industrial study subjects from different domains such as aerospace, automotive, and avionics domains. Specifically, we obtained one case study subject from our industry partner, LuxSpace. We found the other five industrial study subjects in the

Table 5.2: Description of the six industrial subject systems: number of periodic and aperiodic tasks, resource dependencies, triggering relations, and platform cores.

	Task types		Relationships		Platform
System	Periodic	Aperiodic	Dependencies	Triggering	Cores
ICS	3	3	3	0	3
CCS	8	3	3	6	2
UAV	12	4	4	0	3
GAP	15	8	6	5	2
HPSS	23	9	5	0	1
ESAIL	11	14	0	0	1

literature [64], which, consistent with the LuxSpace system, all assume a single-queue, multi-core, fixed-priority scheduling policy. Note that OPAM uses the same scheduling policy (described in Section 5.6.2) as in Alesio's work [64]. This policy uses fixed priorities that are determined offline and therefore do not change dynamically. Table 5.2 summarizes the relevant attributes of these subjects, presenting the number of periodic and aperiodic tasks, resource dependencies, triggering relations, and platform cores. The subjects are characterized by real-time parameters, e.g., periods, deadlines, and priorities, described in Section 5.3. We note that all the study subjects are deadlock-free systems as they do not have circular resource dependencies. Regarding task priorities, all tasks in the six subjects have fixed priorities, which are defined by experts in their domains. The full task descriptions (including WCET, inter-arrival times, periods, deadlines, priorities, and relationship details) of the subjects are available online [109]. The main missions of the six subjects are described as follows:

- ICS is an ignition control system that checks the status of an automotive engine and corrects any errors of the engine [145]. The system was developed by Bosch GmbH[1].

- CCS is a cruise control system that acquires data from vehicle sensors and maintains the specified vehicle speed [11]. Continental AG[2] developed the system.

- UAV is a mini unmanned air vehicle that follows dynamically defined way-points and communicates with a ground station to receive instructions [171]. The system was developed in collaboration with the University of Poitiers France and ENSMA[3].

- GAP is a generic avionics platform for a military aircraft [120]. The system was designed in a joint project with Carnegie Mellon University, the US Navy, and IBM[4], aiming at supporting several missions regarding air-to-surface attacks.

- HPSS is a satellite system for two satellites, named Herschel and Planck [132]. The two satellites share the same computational architecture, although they have different scientific missions. Herschel aims at

[1]Bosch GmbH: https://www.bosch.com/
[2]Continental AG: https://www.continental.com
[3]ENSMA: https://www.ensma.fr/
[4]IBM: https://www.ibm.com/

```
1  Algorithm Synthetic task generation
2  Input n: number of tasks
3  Input u^t: target utilization
4  Input T^min: minimum task period
5  Input T^max: maximum task period
6  Input g: granularity of task periods
7  Input γ: ratio of aperiodic tasks
8  Input μ: range factor to determine maximum inter-arrival times
9  Output Γ: set of tasks
10
11 Γ ← {}, C ← {}
12 // synthesize a set of periodic tasks
13 U ← UUniFast_discard(n, u^t) // task utilizations
14 T ← generate_task_periods(n, T^min, T^max, g) // task periods
15 for each i ∈ [1, n]
16 |  C ← C ∪ {U_i·T_i}, where U_i ∈ U and T_i ∈ T // WCETs
17 Γ ← generate_task_set(T, C) // set of tasks
18 // convert some periodic tasks to aperiodic tasks
19 Γ ← convert_aperiodic_tasks(Γ, γ, μ)
20
21 return Γ
```

Figure 5.8: An algorithm for synthesizing a set of tasks.

studying the origin and evolution of stars and galaxies. Planck's primary mission is the study of the relic radiation from the Big Bang. ESA[5] carried out the HPSS project.

- ESAIL is a microsatellite for tracking ships worldwide by detecting messages that ships radio-broadcast (see Section 5.2). Luxspace, our industry partner, developed ESAIL in an ESA project.

5.7.3 Synthetic study subjects

To investigate RQ3, we use synthetic subjects in order to freely control key parameters in real-time systems. We create a set of tasks by adopting a well-known procedure [70] for synthesizing real-time tasks, which has been applied in many schedulability analysis studies [62, 195, 59, 82, 68].

Figure 5.8 describes a procedure that synthesizes a set of real-time tasks. For a given number n of tasks and a target utilization u^t, the procedure first generates a set $\mathbf{U}$ of task utilization values by using the UUniFast-Discard algorithm [59] (line 13). The UUniFast-Discard algorithm is devised to give an unbiased distribution of utilization values, where a utilization $U_i \in \mathbf{U}$ is a positive value and $\sum_{U_i \in \mathbf{U}} U_i = u^t$.

The procedure then generates a set $\mathbf{T}$ of n task periods according to a log-uniform distribution within a range $[T^{min}, T^{max}]$, i.e., given a task period (random variable) $T_i \in \mathbf{T}$, $\log T_j$ follows a uniform distribution (line 14 in Figure 5.8). For example, when the minimum and maximum task periods are $T^{min} = 10$ms and $T^{max} = 1000$ms, respectively, the procedure generates (approximately) an equal number of tasks in time intervals [10ms, 100ms] and [100ms, 1000ms]. The parameter g is used to choose the granularity of the periods, i.e., task periods are multiples of g. Such a distribution of task periods provides a reasonable degree of realism with respect to what is usually observed in real systems [23].

[5]ESA: https://www.esa.int/

As shown in lines 15–16 of the procedure in Figure 5.8, a set **C** of task WCETs are computed based on the set **U** of task utilization values and the set **T** of task periods. Specifically, a task WCET $C_i \in \mathbf{C}$ is computed as $C_i = U_i \cdot T_i$.

As per line 17 of the listing in Figure 5.8, the procedure synthesizes a set $\mathbf{\Gamma}$ of tasks. A task τ_i is characterized by a period T_i and a WCET C_i and it is associated with a deadline D_i and a priority P_i. According to the rate-monotonic scheduling policy [118], tasks' deadlines are equal to their periods and tasks with shorter periods are given higher priorities.

To synthesize aperiodic tasks, the procedure converts some periodic tasks to aperiodic tasks according to a given ratio γ of aperiodic tasks among all tasks (see line 19 in Figure 5.8). A range factor μ is used to determine maximum inter-arrival times of aperiodic tasks. Specifically, for a task τ_i to be converted, the procedure sets the minimum inter-arrival time T_i^{min} as $T_i^{min} = T_i$. The procedure then selects a uniformly distributed value x from the range $(1, \mu]$ and computes the maximum inter-arrival time T_i^{max} as $T_i^{max} = x \cdot T_i$.

5.7.4 Experimental design

This section describes how we design experiments to answer the RQs described in Section 5.7.1. We conducted four experiments, EXP1, EXP2, EXP3, and EXP4, as described below.

EXP1. To answer RQ1, EXP1 compares OPAM with our baseline, which relies on random search, to ensure that the effectiveness of OPAM is not due to the search problem being simple. Our baseline, named RS, replaces GA with a random search for finding worst-case sequences of task arrivals and NSGAII with a random search for finding best-case priority assignments. Note that RS uses the same internal and external fitness functions (see Section 5.6.3) and also maintains the best populations during search; however, it does not employ any genetic operators, i.e., crossover and mutation. In EXP1, we applied OPAM and RS to the six industrial subjects described in Section 5.7.2.

Recall from Section 5.6.3 that OPAM uses a set **E** of task-arrival sequences that are generated independently from the coevolution process in order to monitor the coevolution progress in a stable manner. As OPAM and RS use the same set **E** of task-arrival sequences, EXP1 first compares OPAM and RS based on **E**. In addition, EXP1 examines how well the solutions, i.e., priority assignments, found by OPAM and RS perform with other sequences of task arrivals. To do so, we create six sets of sequences of task arrivals for each study subject by varying the method to generate task-arrival sequences and the number of task-arrival sequences. Note that task-arrival sequences generated by different methods are valid with respect to the inter-arrival times defined in each study subject. Below we describe the six sets of task-arrival sequences generated for each subject.

- $\mathbf{T}_a^{10}$: A set of task-arrival sequences generated by using an adaptive random search technique [45] that aims at maximizing the diversity of task-arrival sequences. The $\mathbf{T}_a^{10}$ set contains 10 sequences of task arrivals.

- $\mathbf{T}_w^{10}$: A set of task-arrival sequences generated by using a stress test case generation method that aims at maximizing the chances of deadline misses in task executions. The stress test case generation method extends prior work [36]. The extended method uses the fitness function regarding deadline misses and genetic operators that OPAM introduces for evolving worst-case task-arrival sequences (see Section 5.6). The $\mathbf{T}_w^{10}$ set contains 10 sequences of task arrivals.

- $\mathbf{T}_r^{10}$: A set of task-arrival sequences generated randomly. The $\mathbf{T}_r^{10}$ set has 10 sequences of task arrivals.
- $\mathbf{T}_a^{500}$: A set of task-arrival sequences generated by using the adaptive random search technique. The $\mathbf{T}_a^{500}$ set contains 500 sequences of task arrivals.
- $\mathbf{T}_w^{500}$: A set of task-arrival sequences generated by using the stress test case generation method. The $\mathbf{T}_w^{500}$ set contains 500 sequences of task arrivals.
- $\mathbf{T}_r^{500}$: A set of task-arrival sequences generated randomly. The $\mathbf{T}_r^{500}$ set has 500 sequences of task arrivals.

EXP2. To answer RQ2, EXP2 compares OPAM with a priority assignment method, named SEQ, that relies on one-population search algorithms. SEQ first finds a set of worst-case sequences of task arrivals using GA with the fitness function that measures the magnitude of deadline misses (see $fd()$ in Section 5.6.3) and the genetic operators described in Section 5.6.4. Given a set of worst-case task-arrival sequences obtained from GA, SEQ then aims at finding best-case priority assignments using NSGAII with the fitness functions that quantify the magnitude of safety margins and the degree of constraint satisfaction (see $fs()$ and $fc()$, respectively, in Section 5.6.3) and the genetic operators described in Section 5.6.5.

We note that SEQ does not use the external fitness functions as it does not coevolve task-arrival sequences and priority assignments. Hence, the numbers of fitness evaluations of the two methods are not comparable. To fairly compare OPAM and SEQ, we set the same time budget for the two methods. Specifically, we first measure the execution time of OPAM for analyzing each subject. We then split the execution time in half and set each half time as the execution budget of the GA and NSGAII steps in SEQ for the corresponding subject. In order to assess the quality of priority assignments obtained from OPAM and SEQ, we use the sets of task-arrival sequences described in EXP1, i.e., $\mathbf{T}_a^{10}$, $\mathbf{T}_w^{10}$, $\mathbf{T}_r^{10}$, $\mathbf{T}_a^{500}$, $\mathbf{T}_w^{500}$, and $\mathbf{T}_r^{500}$, which are created independently from the two methods.

EXP3. To answer RQ3, EXP3 examines not only the six industrial subjects but also 370 synthetic subjects. We create the synthetic subjects to study correlations between the execution time and memory usage of OPAM and the following parameters: the number of tasks (n), a (part-to-whole) ratio of aperiodic tasks (γ), a range factor for maximum inter-arrival times (μ), and simulation time ($\mathbf{t}$), as described in Sections 5.7.3 and 5.6. We note that we chose to control parameters n, γ, and μ because they are the main parameters on which engineers have control to define tasks in real-time systems. Simulation time $\mathbf{t}$ obviously impacts the execution time of OPAM as well. But EXP3 aims at modeling such correlations precisely and providing experimental results. Regarding the other factors that define, for example, task relationships and platform cores, we note significant diversity across the six industrial subjects.

Recall from Section 5.7.3 that we use the task generation procedure presented in Figure 5.8 to synthesize tasks. For EXP3, we set some parameter values of the procedure as follows: (1) Target utilization $u^t = 0.7$, which is a common objective in the development of a real-time system in order to guarantee the schedulability of tasks [76, 68]. (2) The range of task periods $[T^{min}, T^{max}] = [10\text{ms}, 1\text{s}]$, which are common values in many real-time systems [70, 23]. (3) The granularity of task periods $g = 10\text{ms}$ in order to increase realism as most of the task periods in our industrial subjects are multiples of 10ms. Because of some degree of randomness in the procedure of Figure 5.8, we create ten synthetic subjects per configuration. Below we further describe how synthetic subjects are created for each controlled experiment.

EXP3.1. To study the correlations between the execution time and memory usage of OPAM with the number of tasks n, we create nine sets of ten synthetic subjects such that no two sets have the same number of tasks. Specifically, we create sets with 10, 15, ..., 50 tasks, respectively. Regarding the ratio of aperiodic tasks, $\gamma = 0.4$ as, on average, the ratio of aperiodic tasks to periodic tasks in our industrial subjects is 2/3. For the range factor, $\mu = 2$, which is determined based on the inter-arrival times of aperiodic tasks in our industry subjects. We set the simulation time **t** to 2s in order to ensure that any aperiodic task arrives at least once during that time. We note that, given the maximum task period $T^{max} = 1$s and the range factor $\mu = 2$, the maximum inter-arrival time of an aperiodic task is at most 2s (see Section 5.7.3).

EXP3.2. To study the correlations between the execution time and memory usage of OPAM with the ratio of aperiodic tasks γ, we create ten sets of synthetic subjects by setting this ratio to the following values: 0.05, 0.10, ..., 0.50. We set the number of tasks to 20 ($n = 20$), which is the average number of tasks in our six industrial subjects. Regarding the other parameters, range factor and simulation time, $\mu = 2$ and **t** = 2s are set as discussed in EXP3.1.

EXP3.3. To study the correlations between the execution time and memory usage of OPAM with the range factor μ that is used to determine the maximum inter-arrival times, we create nine sets of synthetic subjects by setting μ to 2, 3, ..., 10. We set the simulation time as follows: **t** = 10s. This ensures that any aperiodic task arrives at least once during the simulation time when μ is at most 10 (see Section 5.7.3). The other parameters, the number of tasks and ratio of aperiodic tasks, $n = 20$ and $\gamma = 0.4$ are set as discussed in EXP3.1 and EXP3.2.

EXP3.4. To study the correlations between the execution time and memory usage of OPAM with the simulation time **t**, we create nine sets of synthetic subjects by setting **t** to 2s, 3s, ..., 10s. The other parameters, e.g., the number of tasks, the ratio of aperiodic tasks, and the range factor, $n = 20$, $\gamma = 0.4$, and $\mu = 2$, are set as discussed in EXP3.1 and EXP3.2.

EXP4. To answer RQ4, EXP4 compares priority assignments optimized by OPAM and those defined by engineers. We apply OPAM to the six industrial subjects (see Section 5.7.2) which include priority assignments defined by practitioners. Note that we focus here on the ESAIL subject in collaboration with our industry partner, LuxSpace; The other five subjects are from the literature [64] and hence we can only collect feedback from practitioners for ESAIL.

5.7.5 Evaluation metrics

Multi-objective evaluation metrics. In order to fairly compare the results of search algorithms, based on existing guidelines [116] for assessing multi-objective search algorithms, we use complementary quality indicators: *Hypervolume* (HV) [199], *Pareto Compliant Generational Distance* (GD+) [98], and *Spread* (Δ) [63]. To compute the GD+ and Δ quality indicators, following the usual procedure [116], we create a reference Pareto front as the union of all the non-dominated solutions obtained from all runs of the algorithms being compared. Identifying the optimal (ideal) Pareto front is typically infeasible for a complex optimization problem [116]. Key features of the three quality indicators are described below.

- HV is defined to measure the volume in the objective space that is covered by members of a Pareto front generated by a search algorithm [199]. The higher the HV values, the more optimal the search outputs.
- GD+ is defined to measure the distance between the points on a Pareto front obtained from a search algorithm and the nearest points on a reference Pareto front [98]. GD+ modifies General Distance

(GD) [174] to account for the dominance relations when computing the distances. The lower the GD+ values, the more optimal the search outputs.

- Δ is defined to measure the extent of spread among the points on a Pareto front computed by a search algorithm [63]. We note that OPAM aims at obtaining a wide variety of equally-viable priority assignments on a Pareto front (see Section 5.6). The lower the Spread values, the more spread out the search outputs.

Interpretable metrics. The two external fitness functions described in Section 5.6 mainly aim at effectively guiding search. It is, however, difficult for practitioners to interpret the computed fitness values. Since they are not intuitive to practitioners, to assess the usefulness of OPAM from a practitioner perspective, we measure (1) the safety margins from tasks' completion times to their deadlines across our experiments and (2) the number of constraint violations in a priority assignment. In addition, we measure the execution time and memory usage of OPAM.

Statistical comparison metrics. To statistically compare our experiment results, we use the Mann-Whitney U-test [125] and Vargha and Delaney's $\hat{A}_{12}$ effect size [172], which have been frequently applied for evaluating search-based algorithms [14, 95, 158]. Mann-Whitney U-test determines whether two independent samples are likely or not to belong to the same distribution. We set the level of significance, α, to 0.05. Vargha and Delaney's $\hat{A}_{12}$ measures probabilistic superiority – effect size – between search algorithms. Two algorithms are considered to be equivalent when the value of $\hat{A}_{12}$ is 0.5.

5.7.6 Parameter tuning and implementation

Parameters for coevolutionary search. For the coevolutionary search parameters, we set the population size to 10, the crossover rate to 0.8, and the mutation rate to $1/|\Gamma|$, where $|\Gamma|$ denotes the number of tasks. We apply these parameter values for both the evolution of task-arrival sequences and priority assignments (see Section 5.6). These values are determined based on existing guidelines [13, 154] and the Chapter 3.

We determine the number of coevolution cycles (see Section 5.6) based on an initial experiment. We applied OPAM to the six industrial subjects and ran OPAM 50 times for each subject. From the experiment results, we observed that there is no notable difference in Pareto fronts generated after 1000 cycles. Hence, we set the number of coevolution cycles to 1000 in our experiments, i.e., EXP1, EXP2, and EXP3 described in Section 5.7.4.

Parameters for evaluating fitness functions. To evaluate external fitness functions, we use a set of task-arrival sequences that are generated independently from the coevolution process (see Section 5.6.6). We use an adaptive random search [45] to generate a set **E** of task-arrival sequences, which varies task arrival times within the specified inter-arrival time ranges of aperiodic tasks. We set the size of **E** to 10. From our initial experiment, we observed that this is sufficient to compute the external fitness functions of OPAM under a reasonable time, i.e., less than 15s. We note that **E** contains two default sequences of task arrivals as follows: (seq. 1) aperiodic tasks always arrive at their maximum inter-arrival times and (seq. 2) aperiodic tasks always arrive at their minimum inter-arrival times. By having those two sequences of task arrivals as initial elements in **E**, the adaptive random search finds other sequences of task arrivals to maximize the diversity of elements in **E**.

If a system contains only periodic tasks, the simulation time is often set as the least common multiple (LCM) of their periods to account for all possible arrivals [144]. However, as the six industrial subjects

include aperiodic tasks, this is not applicable. For the experiments with the six industrial subjects, we set the simulation time to the maximum time between the LCM of periodic tasks' periods and the maximum inter-arrival time among aperiodic tasks. By doing so, all possible arrival patterns of periodic tasks are examined and any aperiodic task arrives at least once during simulation. Recall from Section 5.6.4 that OPAM varies arrival times of aperiodic tasks to find worst-case sequences of task arrivals.

We note that the parameters mentioned above can probably be further tuned to improve the performance of our approach. However, since with our current setting, we were able to convincingly and clearly support our conclusions, we do not report further experiments on tuning those values.

Implementation. We implemented OPAM by extending jMetal [67], which is a metaheuristic optimization framework supporting NSGAII and GA. We conducted our experiments using the high-performance computing cluster [173] at the University of Luxembourg. To account for randomness, we repeated each run of OPAM 50 times for all experiments. Each run of OPAM was executed on a different node (equipped with five 2.5GHz cores and 20GB memory) of the cluster, and took less than 16 hours.

5.7.7 Results

RQ1. Figure 5.9 shows the best Pareto fronts obtained with 50 runs of OPAM and RS, for the six industrial study subjects described in Section 5.7.2. The fitness values presented in the figures are computed based on each subject's set **E** of task-arrival sequences (see Section 5.7.6), which is created independently from OPAM and RS. Figures 5.9a, 5.9c, 5.9d, 5.9e, and 5.9f indicate that OPAM finds significantly better solutions than RS for ICS, UAV, GAP, HPSS, and ESAIL. Regarding CCS (see Figure 5.9b), it is difficult to conclude anything based only on visual inspection. Hence, we compared Pareto fronts obtained by OPAM and RS using the three quality indicators HV, GD+, and Δ, described in Section 5.7.5.

Figure 5.10 depicts distributions of HV (Figure 5.10a), GD+ (Figure 5.10b), and Δ (Figure 5.10c) for the six industrial subjects. The boxplots in the figures present the distributions (25%-50%-75%) of the quality values obtained from 50 runs of OPAM and RS. The quality values are computed based on the Pareto fronts obtained by the algorithms and each subject's set **E** of task-arrival sequences (see Section 5.7.6). In the figures, statistical comparisons of the two corresponding distributions are summarized using p-values and $\hat{A}_{12}$ values, as described in Section 5.7.5, under each subject name.

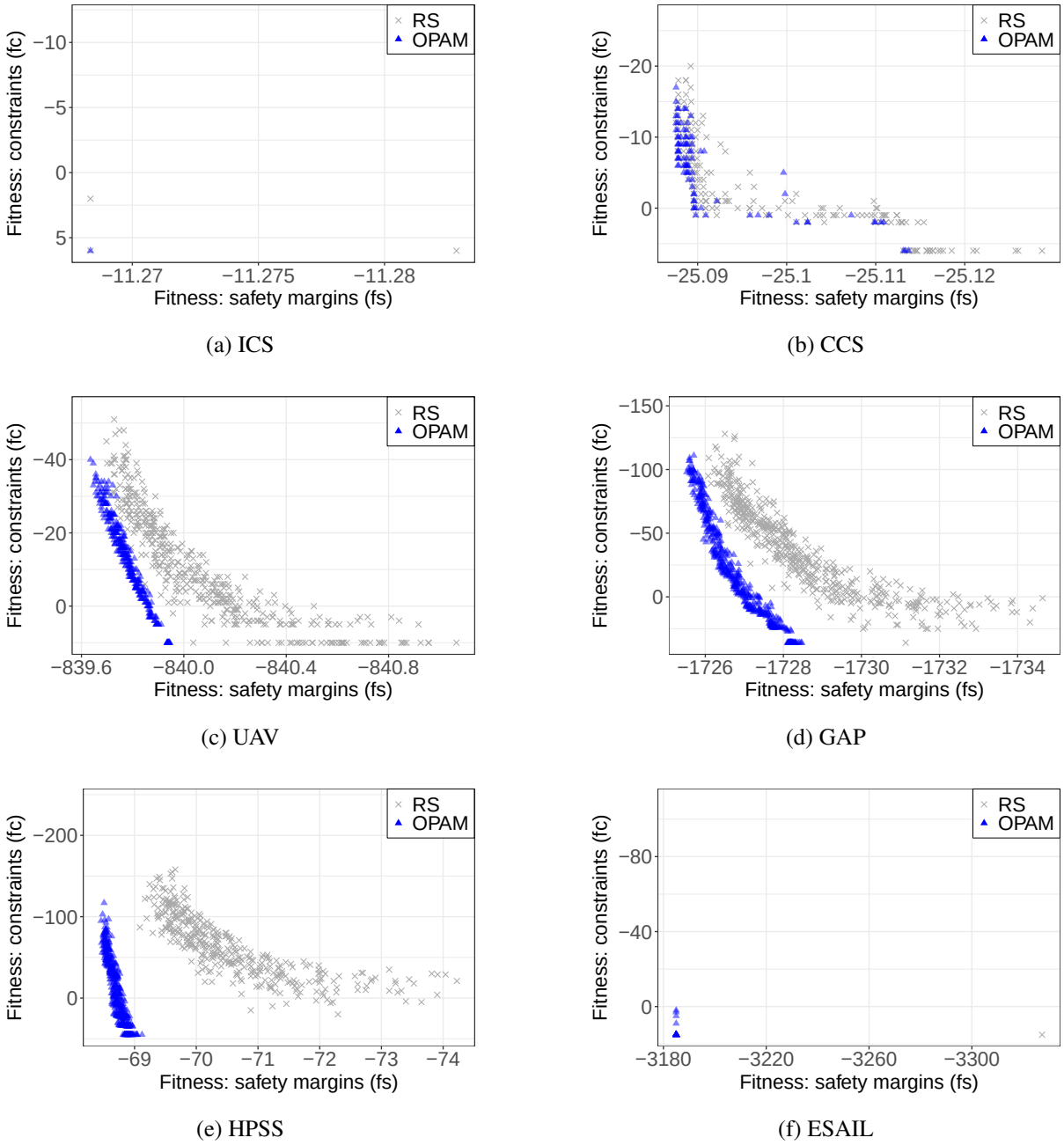
As shown in Figures 5.10a and 5.10b, OPAM obtains better distributions of HV and GD+ compared to RS for all six subjects. All the differences are statistically significant as the p-values are below 0.05. Regarding Δ, as depicted in Figure 5.10c, OPAM yields higher diversity in Pareto front solutions than RS for the following subjects: UAV, GAP, and HPSS. For ICS, CCS, and ESAIL, OPAM and RS obtain similar Δ values. From Figures 5.10a and 5.10b, and Table 5.2, we also observe that the higher the number of aperiodic tasks in a subject, the larger the differences in HV and GD+ between OPAM and RS. Hence, for these two quality indicators, OPAM outperforms RS more significantly for more complex search problems. Note that the number of aperiodic tasks is one of the main factors that drives the degree of uncertainty in task arrivals.

Given the Pareto priority assignments obtained by OPAM and RS, we further assessed the quality values of the solutions by evaluating them with different sets of task-arrival sequences. As described in Section 5.7.4, we created six test sets of task-arrival sequences for each subject by varying the sequence generation methods and the number of task-arrival sequences in a set (see $\mathbf{T}_a^{10}$, $\mathbf{T}_w^{10}$, $\mathbf{T}_r^{10}$, $\mathbf{T}_a^{500}$, $\mathbf{T}_w^{500}$, and $\mathbf{T}_r^{500}$ described in Section 5.7.4). Table 5.3 reports the average quality values measured by HV, GD+, and

Table 5.3: Comparing OPAM and RS using the three quality indicators: HV, GD+, and Δ. Average quality values computed based on 50 runs of OPAM and RS using the different sets of task-arrival sequences (see Section 5.7.4).

			ICS	CCS	UAV	GAP	HPSS	ESAIL
$\mathbf{T}^{10}_{a}$ (adaptive, size 10)	HV	**OPAM**	**1.0000**	**0.7168**	**0.8923**	**0.8864**	**0.9629**	**0.9998**
		RS	0.9000	0.6633	0.7488	0.6278	0.5120	0.0000
		$p\|\hat{A}_{12}$	0.02|0.55	0.00|0.80	0.00|1.00	0.00|1.00	0.00|1.00	0.00|1.00
	GD+	**OPAM**	**0.0000**	**0.0203**	**0.0068**	**0.0067**	**0.0073**	**0.0135**
		RS	0.0883	0.0472	0.0745	0.0780	0.1380	1.0000
		$p\|\hat{A}_{12}$	0.02|0.45	0.00|0.04	0.00|0.00	0.00|0.00	0.00|0.00	0.00|0.00
	Δ	**OPAM**	1.0000	0.7650	**0.4256**	**0.3631**	**0.5355**	0.9433
		RS	0.9766	**0.5879**	0.6112	0.6605	0.7508	1.0000
		$p\|\hat{A}_{12}$	0.16|0.52	0.00|0.76	0.00|0.15	0.00|0.03	0.00|0.12	0.08|0.47
$\mathbf{T}^{10}_{w}$ (worst, size 10)	HV	**OPAM**	0.0000	**0.7878**	**0.9152**	**0.9280**	**0.9652**	**0.9997**
		RS	0.0000	0.7591	0.7782	0.6743	0.5180	0.0000
		$p\|\hat{A}_{12}$	1.00|0.50	0.01|0.65	0.00|1.00	0.00|1.00	0.00|1.00	0.00|1.00
	GD+	**OPAM**	0.0000	0.0809	**0.0053**	**0.0042**	**0.0108**	**0.0135**
		RS	0.0200	0.0866	0.0740	0.0760	0.1405	1.0000
		$p\|\hat{A}_{12}$	0.16|0.48	0.75|0.52	0.00|0.00	0.00|0.00	0.00|0.00	0.00|0.00
	Δ	**OPAM**	1.0000	0.7012	**0.4508**	**0.4009**	**0.4872**	0.9433
		RS	0.9600	**0.4764**	0.6032	0.7002	0.7328	1.0000
		$p\|\hat{A}_{12}$	0.16|0.52	0.00|0.79	0.00|0.22	0.00|0.03	0.00|0.11	0.08|0.47
$\mathbf{T}^{10}_{r}$ (random, size 10)	HV	**OPAM**	0.0000	**0.8976**	**0.9792**	**0.9449**	**0.9837**	**0.9999**
		RS	0.0000	0.8517	0.8191	0.6879	0.5183	0.0000
		$p\|\hat{A}_{12}$	1.00|0.50	0.00|0.90	0.00|1.00	0.00|1.00	0.00|1.00	0.00|1.00
	GD+	**OPAM**	0.0000	**0.0806**	**0.0035**	**0.0043**	**0.0211**	**0.0134**
		RS	0.0200	0.1252	0.0912	0.0789	0.1580	1.0000
		$p\|\hat{A}_{12}$	0.16|0.48	0.00|0.09	0.00|0.00	0.00|0.00	0.00|0.00	0.00|0.00
	Δ	**OPAM**	1.0000	0.8662	**0.4603**	**0.3951**	**0.4728**	0.9433
		RS	0.9600	**0.6579**	0.6331	0.7035	0.7617	1.0000
		$p\|\hat{A}_{12}$	0.16|0.52	0.00|0.73	0.00|0.20	0.00|0.02	0.00|0.05	0.08|0.47
$\mathbf{T}^{500}_{a}$ (adaptive, size 500)	HV	**OPAM**	**1.0000**	**0.7032**	**0.9424**	**0.9089**	**0.9803**	**0.9999**
		RS	0.9000	0.6518	0.7893	0.6561	0.5167	0.0000
		$p\|\hat{A}_{12}$	0.02|0.55	0.00|0.86	0.00|1.00	0.00|1.00	0.00|1.00	0.00|1.00
	GD+	**OPAM**	**0.0000**	**0.0159**	**0.0035**	**0.0051**	**0.0064**	**0.0134**
		RS	0.0883	0.0393	0.0850	0.0746	0.1422	1.0000
		$p\|\hat{A}_{12}$	0.02|0.45	0.00|0.03	0.00|0.00	0.00|0.00	0.00|0.00	0.00|0.00
	Δ	**OPAM**	1.0000	0.7842	**0.4715**	**0.3680**	**0.4850**	0.9433
		RS	0.9766	**0.5354**	0.6357	0.6850	0.7565	1.0000
		$p\|\hat{A}_{12}$	0.16|0.52	0.00|0.84	0.00|0.21	0.00|0.01	0.00|0.09	0.08|0.47
$\mathbf{T}^{500}_{w}$ (worst, size 500)	HV	**OPAM**	**1.0000**	**0.6535**	**0.9223**	**0.9307**	**0.9635**	**0.9997**
		RS	0.9000	0.6050	0.7791	0.6770	0.5032	0.0000
		$p\|\hat{A}_{12}$	0.02|0.55	0.00|0.77	0.00|1.00	0.00|1.00	0.00|1.00	0.00|1.00
	GD+	**OPAM**	**0.0000**	**0.0302**	**0.0037**	**0.0040**	**0.0054**	**0.0136**
		RS	0.0883	0.0545	0.0768	0.0763	0.1408	1.0000
		$p\|\hat{A}_{12}$	0.02|0.45	0.00|0.09	0.00|0.00	0.00|0.00	0.00|0.00	0.00|0.00
	Δ	**OPAM**	1.0000	0.7899	**0.4640**	**0.4077**	**0.5083**	0.9433
		RS	0.9766	**0.5910**	0.6114	0.7052	0.7448	1.0000
		$p\|\hat{A}_{12}$	0.16|0.52	0.00|0.84	0.00|0.22	0.00|0.02	0.00|0.11	0.08|0.47
$\mathbf{T}^{500}_{r}$ (random, size 500)	HV	**OPAM**	**1.0000**	**0.6936**	**0.9742**	**0.9481**	**0.9810**	**0.9999**
		RS	0.9000	0.6401	0.8138	0.6904	0.5183	0.0000
		$p\|\hat{A}_{12}$	0.02|0.55	0.00|0.85	0.00|1.00	0.00|1.00	0.00|1.00	0.00|1.00
	GD+	**OPAM**	**0.0000**	**0.0169**	**0.0031**	**0.0041**	**0.0062**	**0.0134**
		RS	0.0883	0.0394	0.0914	0.0794	0.1420	1.0000
		$p\|\hat{A}_{12}$	0.02|0.45	0.00|0.03	0.00|0.00	0.00|0.00	0.00|0.00	0.00|0.00
	Δ	**OPAM**	1.0000	0.7415	**0.4637**	**0.4077**	**0.4854**	0.9433
		RS	0.9766	**0.5251**	0.6358	0.7042	0.7535	1.0000
		$p\|\hat{A}_{12}$	0.16|0.52	0.00|0.80	0.00|0.20	0.00|0.03	0.00|0.09	0.08|0.47

n.nnnn: OPAM outperforms RS **n.nnnn**: RS outperforms OPAM



(a) ICS

(b) CCS

(c) UAV

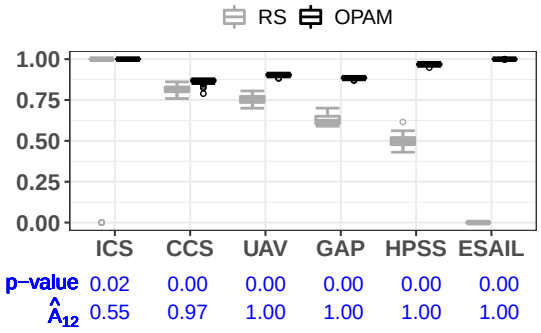
(d) GAP

(e) HPSS

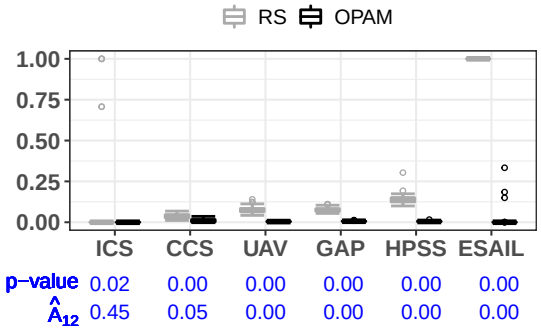
(f) ESAIL

Figure 5.9: Pareto fronts obtained by OPAM and RS for the six industrial subjects: (a) ICS, (b) CCS, (c) UAV, (d) GAP, (e) HPSS, and (f) ESAIL. The fitness values are computed based on each subject's set **E** of task-arrival sequences (see Section 5.7.6). The points located closer to the bottom left of each plot are considered to be better priority assignments when compared to points closer to the top right.

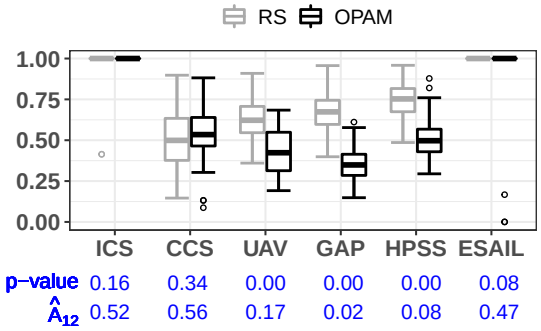
Δ based on 50 runs of OPAM and RS with the different test sets of task-arrival sequences. The results indicate that OPAM significantly outperforms RS in most comparison cases. Specifically, out of a total of 108 comparisons, OPAM outperforms RS 87 times (see the blue-colored cells related to OPAM in Table 5.3). Regarding Δ, RS outperforms OPAM for the CCS subject (see the gray-colored cells related to RS in Table 5.3). As shown in Table 5.2, CCS has only 3 aperiodic tasks and RS was therefore able to find better solutions with respect to Δ for such a simple subject.



(a) HV



(b) GD+



(c) Δ

Figure 5.10: Comparing OPAM and RS using the three quality indicators: (a) HV, (b) GD+, and (c) Δ. The boxplots (25%-50%-75%) show the quality values obtained from 50 runs of OPAM and RS. The quality values are computed based on the Pareto fronts obtained by the algorithms and each subject's set **E** of task-arrival sequences (see Section 5.7.6).

The answer to **RQ1** *is that* OPAM significantly outperforms RS with respect to HV and GD+. In particular, OPAM performs considerably better than RS when more aperiodic tasks are involved.

RQ2. To compare OPAM and SEQ, we first visually inspect the best Pareto fronts obtained from 50 runs of OPAM and SEQ for the six study systems described in Section 5.7.2 by varying the test sets of task-arrival sequences for each subject (see $\mathbf{T}_a^{10}$, $\mathbf{T}_w^{10}$, $\mathbf{T}_r^{10}$, $\mathbf{T}_a^{500}$, $\mathbf{T}_w^{500}$, and $\mathbf{T}_r^{500}$ described in Section 5.7.4), which are created independently from OPAM and SEQ. Overall, we observed that OPAM finds significantly better priority assignments in most cases. For example, Figure 5.11 depicts the best Pareto fronts obtained by OPAM and SEQ when the fitness values are computed based on each subject's test set $\mathbf{T}_a^{500}$ of 500

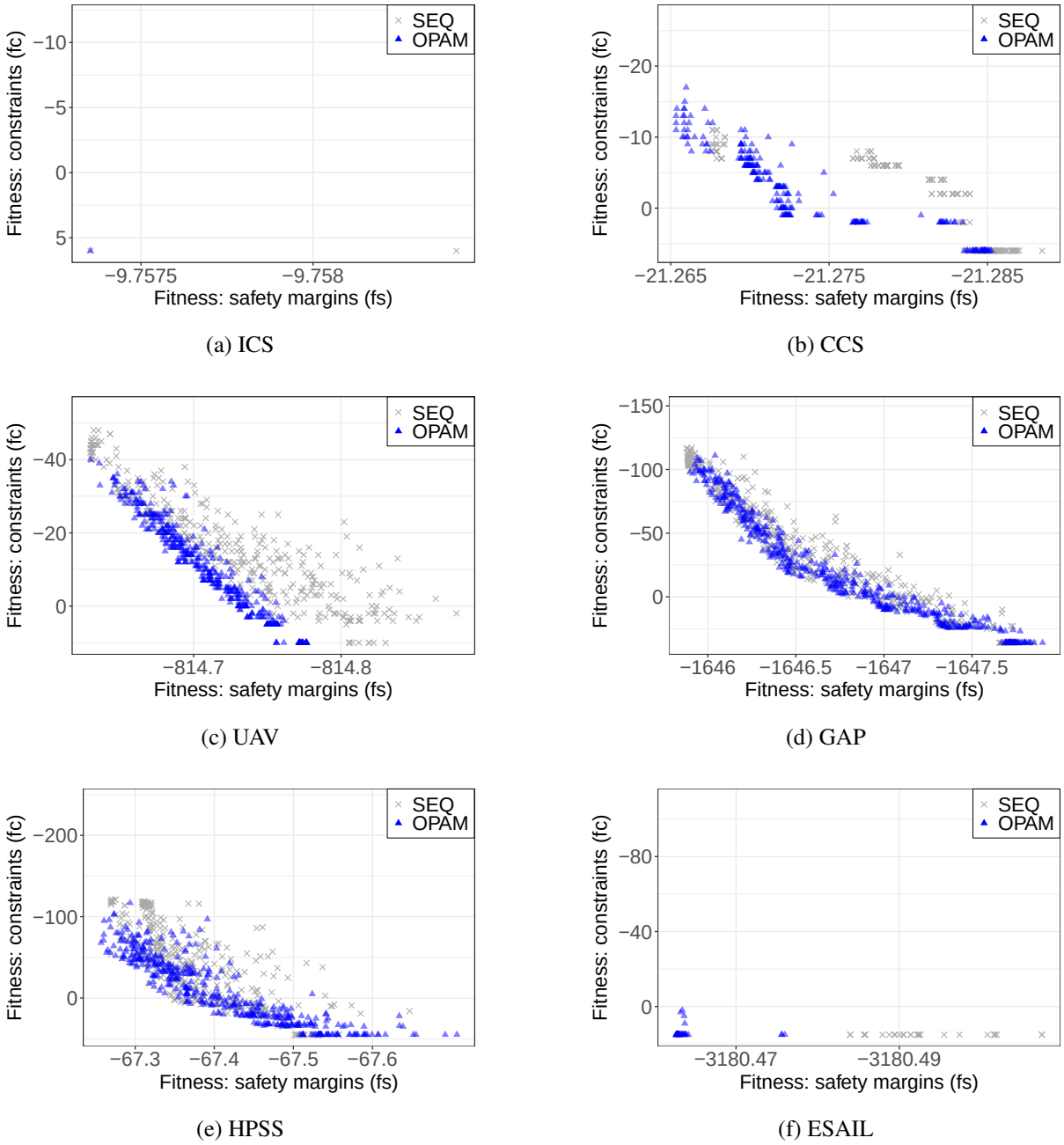


Figure 5.11: Pareto fronts obtained by OPAM and SEQ for the six industrial subjects: (a) ICS, (b) CCS, (c) UAV, (d) GAP, (e) HPSS, and (f) ESAIL. The fitness values are computed based on each subject's set $\mathbf{T}_a^{500}$ of task-arrival sequences (see Section 5.7.4). The points located closer to the bottom left of each plot are considered to be better priority assignments when compared to points closer to the top right.

task-arrival sequences, which are generated with adaptive random search. The results clearly show that OPAM outperforms SEQ with respect to producing more optimal Pareto fronts for ICS, CCS, UAV, HPSS, and ESAIL. For GAP, the visual inspection is not sufficient to provide any conclusions. Hence, we further compare OPAM and SEQ based on the quality indicators described in Section 5.7.5.

Table 5.4 compares the quality values measured by HV, GD+, and Δ for the six study subjects. To fairly compare the priority assignments obtained by OPAM and SEQ, we assess them with the test sets of task-arrival sequences for each subject (see $\mathbf{T}_a^{10}$, $\mathbf{T}_w^{10}$, $\mathbf{T}_r^{10}$, $\mathbf{T}_a^{500}$, $\mathbf{T}_w^{500}$, and $\mathbf{T}_r^{500}$ described in Section 5.7.4).

Table 5.4: Comparing OPAM and SEQ using the three quality indicators: HV, GD+, and Δ. Average quality values computed based on 50 runs of OPAM and SEQ using the different sets of task-arrival sequences (see Section 5.7.4).

			ICS	CCS	UAV	GAP	HPSS	ESAIL
$\mathbf{T}^{10}_{a}$ (adaptive, size 10)	HV	**OPAM**	0.0000	**0.6052**	**0.6011**	**0.6088**	**0.6290**	**0.9808**
		SEQ	0.0000	0.4172	0.5354	0.5868	0.6086	0.4470
		$p\|\hat{A}_{12}$	1.00|0.50	0.00|1.00	0.00|0.95	0.00|0.76	0.02|0.63	0.00|1.00
	GD+	**OPAM**	**0.0000**	**0.0244**	**0.0175**	**0.0148**	**0.0529**	**0.0249**
		SEQ	0.2191	0.0835	0.0350	0.0201	0.0625	0.1887
		$p\|\hat{A}_{12}$	0.00|0.01	0.00|0.00	0.00|0.01	0.00|0.25	0.00|0.26	0.00|0.03
	Δ	**OPAM**	1.0000	0.7653	0.4239	0.3343	0.5297	0.9444
		SEQ	**0.0200**	**0.5656**	**0.3628**	**0.2875**	0.5706	**0.8285**
		$p\|\hat{A}_{12}$	0.00|0.99	0.00|0.81	0.01|0.64	0.01|0.65	0.33|0.44	0.00|0.75
$\mathbf{T}^{10}_{w}$ (worst, size 10)	HV	**OPAM**	0.0000	**0.7345**	**0.6258**	**0.6290**	**0.7460**	**0.9059**
		SEQ	0.0000	0.6794	0.5933	0.5928	0.6856	0.5046
		$p\|\hat{A}_{12}$	1.00|0.50	0.00|0.82	0.00|0.82	0.00|0.88	0.00|0.87	0.00|1.00
	GD+	**OPAM**	0.0000	0.0912	**0.0191**	**0.0131**	**0.0340**	**0.0724**
		SEQ	0.0000	**0.0695**	0.0272	0.0211	0.0667	0.1720
		$p\|\hat{A}_{12}$	1.00|0.50	0.00|0.86	0.00|0.12	0.00|0.14	0.00|0.03	0.00|0.07
	Δ	**OPAM**	1.0000	0.7009	0.4835	0.3616	**0.4695**	0.9470
		SEQ	1.0000	**0.5376**	**0.3111**	**0.3054**	0.5453	**0.7547**
		$p\|\hat{A}_{12}$	1.00|0.50	0.00|0.74	0.00|0.83	0.01|0.66	0.01|0.35	0.00|0.67
$\mathbf{T}^{10}_{r}$ (random, size 10)	HV	**OPAM**	0.0000	**0.8720**	**0.8653**	**0.6340**	0.7714	**0.9055**
		SEQ	0.0000	0.5478	0.7246	0.5879	0.7935	0.1139
		$p\|\hat{A}_{12}$	1.00|0.50	0.00|0.99	0.00|1.00	0.00|0.92	0.06|0.39	0.00|1.00
	GD+	**OPAM**	0.0000	**0.0911**	**0.0205**	**0.0160**	**0.0472**	**0.0718**
		SEQ	0.0000	0.1358	0.0882	0.0277	0.0646	0.2838
		$p\|\hat{A}_{12}$	1.00|0.50	0.00|0.01	0.00|0.00	0.00|0.10	0.00|0.19	0.00|0.06
	Δ	**OPAM**	1.0000	0.8605	0.4644	0.3825	0.4658	0.9456
		SEQ	1.0000	**0.5896**	**0.4072**	**0.3253**	0.4620	**0.9670**
		$p\|\hat{A}_{12}$	1.00|0.50	0.00|0.82	0.02|0.64	0.01|0.66	0.90|0.49	0.00|0.67
$\mathbf{T}^{500}_{a}$ (adaptive, size 500)	HV	**OPAM**	0.0000	**0.6781**	**0.7134**	**0.6261**	**0.7332**	**0.9744**
		SEQ	0.0000	0.4854	0.6179	0.5981	0.7056	0.3571
		$p\|\hat{A}_{12}$	1.00|0.50	0.00|1.00	0.00|1.00	0.00|0.83	0.00|0.73	0.00|1.00
	GD+	**OPAM**	**0.0000**	**0.0174**	**0.0140**	**0.0134**	**0.0320**	**0.0285**
		SEQ	0.2191	0.0727	0.0549	0.0197	0.0565	0.2153
		$p\|\hat{A}_{12}$	0.00|0.01	0.00|0.00	0.00|0.00	0.00|0.20	0.00|0.08	0.00|0.04
	Δ	**OPAM**	1.0000	0.7833	0.4964	0.3588	**0.4564**	0.9442
		SEQ	**0.0200**	0.7319	**0.4002**	0.3315	0.5312	**0.8554**
		$p\|\hat{A}_{12}$	0.00|0.99	0.23|0.57	0.00|0.72	0.07|0.60	0.02|0.36	0.00|0.75
$\mathbf{T}^{500}_{w}$ (worst, size 500)	HV	**OPAM**	0.0000	0.4732	**0.6330**	**0.6181**	**0.6990**	**0.8755**
		SEQ	0.0000	**0.5564**	0.5958	0.5792	0.6800	0.1183
		$p\|\hat{A}_{12}$	1.00|0.50	0.00|0.04	0.00|0.85	0.00|0.90	0.00|0.70	0.00|1.00
	GD+	**OPAM**	**0.0000**	0.0511	**0.0141**	**0.0135**	**0.0258**	**0.0911**
		SEQ	0.2191	**0.0343**	0.0267	0.0226	0.0336	0.2849
		$p\|\hat{A}_{12}$	0.00|0.01	0.00|0.96	0.00|0.05	0.00|0.11	0.00|0.24	0.00|0.06
	Δ	**OPAM**	1.0000	0.7569	0.4950	0.3751	0.5379	0.9469
		SEQ	**0.0200**	0.7259	**0.3315**	**0.3139**	0.5102	**0.8957**
		$p\|\hat{A}_{12}$	0.00|0.99	0.43|0.55	0.00|0.82	0.01|0.66	0.20|0.57	0.00|0.67
$\mathbf{T}^{500}_{r}$ (random, size 500)	HV	**OPAM**	0.0000	**0.6646**	**0.8446**	**0.6321**	**0.7087**	**0.8782**
		SEQ	0.0000	0.4876	0.7242	0.5839	0.6786	0.1965
		$p\|\hat{A}_{12}$	1.00|0.50	0.00|1.00	0.00|1.00	0.00|0.93	0.00|0.72	0.00|1.00
	GD+	**OPAM**	**0.0000**	**0.0184**	**0.0172**	**0.0165**	**0.0327**	**0.0900**
		SEQ	0.2191	0.0684	0.0791	0.0285	0.0580	0.2620
		$p\|\hat{A}_{12}$	0.00|0.01	0.00|0.00	0.00|0.00	0.00|0.09	0.00|0.06	0.00|0.06
	Δ	**OPAM**	1.0000	0.7449	0.5059	0.3960	**0.4502**	0.9472
		SEQ	**0.0200**	0.6798	**0.4156**	**0.3341**	0.5148	**0.8546**
		$p\|\hat{A}_{12}$	0.00|0.99	0.19|0.58	0.00|0.71	0.01|0.66	0.03|0.38	0.00|0.67

n.nnnn : OPAM outperforms SEQ **n.nnnn** : SEQ outperforms OPAM

Table 5.5: Execution times and memory usage required to run OPAM for the six industrial subjects. Average values computed based on 50 runs of OPAM are reported.

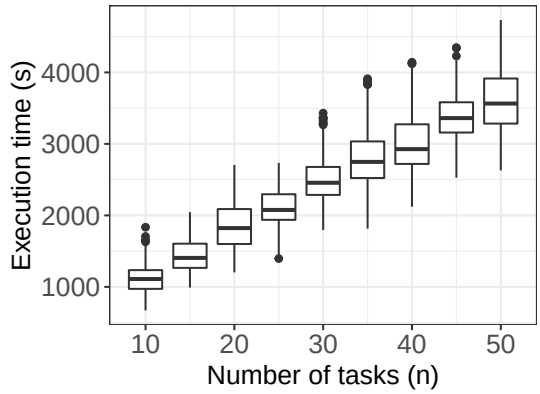
Subject	Execution time (s)	Memory usage (MB)
ICS	104.34	89.97
CCS	165.50	111.85
UAV	1455.35	312.85
GAP	2819.03	730.29
HPSS	226.98	127.77
ESAIL	55844.23	2879.79

Table 5.4 reports the average quality values computed based on 50 runs of OPAM and SEQ. In Table 5.4, the statistical comparison of the two corresponding distributions are reported using p-values and $\hat{A}_{12}$ values.

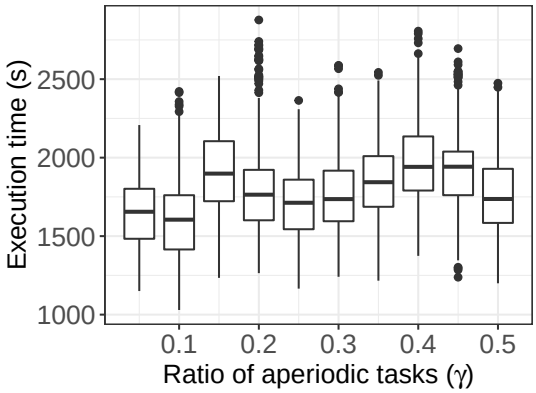
As shown in Table 5.4, we compared OPAM and SEQ 108 times by varying the study subjects, the quality indicators, the number of task-arrival sequences, and the task-arrival sequence generation methods. Out of 108 comparisons, OPAM significantly outperforms SEQ 63 times. Specifically, out of 36 HV comparisons, OPAM obtains better HV values than SEQ 28 times. For ICS (6 HV comparisons), the differences in HV values between OPAM and SEQ are not statistically significant. In only one HV comparison for CCS, SEQ outperforms OPAM (see the gray-colored cell related to HV and CCS in Table 5.4). To interpret these results, one must recall from Table 5.2 that ICS and CCS have only three aperiodic tasks that impact the degree of uncertainty in task arrivals and therefore represent simple cases. Out of 36 GD+ comparisons, OPAM outperforms SEQ 32 times. SEQ outperforms OPAM only two times for CCS. Hence, overall, the results indicate that OPAM outperforms SEQ, in terms of generating more optimal Pareto fronts, when the subjects feature a considerable degree of uncertainty in task arrivals and therefore make our search problem more complex. Otherwise differences are not statistically or practically significant. Regarding Δ, which focuses on the diversity of solutions on the Pareto front, SEQ outperforms OPAM 24 times out of 36 comparisons (see the gray-colored cells related to Δ in Table 5.4). However, since OPAM produces enough alternative priority assignments spreading across Pareto fronts (as visible from the solutions obtained by OPAM in Figure 5.11), these differences in Δ have limited implications in practice.

The answer to **RQ2** *is that* OPAM significantly outperforms SEQ with respect to HV and GD+ when in the presence of more than a few aperiodic tasks and therefore higher uncertainty in terms of task arrivals. OPAM therefore generate solutions on a Pareto front that is closer to the unknown, optimal one. In other words, coevolution is a suitable and successful strategy for finding better priority assignments in complex systems.

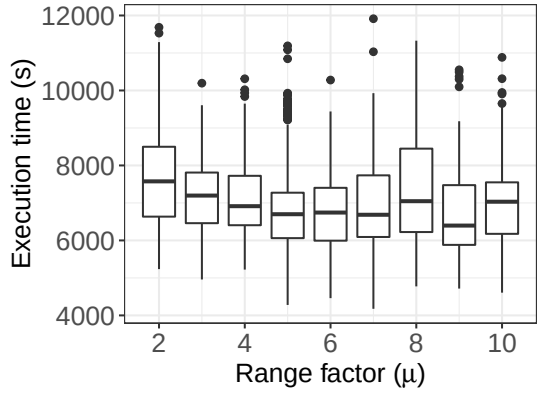
RQ3. Table 5.5 reports the average execution times and memory usage required to run OPAM for the six industrial subjects, over 50 runs. As shown in Table 5.5, finding optimal priority assignments for ESAIL requires the largest execution time (≈15.5h) and memory usage (≈2.9GB), compared to the other subjects. We note that such execution time and memory usage are acceptable as OPAM can be executed offline in practice.



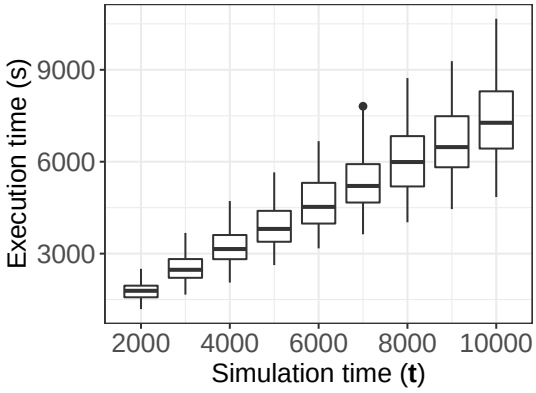
(a) Number of tasks (n)



(b) Ratio of aperiodic tasks(γ)



(c) Range factor (μ)

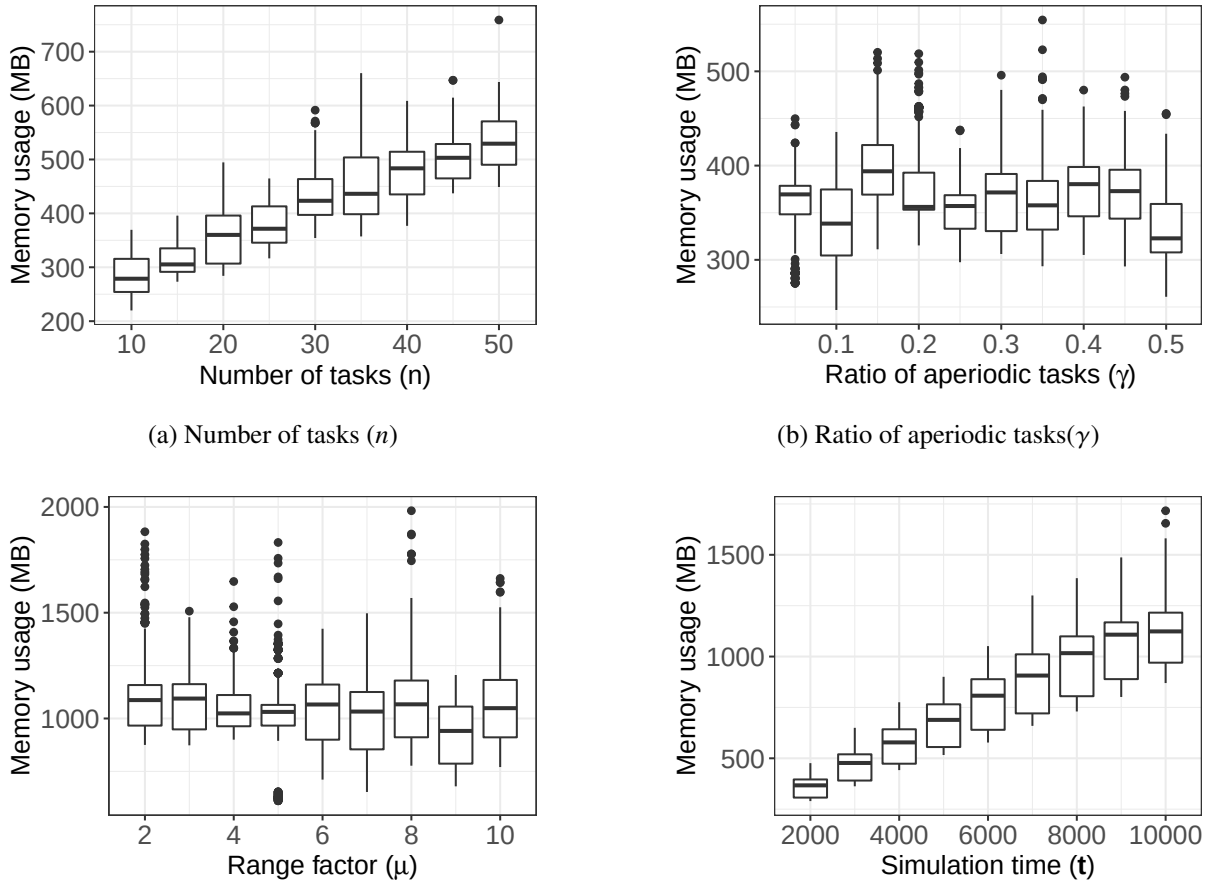


(d) Simulation time (**t**)

Figure 5.12: Execution times of OPAM when varying the values of the following parameters: (a) number of tasks n, (b) ratio of aperiodic tasks γ, (c) range factor μ, and (d) simulation time **t**. The boxplots (25%-50%-75%) show the execution times obtained from 500 runs of OPAM, i.e., 50 runs for each of the 10 synthetic subjects with the same configuration.

Figures 5.12 and 5.13 show, respectively, the execution times and memory usage from EXP3.1 (a), EXP3.2 (b), EXP3.3 (c), and EXP3.4 (d), described in Section 5.7.4. The boxplots in the figures show distributions (25%-50%-75%) obtained from 50 × 10 runs of OPAM for a set of 10 synthetic subjects, which are created with the same experimental setting. Regarding the execution time of OPAM, Figures 5.12a and 5.12d show that the execution time of OPAM is linear both in the number of tasks and simulation time. As for the memory usage of OPAM, results in Figures 5.13a and 5.13d indicate that memory usage is linear both in the number of tasks and in the simulation time. However, the results depicted in Figures 5.12b, 5.12c, 5.13b, and 5.13c indicate that there are no correlations between OPAM execution time and memory usage and the following two parameters: ratio of aperiodic tasks and range factor. Therefore, we expect OPAM to scale well as the numbers of tasks and simulation time increase.

The answer to **RQ3** *is that* the execution time and memory usage of OPAM are linear in the number of tasks and simulation time, thus scaling to industrial systems. Further, across our experiments, OPAM takes at most 15.5h using 2.9GB of memory to optimize priority assignments, an acceptable result since this is done offline.



(a) Number of tasks (n)

(b) Ratio of aperiodic tasks(γ)

(c) Range factor (μ)

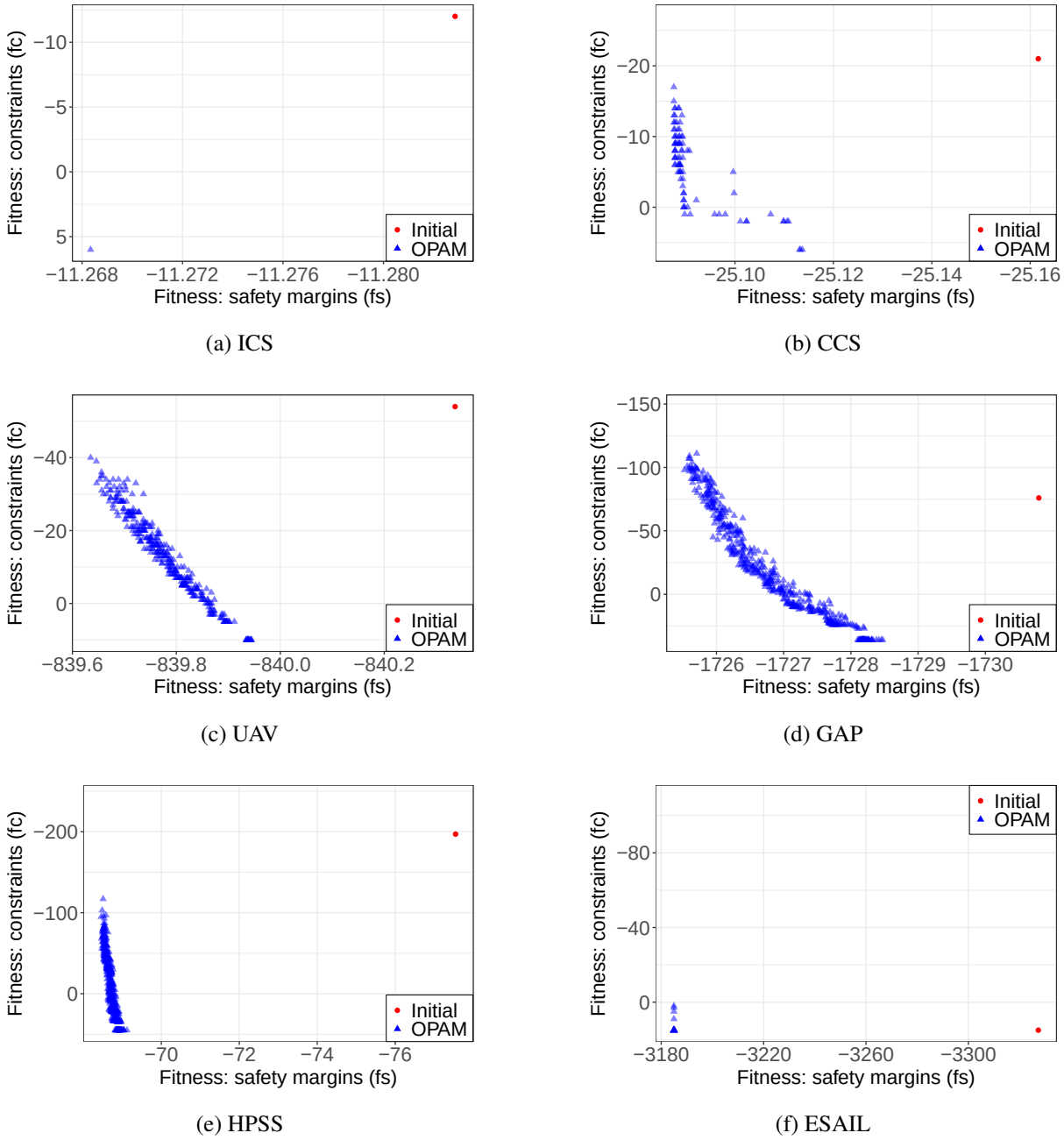
(d) Simulation time (**t**)

Figure 5.13: Memory usage of OPAM when varying the values of the following parameters: (a) number of tasks n, (b) ratio of aperiodic tasks γ, (c) range factor μ, and (d) simulation time **t**. The boxplots (25%-50%-75%) show the memory usage obtained from 500 runs of OPAM, i.e., 50 runs for each of the synthetic subjects with the same configuration.

RQ4. Figure 5.14 compares, with respect to external fitness (see the $fs()$ and $fc()$ fitness functions and the set **E** of sequences of task arrivals described in Section 5.6.6), the Pareto solutions obtained by OPAM against the priority assignments defined by engineers for the six industrial subjects: ICS (Figure 5.14a), CCS (Figure 5.14b), UAV (Figure 5.14c), GAP (Figure 5.14d), HPSS (Figure 5.14e), and ESAIL (Figure 5.14f).

As shown in the figure, the solutions obtained by OPAM clearly outperform the priority assignments defined by engineers regarding the two external objectives: the magnitude of safety margins and the extent to which constraints are satisfied.

Table 5.6 summarizes safety margins from the task executions of ESAIL when using one of our priority assignments optimized by OPAM and the one defined by engineers at LuxSpace. Note that we focus on ESAIL as it is not possible to access the engineers who developed the other five industrial subjects reported in the literature [120, 171, 145, 132, 11]. For comparison, we chose the bottom-left solution in Figure 5.14f since it is optimal for the constraint fitness, which is the same as the fitness value of the priority assignment defined by engineers, and the differences in safety margin fitness among our solutions are negligible.



(a) ICS

(b) CCS

(c) UAV

(d) GAP

(e) HPSS

(f) ESAIL

Figure 5.14: Comparing Pareto solutions obtained by OPAM and priority assignments defined by engineers for the six industrial subjects: (a) ICS, (b) CCS, (c) UAV, (d) GAP, (e) HPSS, and (f) ESAIL. The points located closer to the bottom left of each plot are considered to be better priority assignments when compared to points closer to the top right.

As shown in Table 5.6, our optimized priority assignment significantly outperforms the one of engineers. Our solution increases safety margins, on average, by 5.33% compared to the engineers' solution. For aperiodic tasks, our solution decreases safety margins by 0.01% (4.2ms difference) when the safety margins being compared are the maximum margins observed in both solutions (see the maximum safety margins, 59710.3ms obtained by engineers' solution and 59707.2ms obtained by OPAM, in Table 5.6). Such a small decrease is however negligible in the context of ESAIL as the maximum safety margin obtained by our solution is still large, i.e., ≈1m. For periodic tasks, we note that our solution

Table 5.6: Comparing safety margins from the task executions of ESAIL when using our optimized priority assignment and the one defined by engineers.

		Periodic tasks	Aperiodic tasks	All tasks
Engineer	Min	-44.5	9.4	-44.5
	Max	1879.7	59710.3	59710.3
	Avg.	126.6	52.6	78.1
	Median	82.1	9.4	48.1
OPAM	Min	48.1	9.4	9.4
	Max	1879.7	59707.2	59707.2
	Avg.	129.8	57.2	82.3
	Median	85.7	9.4	48.1
% Difference	Min	208.09%	0.00%	121.12%
	Max	0.00%	-0.01%	-0.01%
	Avg.	2.53%	8.89%	5.33%
	Median	4.38%	0.00%	0.00%

∗ Unit of time: ms

increases safety margins by 208.09% when the safety margins being compared are the minimum margins observed in both solutions (see the minimum safety margins, -44.5ms obtained by engineers' solution and 48.1ms obtained by OPAM, in Table 5.6). Note that the minimum safety margin of -44.5ms obtained with the engineers' solution indicates that a task violates its deadline. In the context of ESAIL, which is a mission-critical system, such gain in safety margins in the executions of periodic tasks is important because the hard deadlines of periodic tasks are more critical than the soft deadlines of aperiodic tasks.

Investigating practitioners' perceptions of the benefits of OPAM is necessary to adopt OPAM in practice. To do so, we draw on the qualitative reflections of three software engineers at LuxSpace, with whom we have been collaborating on this research. They have had four to seven years of experience developing satellite systems at LuxSpace, with more than 50 years of collective experience in companies. All the reflections are based on observations made throughout our interactions. The engineers at LuxSpace deemed OPAM to be an improvement over their current practice as it allows them to perform domain-specific trade-off analysis among Pareto solutions and is useful in practice to support decision making with respect to their task design. Encouraged by the promising results, we are now applying OPAM to new systems in collaboration with LuxSpace.

The answer to **RQ4** *is that* OPAM helps optimize priority assignments such that they outperform those manually defined by engineers based on domain expertise. Our results show that OPAM, compared to current practice, increases safety margins, on average, by 5.33%.

5.7.8 Threats to validity

To mitigate the main threats that arise from not accounting for random variation, we compared OPAM against RS under identical parameter settings. We present all the underlying parameters and provide the full package of our experiments to facilitate replication. Also, we ran OPAM 50 times for each

study subject and compared results using statistical analysis, i.e., Mann-Whitney U-test and Vargha and Delaney's $\hat{A}_{12}$.

We note that there are prior studies that aim at optimizing priority assignments such as OPA [19] and RPA [56]. However, to our knowledge, none of the existing works offer ways to analyze trade-offs among equally viable priority assignments with respect to safety margins and the satisfaction of constraints. Nevertheless, we attempted to compare OPAM with an extension of an existing method, e.g., RPA [56]. To do so, we first applied an exhaustive schedulability analysis technique to the ESAIL subject – our motivating case study – in order to verify whether the ESAIL tasks are schedulable for a given priority assignment. Note that existing priority assignment techniques are built on such schedulability analysis methods, which are therefore a prerequisite. We chose UPPAAL [25], a model checker, for schedulability analysis as it has been used in real-time system studies [132, 192, 190]. However, our experiment results using UPPAAL for ESAIL showed that it was not able to complete the analysis task, even after 5 days of execution, for a single priority assignment. We were therefore not able to perform experimental comparisons with existing priority assignment methods. Since this evaluation is not the main focus of this chapter, we point the reader to the UPPAAL specification of ESAIL available online [109].

Recall from Section 5.6.2 that OPAM assigns tasks' WCETs to their execution times when it simulates the worst-case executions of tasks while varying task arrival times. In many real-time systems studies [36, 83, 117, 11, 194, 64, 68], static WCETs are often used instead of varying task execution times for the purpose of real-time analysis. For example, practitioners typically use WCETs to estimate the lowest bound of CPU utilization required to properly apply the rate monotonic scheduling policy [76] to their systems. Similarly, OPAM assumes that near-worst-case schedule scenarios can be simulated by assigning tasks' WCETs to their execution times and varying tasks' arrival times using search. A near-worst-case schedule scenario entails that the magnitude of deadline misses is maximized when tasks execute as per this scenario. Under this working assumption, we were able to empirically evaluate the sanity, coevolution, scalability, and usefulness aspects of OPAM (see Section 5.7). The results indicate that OPAM is a promising and useful tool. However, the formal proof of whether or not the WCET assumption holds in the system model described in Section 5.3 requires complex analysis, accounting for varying task arrival times, triggering relationships, resource dependencies, and multiple cores. When task execution times need to be varied during simulation, engineers can adapt OPAM by utilizing Monte-Carlo simulation [105] to account for such variations.

The main threat to external validity is that our results may not generalize to other systems. We mitigate potential biases and errors in our experiments by drawing on real industrial subjects from different domains and several synthetic subjects. Specifically, we selected two subjects from the aerospace domain, two from the automotive domain, and two from the avionics domain. The positive feedback obtained from LuxSpace and the encouraging results from our industrial case studies indicate that OPAM is a scalable and practical solution. Furthermore, we believe OPAM introduces a promising avenue for addressing the problem of priority assignment by applying coevolutionary algorithms, even for systems that use other scheduling policies, e.g., priority inheritance. In order for OPAM to support different scheduling policies, the main requirement is to replace the existing simulator (described in Section 5.6) with a new simulator supporting the desired scheduling policy. In our approach, the coevolution part of OPAM is separated from the scheduling policy, which is contained in the simulator. Hence, we deem the expected changes for the coevolution part of OPAM to be minimal. Future studies are nevertheless necessary to investigate how

OPAM can be adapted to find near-optimal priority assignments for other real-time systems in different contexts.

5.8 Conclusion

We developed OPAM, a priority assignment method for real-time systems, that aims to find equally viable priority assignments that maximize the magnitude of safety margins and the extent to which engineering constraints are satisfied. OPAM uses a novel approach, based on multi-objective, competitive coevolutionary search, that simultaneously evolves different species, i.e., populations of priority assignments and stress test scenarios, that compete with one another with opposite objectives, the former trying to minimize chances of deadline misses while the latter attempts to maximize them. We evaluated OPAM on a number of synthetic systems as well as six industrial systems from different domains. The results indicate that OPAM is able to find significantly better solutions than both those manually defined by engineers based on expert knowledge and those obtained by our baselines: random search and sequential search. Further, OPAM scales linearly with the number of tasks in a system and the time required to simulate task executions. Execution times on our industrial systems are practically acceptable.

In the future, we will continue to study the problem of optimal priority assignment by accounting for (1) priority assignments that change dynamically, (2) WCET value ranges that account for non-deterministic computation times, (3) interrupt handling routines that execute differently compared to real-time tasks, and (4) hybrid scheduling policies that combine multiple standard policies. We also plan to develop a real-time task modeling language to specify task characteristics such as resource dependencies, triggering relationships, engineering constraints, and behaviors of real-time tasks and to facilitate real-time system analysis, e.g., optimal priority assignment and schedulability analysis. In addition, we would like to incorporate additional analysis capabilities into OPAM in order to verify whether or not a system satisfies the required properties, e.g., schedulability of tasks and absence of deadlocks, for a given priority assignment. For example, statistical model checking [114] may allow us to verify whether tasks meet their deadlines for a given priority assignment with a probabilistic guarantee. In the long term, we plan to more conclusively validate the usefulness of OPAM by applying it to additional case studies in different application domains.

Chapter 6

Conclusion

6.1 Summary

In this dissertation, we addressed the problems that arise in estimating safe WCET ranges and in finding optimal priority assignments. To address these problems, we proposed three approaches.

With regard to estimating safe WCET ranges, we proposed SAFE aiming at precisely estimating safe WCET ranges within which real-time tasks likely meet their deadlines with a high-level of confidence. SAFE consists of two phases, i.e., searching worst-case task arrivals and learning safe WCET ranges, which are applicable at early design stages. A meta-heuristic search algorithm is employed to generate worst-case sequences of task arrivals that maximize the magnitude of deadline misses. The search results are applied to learn a logistic regression model, which SAFE uses to determine safe WCET ranges with a probability that tasks will likely meet their deadlines. Furthermore, as modern software systems have become increasingly complicated over time, SAFE is designed to be scalable by leveraging scalable techniques. In phase 1, SAFE relies on a genetic algorithm and simulation. To implement phase 2, SAFE employs feature reduction, an effective sampling strategy, and polynomial logistic regression. We evaluated SAFE on a real-time satellite system in collaboration with an industrial partner (LuxSpace) and two other industrial systems in other domains. The experimental results show that SAFE is a successful method for precisely estimating safe WCET ranges. The ranges computed by SAFE are more conservative than the initial ranges estimated by practitioners. We generated 800 synthetic systems to evaluate the scalability of SAFE. The results indicate that SAFE scales to complex systems. Overall, the execution time of SAFE was at most 27h which is acceptable as an offline analysis method in practice.

Although SAFE can analyze hard real-time systems, it is also desirable to analyze weakly hard real-time systems to assess their quality of service. Toward this end, we proposed SWEAK by extending SAFE to analyze weakly hard real-time systems. SWEAK uses a multi-objective genetic algorithm to search for the worst test cases that maximize the magnitude of deadline misses and the extent of consecutive deadline misses. Once the results are obtained from the search, SWEAK infers safe WCET ranges by leveraging a logistic regression model and selects a probability that minimizes the violations of deadline constraints and that maximizes the WCET ranges. For more accurate WCET estimation in a wide

range of industrial software, SWEAK accounts for the context switching times and leverages an industry scheduler, i.e., APS, provided by our industrial partner, Blackberry. We evaluated SWEAK on a real-time satellite system and on synthetic systems following the guidelines defined by our industrial partner. The results indicate that SWEAK infers satisfying WCET ranges for weakly hard real-time systems with high flexibility in selecting ranges by practitioners. In addition, we evaluated SWEAK by applying it to 600 synthetic systems with various degrees of complexity. The results indicate that SWEAK is scalable. Overall, SWEAK completed all the experiments at most 22.1h. Practitioners confirmed that SWEAK is acceptable for practical use as an offline analysis technique.

For finding the (near-)optimal priority assignments for real-time systems, we proposed OPAM. OPAM aims at finding equally viable priority assignments that maximize the magnitude of safety margins and the extent to which engineering constraints are satisfied. OPAM leverages a multi-objective, competitive coevolutionary search algorithm that simultaneously evolves different species, i.e., populations of priority assignments and stress test scenarios. In the coevolution algorithm, two populations compete with each other to evolve themselves simultaneously. The former population aims to minimize chances of deadline misses while the latter attempts to maximize them. We evaluated OPAM on six industrial systems from different domains and 370 synthetic systems. The results show that OPAM finds notably better priority assignments than both those manually defined by engineers based on expert knowledge and those obtained by our baselines: random search and sequential search. Moreover, OPAM linearly scales with the number of tasks in a system and the time required to simulate task executions. Based on the discussions with practitioners, the execution times of OPAM on our industrial systems are practically acceptable.

6.2 Future work

This section describes future research directions that are related to this dissertation.

Generalization. Using simulation-based approaches, we were able to analyze realistic real-time systems when estimating safe WCET ranges and predicting (near-)optimal priority assignments. However, our approaches for the WCET estimation are not applicable when they are applied to systems with the following characteristics: (1) dynamic priority assignments and (2) hybrid scheduling policies. Additionally, OPAM can be generally applicable to a wide range of real-time systems by considering weakly hard constraints and by applying a proper and complex scheduler such as APS.

More solid validation of our work. We evaluated our approaches with a limited number of industrial systems and a large number of synthetic systems to mitigate any potential biases in our experimental results. Although we try our best to avoid potential biases in the results, better strategies may exist to evaluate our approaches. In the long term, therefore, we plan to apply our approaches to many industrial systems in different application domains.

Real-time task modeling language. Designing a modeling language for real-time systems is a potential direction for future work. A language that is capable of describing various features, such as dependencies, constraints, and behaviors of real-time systems, may facilitate schedulability analysis. We believe that if practitioners describe the system and their needs with a language specifically designed for schedulability analysis, the analysis approaches can be more precise in inferring safe WCET ranges and in finding (near-) optimal priority assignments.

Bibliography

[1] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. Automated repair of feature interaction failures in automated driving systems. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'20)*, pages 88–100, 2020.

[2] Jaume Abella, Maria Padilla, Joan Del Castillo, and Francisco J. Cazorla. Measurement-based worst-case execution time estimation using the coefficient of variation. *ACM Transactions Design Automation of Electronic Systems*, 22(4), Jun 2017.

[3] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I. Davis. An Empirical Survey-based Study into Industry Practice in Real-time Systems. In *Proceedings of the 2020 IEEE Real-Time Systems Symposium (RTSS'20)*, volume 2020-Decem, pages 3–11, Dec 2020.

[4] T.A. AlEnawy and H. Aydin. Energy-constrained scheduling for weakly-hard real-time systems. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS'05)*, pages 376–385, 2005.

[5] Stefano Di Alesio, Arnaud Gotlieb, Shiva Nejati, and Lionel C. Briand. Testing deadline misses for real-time systems using constraint optimization techniques. In *Proceedings of the 5th IEEE International Conference on Software Testing, Verification and Validation (ICST'12)*, pages 764–769, Montreal, QC, Canada, 2012. IEEE.

[6] Stefano Di Alesio, Shiva Nejati, Lionel C. Briand, and Arnaud Gotlieb. Stress testing of task deadlines: A constraint programming approach. In *Proceedings of the 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE'13)*, pages 158–167, Pasadena, CA, USA, 2013. IEEE.

[7] Peter Altenbernd, Jan Gustafsson, Björn Lisper, and Friedhelm Stappert. Early execution time-estimation through automatically generated timing models. *Real-Time Systems*, 52(6):731–760, 2016.

[8] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. Software engineering for machine learning: A case study. In *Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300, 2019.

[9] Sandro S. Andrade and Raimundo José de A. Macêdo. A search-based approach for architectural design of feedback control concerns in self-adaptive systems. In *Proceedings of the 2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems (SASO'13)*, pages 61–70, 2013.

[10] Étienne André, Jawher Jerray, and Sahar Mhiri. Time4sys2imi: A tool to formalize real-time system models under uncertainty. In *Proceedings of the 16th International Colloquium on Theoretical Aspects of Computing (ICTAC'19)*, pages 113–123, Cham, 2019. Springer.

[11] Saoussen Anssi, Sara Tucci Piergiovanni, Stefan Kuntz, Sébastien Gérard, and François Terrier. Enabling scheduling analysis for AUTOSAR systems. In *Proceedings of the 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC'11)*, pages 152–159, Newport Beach, CA, USA, 2011. IEEE.

[12] Andrea Arcuri and Lionel C. Briand. A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability*, 24(3):219–250, 2014.

[13] Andrea Arcuri and Gordon Fraser. On parameter tuning in search based software engineering. In *Proceedings of the 3th International Symposium on Search Based Software Engineering (SSBSE'11)*, pages 33–47, 2011.

[14] Andrea Arcuri, Muhammad Zohaib Iqbal, and Lionel C. Briand. Black-box system testing of real-time embedded systems using random and search-based testing. In *Proceedings of the IFIP International Conference on Testing Software and Systems (ICTSS'10)*, volume 6435, pages 95–110, 2010.

[15] Andrea Arcuri and Xin Yao. A novel co-evolutionary approach to automatic software bug fixing. In *Proceedings of the 2008 IEEE Congress on Evolutionary Computation (CEC'08)*, pages 162–168, 2008.

[16] Andrea Arcuri and Xin Yao. Co-evolutionary automatic programming for software development. *Information Sciences*, 259:412–432, 2014.

[17] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, 1.00 edition, 2018.

[18] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A.J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284, 1993.

[19] Neil C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report, Dept. Computer Science, University of York, 1991.

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

[21] Jakob Axelsson. A method for evaluating uncertainties in the early development phases of embedded real-time systems. In *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*, pages 72–75, Hong Kong, China, 2005. IEEE.

[22] Sanjoy Baruah, Deji Chen, Sergey Gorinsky, and Aloysius Mok. Generalized multiframe tasks. *Real-Time Systems*, 17(1):5–22, 1999.

[23] S.K. Baruah, A. Burns, and R.I. Davis. Response-Time Analysis for Mixed Criticality Systems. In *Proceedings of the IEEE 32nd Real-Time Systems Symposium (RTSS'11)*, pages 34–43, Vienna, Austria, Nov 2011. IEEE.

[24] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explorations*, 6(1):20–29, 2004.

[25] Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on UPPAAL. In *Formal Methods for the Design of Real-Time Systems: International School on Formal Methods for the Design of Computer, Communication, and Software Systems*, pages 200–236, 2004.

[26] Kostiantyn Berezovskyi, Luca Santinelli, Konstantinos Bletsas, and Eduardo Tovar. Wcet measurement-based and extreme value theory characterisation of cuda kernels. In *Proceedings of the 22nd International Conference on Real-Time Networks and Systems (RTNS'14)*, page 279–288, New York, NY, USA, 2014. ACM.

[27] Guillem Bernat, Alan Burns, and Albert Llamosí. Weakly hard real-time systems. *IEEE Transactions on Computers*, 50(4):308–321, 2001.

[28] Guillem Bernat and Ricardo Cayssials. Guaranteed on-line weakly-hard real-time systems. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01)*, pages 25–35, 2001.

[29] Guillem Bernat, Antoine Colin, and Stefan M. Petters. WCET analysis of probabilistic hard real-time system. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, pages 279–288, Austin, TX, USA, 2002. IEEE.

[30] D. Berndt, J. Fisher, L. Johnson, J. Pinglikar, and A. Watkins. Breeding software test cases with genetic algorithms. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, pages 1–10, 2003.

[31] Enrico Bini and Giorgio C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, 2004.

[32] Enrico Bini, Marco Di Natale, and Giorgio Buttazzo. Sensitivity analysis for fixed-priority real-time systems. *Real-Time Systems*, 39(1):5–30, 2008.

[33] Armelle Bonenfant, Denis Claraz, Marianne De Michiel, and Pascal Sotin. Early WCET prediction using machine learning. In *Proceedings of the 17th International Workshop on Worst-Case Execution Time Analysis (WCET'17)*, volume 57, pages 1–9, Dagstuhl, Germany, 2017. Schloss Dagstuhl.

[34] Mohamed Boussaa, Wael Kessentini, Marouane Kessentini, Slim Bechikh, and Soukeina Ben Chikha. Competitive coevolutionary code-smells detection. In *Proceedings of the 5th International Symposium on Search Based Software Engineering (SSBSE'13)*, pages 50–65, 2013.

[35] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[36] Lionel C. Briand, Yvan Labiche, and Marwa Shousha. Stress testing real-time systems with genetic algorithms. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO'05)*, pages 1021–1028, New York, NY, USA, 2005. ACM.

[37] Lionel C Briand, William M Thomas, and Christopher J Hetmanski. Modeling and managing risk early in software development. In *Proceedings of the 15th International Conference on Software Engineering (ICSE'93)*, pages 55–65. IEEE, 1993.

[38] A. Burns and S. Edgar. Predicting computation time for advanced processor architectures. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems (ECRTS'00)*, pages 89–96, 2000.

[39] Alan Burns and Andrew J. Wellings. *Real-Time Systems and Programming Languages - Ada, Real-Time Java and C / Real-Time POSIX*. Addison-Wesley, 4th edition, 2009.

[40] G.C. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo. *Soft Real-Time Systems: Predictability vs. Efficiency*. Springer, 2006.

[41] Roberto Cavada, Alessandro Cimatti, Anders Franzén, Krishnamani Kalyanasundaram, Marco Roveri, and R. K. Shyamasundar. Computing predicate abstractions by integrating bdds and SMT solvers. In *Proceedings of the 7th International Conference on Formal Methods in Computer-Aided Design (FMCAD'07)*, pages 69–76, Austin, TX, USA, 2007. IEEE.

[42] Francisco J. Cazorla, Leonidas Kosmidis, Enrico Mezzetti, Carles Hernandez, Jaume Abella, and Tullio Vardanega. Probabilistic Worst-Case Timing Analysis: Taxonomy and Comprehensive Survey. *ACM Computing Surveys*, 52(1):1–35, Jan 2020.

[43] Jian Jia Chen, Georg Von Der Brüggen, and Niklas Ueter. Push forward: Global fixed-priority scheduling of arbitrary-deadline sporadic task systems. *Leibniz International Proceedings in Informatics, LIPIcs*, 106:1–24, 2018.

[44] Tao Chen, Ke Li, Rami Bahsoon, and Xin Yao. FEMOSAA: Feature-guided and knee-driven multi-objective optimization for self-adaptive software. *ACM Transactions on Software Engineering and Methodology*, 27(2):1–50, 2018.

[45] Tsong Yueh Chen, Fei Ching Kuo, Robert G. Merkel, and T. H. Tse. Adaptive Random Testing: The ART of test case diversity. *Journal of Systems and Software*, 83(1):60–66, 2010.

[46] Yanching Chu and Alan Burns. Flexible hard real-time scheduling for deliberative AI systems. *Real-Time Systems*, 40(3):241–263, 2008.

[47] Alessandro Cimatti, Luigi Palopoli, and Yusi Ramadian. Symbolic computation of schedulability regions using parametric timed automata. In *Proceedings of the 29th IEEE Real-Time Systems Symposium (RTSS'08)*, pages 80–89, Barcelona, Spain, 2008. IEEE.

[48] Edmund M. Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani. *Tools for Practical Software Verification*, volume 7682, chapter Model Checking and the State Explosion Problem, pages 1–30. Springer, 2012.

[49] David W. Corne, Nick R. Jerram, Joshua D. Knowles, and Martin J. Oates. Pesa-ii: Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation (GECCO'01)*, pages 283–290, San Francisco, CA, USA, 2001.

[50] Francis Cottet, Joëlle Delacroix, Claude Kaiser, and Zoubir Mammeri. *Scheduling in Real-Time Systems*. Wiley, 2002.

[51] Liliana Cucu-Grosjean, Luca Santinelli, Michael Houston, Code Lo, Tullio Vardanega, Leonidas Kosmidis, Jaume Abella, Enrico Mezzetti, Eduardo Quiñones, and Francisco J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS'12)*, pages 91–101, 2012.

[52] Dakshina Dasari, Matthias Becker, Daniel Casini, and Tobias Blas. End-to-End Analysis of Event Chains under the QNX Adaptive Partitioning Scheduler. In *Proceedings of the IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS'22)*, pages 214–227. IEEE, May 2022.

[53] Dakshina Dasari, Arne Hamann, Holger Broede, Michael Pressler, and Dirk Ziegenbein. Brief Industry Paper: Dissecting the QNX Adaptive Partitioning Scheduler. In *Proceedings of the IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS'21)*, pages 477–480. IEEE, May 2021.

[54] Robert Davis and Liliana Cucu-Grosjean. A Survey of Probabilistic Timing Analysis Techniques for Real-Time Systems. *LITES: Leibniz Transactions on Embedded Systems*, pages 1–60, 2019.

[55] Robert I. Davis and Marko Bertogna. Optimal fixed priority scheduling with deferred pre-emption. In *Proceedings of the 2012 IEEE 33rd Real-Time Systems Symposium (RTSS'12)*, pages 39–50, 2012.

[56] Robert I. Davis and Alan Burns. Robust priority assignment for fixed priority real-time systems. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS'07)*, pages 3–14, 2007.

[57] Robert I Davis and Alan Burns. A Survey of Hard Real-Time Scheduling Algorithms and Schedulability Analysis Techniques for Multiprocessor Systems. *Systems Research*, 43:1–44, 2009.

[58] Robert I. Davis and Alan Burns. Robust priority assignment for messages on Controller Area Network (CAN). *Real-Time Systems*, 41(2):152–180, 2009.

[59] Robert I. Davis and Alan Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems*, 47(1):1–40, Jan 2011.

[60] Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys (CSUR)*, 43(4), Oct 2011.

[61] Robert I. Davis, Liliana Cucu-Grosjean, Marko Bertogna, and Alan Burns. A review of priority assignment in real-time systems. *Journal of Systems Architecture*, 65:64–82, 2016.

[62] Robert I. Davis, Attila Zabos, and Alan Burns. Efficient exact schedulability tests for fixed priority real-time systems. *IEEE Transactions on Computers*, 57(9):1261–1276, 2008.

[63] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[64] Stefano Di Alesio, Lionel C. Briand, Shiva Nejati, and Arnaud Gotlieb. Combining genetic algorithms and constraint programming to support stress testing of task deadlines. *ACM Transactions on Software Engineering and Methodology*, 25(1):1–37, 2015.

[65] Stefano Di Alesio and Sagar Sen. Using UML/MARTE to support performance tuning and stress testing in real-time systems. *Software and Systems Modeling*, 17(2):479–508, 2018.

[66] D. Doval, S. Mancoridis, and B.S. Mitchell. Automatic clustering of software systems using a genetic algorithm. In *Proceedings of the 9th International Workshop Software Technology and Engineering Practice (STEP'99)*, pages 73–81, 1999.

[67] Juan J. Durillo and Antonio J. Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771, 2011.

[68] Marco Dürr, Georg Von Der Brüggen, Kuan Hsun Chen, and Jian-Jia Chen. End-to-End Timing Analysis of Sporadic Cause-Effect Chains in Distributed Systems. *ACM Transactions on Embedded Computing Systems*, 18(5s):1–24, Oct 2019.

[69] Jens Eickhoff. *Onboard Computers, Onboard Software and Satellite Operations: An Introduction*. Springer, 2011.

[70] Paul Emberson, Roger Stafford, and Robert I. Davis. Techniques for the synthesis of multiprocessor tasksets. In *Proceedings of the 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS'10)*, pages 6–11, 2010.

[71] M.S. Feather and T. Menzies. Converging on the optimal attainment of requirements. In *Proceedings of the IEEE Joint International Conference on Requirements Engineering (ICRE'02)*, pages 263–270, 2002.

[72] Timo Feld, Alessandro Biondi, Robert I. Davis, Giorgio Buttazzo, and Frank Slomka. A survey of schedulability analysis techniques for rate-dependent tasks. *Journal of Systems and Software*, 138:100–107, 2018.

[73] Christian Ferdinand and Reinhard Wilhelm. On predicting data cache behavior for real-time systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1474:16–30, 1998.

[74] Filomena Ferrucci, Mark Harman, Jian Ren, and Federica Sarro. Not going to take this anymore: Multi-objective overtime planning for software engineering projects. In *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*, pages 462–471, San Francisco, CA, USA, 2013. IEEE.

[75] Sevan Gregory Ficici. *Solution Concepts in Coevolutionary Algorithms*. Ph.d. thesis, Brandeis University, Department of Computer Science, Waltham, MA, 2004.

[76] Mark S. Fineberg and Omri Serlin. Multiprogramming for hybrid computation. In *Proceedings of the AFIPS Fall Joint Computing Conference (AFIPS'67)*, pages 1–13, 1967.

[77] Michel Gendreau and Jean-Yves Potvin. *Handbook of Metaheuristics*. Springer, 2nd edition, 2010.

[78] Laurent George, Nicolas Rivierre, and Marco Spuri. Preemptive and non-preemptive real-time uniprocessor scheduling. Research Report RR-2966, INRIA, 1996. Projet REFLECS.

[79] Oliver Gettings, Sophie Quinton, and Robert I. Davis. Mixed criticality systems with weakly-hard constraints. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems (RTNS'15)*, page 237–246. ACM, 2015.

[80] David E. Goldberg and Robert Lingle. Alleleslociand the traveling salesman problem. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 154–159, 1985.

[81] Sunlu Gong and Jian Jun Han. Global emergency-based job-level scheduling for weakly-hard real-time systems. *Journal of Systems Architecture*, 117(March):102150, 2021.

[82] Werner Grass and Thi Huyen Chau Nguyen. Improved response-time bounds in fixed priority scheduling with arbitrary deadlines. *Real-Time Systems*, 54(1):1–30, 2018.

[83] Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. New response time bounds for fixed priority multiprocessor scheduling. In *Proceedings of the 2009 30th IEEE International Real-Time Systems Symposium (RTSS'09)*, pages 387–397, 2009.

[84] Jan Gustafsson, Peter Altenbernd, Andreas Ermedahl, and Björn Lisper. Approximate worst-case execution time analysis for early stage embedded systems development. In *Proceedings of the 7th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS'09)*, pages 308–319, Berlin, Heidelberg, 2009. Springer.

[85] Jeffery P. Hansen, Scott A. Hissam, and Gabriel A. Moreno. Statistical-based WCET estimation and validation. In *Proceedings of the 9th International Workshop on Worst-Case Execution Time Analysis (WCET'09)*, pages 1–11, Dagstuhl, Germany, 2009. Schloss Dagstuhl.

[86] Damien Hardy and Isabelle Puaut. WCET analysis of instruction cache hierarchies. *Journal of Systems Architecture*, 57(7):677–694, Aug 2011.

[87] Damien Hardy, Isabelle Puaut, and Yiannakis Sazeides. Probabilistic WCET estimation in presence of hardware for mitigating the impact of permanent faults. In *Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE'16)*, pages 91–96, Dresden, Germany, 2016. IEEE.

[88] Mark Harman, Robert Hierons, and Mark Proctor. A new representation and crossover operator for search-based optimization of software modularization. In *Proceedings of the 5th Annual Conference on Genetic and Evolutionary Computation (GECCO'02)*, pages 1351—1358. ACM, 2002.

[89] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Computing Survey*, 45(1):1–61, 2012.

[90] Mark Harman, Stephen Swift, and Kiarash Mahdavi. An empirical study of the robustness of two module clustering fitness functions. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO'05)*, pages 1029–1036, New York, NY, USA, 2005. ACM.

[91] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2009.

[92] Leo Hatvani, Sara Afshar, and Reinder J. Bril. Optimal priority and threshold assignment for fixed-priority preemption threshold scheduling. *ACM SIGBED Review*, 15(1):43–49, 2018.

[93] Randy L. Haupt and Sue Ellen Haupt. *Practical Genetic Algorithms*. John Wiley & Sons, Inc., 1998.

[94] Randy L Haupt, Sue Ellen Haupt, and A John Wiley. *Practical Genetic Algorithms*. Wiley, 2nd edition, 2004.

[95] Hadi Hemmati, Andrea Arcuri, and Lionel C. Briand. Achieving scalable model-based testing through test case diversity. *ACM Transactions on Software Engineering and Methodology*, 22(1):1–42, 2013.

[96] Kawakubo Hideko and Yoshida Hiroaki. Rapid feature selection based on random forests for high-dimensional data. *Expert Systems with Applications*, 40(1):6241–6252, 2012.

[97] John Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[98] Hisao Ishibuchi, Hiroyuki Masuda, Yuki Tanigaki, and Yusuke Nojima. Modified distance calculation in generational distance and inverted generational distance. In *Proceedings of the 8th International Conference on Evolutionary Multi-Criterion Optimization (EMO'15)*, pages 110–125, 2015.

[99] Md. Mahfuzul Islam, Alessandro Marchetto, Angelo Susi, and Giuseppe Scanniello. A multi-objective technique to prioritize test cases based on latent semantic indexing. In *Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering (CSMR'12)*, pages 21–30, 2012.

[100] David W. Hosmer Jr., Stanley Lemeshow, and Rodney X. Sturdivant. *Applied Logistic Regression*. John Wiley & Sons, Inc., 3rd edition, 2013.

[101] André I Khuri and Siuli Mukhopadhyay. Response surface methodology. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(2):128–149, 2010.

[102] Joshua D. Knowles and David W. Corne. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[103] V. P. Kozyrev. Estimation of the execution time in real-time systems. *Programming and Computer Software*, 42(1):41–48, 2016.

[104] Oliver Kramer. *Dimensionality reduction with unsupervised nearest neighbors*, chapter K-nearest neighbors, pages 13–23. Springer, 2013.

[105] Dirk P. Kroese, Tim J. Brereton, Thomas Taimre, and Zdravko I. Botev. Why the Monte Carlo method is so important today. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6:386–392, 2014.

[106] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11)*, pages 585–591, Berlin, Heidelberg, 2011. Springer.

[107] William B. Langdon, Mark Harman, and Yue Jia. Efficient multi-objective higher order mutation testing with genetic programming. *Journal of Systems and Software*, 83(12):2416–2430, 2010.

[108] Hyunsung Lee, Jinkyu Lee, Ikjun Yeom, and Honguk Woo. Panda: Reinforcement learning-based priority assignment for multi-processor real-time scheduling. *IEEE Access*, 8:185570–185583, 2020.

[109] Jaekwon Lee, Seung Yeob Shin, Shiva Nejati, and Lionel C. Briand. [Evaluation package] Optimal priority assignment method for real-time systems. `https://github.com/SNTSVV/OPAM`, 2021.

[110] Jaekwon Lee, Seung Yeob Shin, Shiva Nejati, and Lionel C. Briand. Optimal priority assignment for real-time systems: A coevolution-based approach. *CoRR*, abs/2102.07694, 2021.

[111] Jaekwon Lee, Seung Yeob Shin, Shiva Nejati, and Lionel C. Briand. [case study data] estimating probabilistic safe wcet ranges for weakly hard real-time systems. `https://figshare.com/s/27ac878bd5a30013824b`, 2022.

[112] Jaekwon Lee, Seung Yeob Shin, Shiva Nejati, Lionel C. Briand, and Yago Isasi Parache. [case study data] estimating probabilistic safe wcet ranges of real-time systems at design stages. `https://figshare.com/s/d63d32c8ee726912e3f0`, 2022.

[113] Jaekwon Lee, Seung Yeob Shin, Shiva Nejati, Lionel C. Briand, and Yago Isasi Parache. Estimating probabilistic safe wcet ranges of real-time systems at design stages. *ACM Transactions on Software Engineering and Methodology*, Jun 2022. Just Accepted.

[114] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In *Proceedings of the International Conference on Runtime Verification (RV'10)*, pages 122–135, 2010.

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

[116] Miqing Li, Tao Chen, and Xin Yao. How to evaluate solutions in pareto-based search-based software engineering: A critical review and methodological guidance. *IEEE Transactions on Software Engineering*, 48(5):1771–1799, 2022.

[117] Man Lin, Li Xu, Laurence T. Yang, Xiao Qin, Nenggan Zheng, Zhaohui Wu, and Meikang Qiu. Static security optimization for real-time systems. *IEEE Transactions on Industrial Informatics*, 5(1):22–37, 2009.

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

[119] Jane W. S. Liu. *Real-Time Systems*. Prentice Hall PTR, 1st edition, 2000.

[120] Douglas Locke, Lee Lucas, and John Goodenough. Generic avionics software specification. Technical Report CMU/SEI-90-TR-008, Software Engineering Institute, Carnegie Mellon University, 1990.

[121] Manuel Lozano, Francisco Herrera, and José Ramón Cano. Replacement strategies to preserve useful diversity in steady-state genetic algorithms. *Information Sciences*, 178(23):4421–4433, 2008.

[122] Sean Luke. *Essentials of Metaheuristics*. Lulu, 2nd edition, 2013. Available for free at http://cs.gmu.edu/~sean/book/metaheuristics/.

[123] LuxSpace. ESAIL. `https://luxspace.lu/triton-2/`, 2021.

[124] Ahmed Maged, Salah Haridy, Mohammad Shamsuzzaman, Imad Alsyouf, and Roubi Zaied. Statistical monitoring and optimization of electrochemical machining using shewhart charts and response surface methodology. *International Journal of Engineering Materials and Manufacture*, 3(2):68–77, 2018.

[125] Henry B. Mann and Donald R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60, 1947.

[126] Sorin Manolache, Petru Eles, and Zebo Peng. Schedulability analysis of applications with stochastic task execution times. *ACM Transactions on Embedded Computer System (TECS)*, 3(4):706–735, 2004.

[127] Alessandro Marchetto, Md. Mahfuzul Islam, Waseem Asghar, Angelo Susi, and Giuseppe Scanniello. A multi-objective technique to prioritize test cases. *IEEE Transactions on Software Engineering*, 42(10):918–940, 2016.

[128] Dorin Maxim and Liliana Cucu-Grosjean. Response time analysis for fixed-priority tasks with multiple probabilistic parameters. In *Proceedings of the 2013 IEEE 34th Real-Time Systems Symposium (RTSS'13)*, pages 224–235, Vancouver, BC, Canada, 2013. IEEE.

[129] Michael R. May and Brian R. Moore. How well can we detect lineage-specific diversification-rate shifts? a simulation study of sequential aic methods. *Systematic Biology*, 65(6):1076–1084, 2016.

[130] Ivan Reinaldo Meneghini, Frederico Gadelha Guimarães, and Antonio Gaspar-Cunha. Competitive coevolutionary algorithm for robust multi-objective optimization: The worst case minimization. In *Proceedings of the 2016 IEEE Congress on Evolutionary Computation (CEC'16)*, pages 586–593, 2016.

[131] Marius Mikucionis, Kim Guldstrand Larsen, and Brian Nielsen. T-UPPAAL: online model-based testing of real-time systems. In *Proceedings of the 19th International Conference on Automated Software Engineering (ASE'04)*, pages 396–397, Linz, Austria, 2004. IEEE.

[132] Marius Mikučionis, Kim Guldstrand Larsen, Jacob Illum Rasmussen, Brian Nielsen, Arne Skou, Steen Ulrik Palm, Jan Storbank Pedersen, and Poul Hougaard. Schedulability analysis using UPPAAL: Herschel-Planck case study. In *Proceedings of the International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA'10)*, pages 175—190, Berlin, Heidelberg, 2010. Springer.

[133] Frank Mueller. Timing analysis for instruction caches. *Real-Time Systems*, 18(2):217–247, 2000.

[134] Pranab K. Muhuri and Kaushal Kumar Shukla. Real-time scheduling of periodic tasks with processing times and deadlines as parametric fuzzy numbers. *Applied Soft Computing*, 9(3):936–946, 2009.

[135] Jagannath Munda and Bijoy Bhattacharyya. Investigation into electrochemical micromachining (emm) through response surface methodology based approach. *The International Journal of Advanced Manufacturing Technology*, 35:821–832, 2008.

[136] In Jae Myung. Tutorial on maximum likelihood estimation. *Journal of Mathematical Psychology*, 47(1):90–100, 2003.

[137] Shiva Nejati, Morayo Adedjouma, Lionel C. Briand, Jonathan Hellebaut, Julien Begey, and Yves Clement. Minimizing CPU time shortage risks in integrated embedded software. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE'13)*, pages 529–539, 2013.

[138] Shiva Nejati, Stefano Di Alesio, Mehrdad Sabetzadeh, and Lionel Briand. Modeling and analysis of CPU usage in safety-critical embedded systems to support stress testing. In *Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems (MODELS'12)*, pages 759–775, Berlin, Heidelberg, 2012. Springer.

[139] Shiva Nejati and Lionel C. Briand. Identifying optimal trade-offs between CPU time usage and temporal constraints using search. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA'14)*, pages 351–361, 2014.

[140] John A. Nelder and Roger Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[141] Thanh-Tung Nguyen, Joshua Zhexue Huang, and Thuy Thi Nguyen. Unbiased feature selection in learning random forests for high-dimensional data. *The Scientific World Journal*, 2015:1–18, 2015.

[142] Fred C. Pampel. *Logistic Regression: A Primer*. Sage publications, 2000.

[143] Paolo Pazzaglia, Youcheng Sun, and Marco Di Natale. Generalized weakly hard schedulability analysis for real-time periodic tasks. *ACM Transactions on Embedded Computing Systems*, 20(1):1–26, 2021.

[144] Dar-Tzen Peng, Kang G. Shin, and Tarek F. Abdelzaher. Assignment and scheduling communicating periodic tasks in distributed real-time systems. *IEEE Transactions on Software Engineering*, 23(12):745–758, 1997.

[145] Marie-Agnès Peraldi-Frati and Yves Sorel. From high-level modelling of time in marte to real-time scheduling analysis. In *Proceedings of the MODELS'08 Workshop on Model Based Architecting and Construction of Embedded Systems (ACESMB'08)*, pages 129–144, Toulouse, France, 2008. CEUR-WS.

[146] Elena Popovici, Anthony Bucci, R. Paul Wiegand, and Edwin D. De Jong. *Handbook of Natural Computing*, chapter Coevolutionary principles, pages 987–1033. Springer, 2012.

[147] Mitchell A. Potter and Kenneth A. De Jong. A cooperative coevolutionary approach to function optimization. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 866 LNCS:249–257, 1994.

[148] Paul Ralph. The sensemaking-coevolution-implementation theory of software design. *Science of Computer Programming*, 101:21–41, 2015.

[149] Paul Ralph, Nauman bin Ali, Sebastian Baltes, Domenico Bianculli, Jessica Diaz, Yvonne Dittrich, Neil Ernst, Michael Felderer, Robert Feldt, Antonio Filieri, Breno Bernard Nicolau de França, Carlo Alberto Furia, Greg Gay, Nicolas Gold, Daniel Graziotin, Pinjia He, Rashina Hoda, Natalia Juristo, Barbara Kitchenham, Valentina Lenarduzzi, Jorge Martínez, Jorge Melegati, Daniel Mendez, Tim Menzies, Jefferson Molleri, Dietmar Pfahl, Romain Robbes, Daniel Russo, Nyyti Saarimäki, Federica Sarro, Davide Taibi, Janet Siegmund, Diomidis Spinellis, Miroslaw Staron, Klaas Stol, Margaret-Anne Storey, Davide Taibi, Damian Tamburri, Marco Torchiano, Christoph Treude, Burak Turhan, Xiaofeng Wang, and Sira Vegas. Empirical standards for software engineering research, 2020.

[150] Jian Ren, Mark Harman, and Massimiliano Di Penta. Cooperative co-evolutionary optimization of software project staff assignments and job scheduling. In *Proceedings of the International Symposium on Search Based Software Engineering (SSBSE'11)*, pages 127–141, Berlin, Heidelbergl, 2011. Springer.

[151] Irina Rish et al. An empirical study of the naive bayes classifier. In *Proceedings of the IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001.

[152] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach*. Pearson Education, 3rd edition, 2010.

[153] Luca Santinelli, Fabrice Guet, and Jerome Morio. Revising measurement-based probabilistic timing analysis. In *Proceedings of the 2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'17)*, pages 199–208, 2017.

[154] Abdel Salam Sayyad, Katerina Goseva-Popstojanova, Tim Menzies, and Hany Ammar. On parameter tuning in search based software engineering: A replicated empirical study. In *Proceedings of the 2013 3rd International Workshop on Replication in Empirical Software Engineering Research (RESER'13)*, pages 84–90, 2013.

[155] Bernhard Schölkopf, Alex J Smola, Robert C Williamson, and Peter L Bartlett. New support vector algorithms. *Neural computation*, 12(5):1207–1245, 2000.

[156] Syed Abdul Baqi Shah, Muhammad Rashid, and Muhammad Arif. Estimating WCET using prediction models to compute fitness function of a genetic algorithm. *Real-Time Systems*, 56(1):28–63, 2020.

[157] Seung Yeob Shin, Shiva Nejati, Mehrdad Sabetzadeh, Lionel C. Briand, Chetan Arora, and Frank Zimmer. Dynamic adaptation of software-defined networks for IoT systems: A search-based approach. In *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'20)*, pages 137–148, 2020.

[158] Seung Yeob Shin, Shiva Nejati, Mehrdad Sabetzadeh, Lionel C. Briand, and Frank Zimmer. Test case prioritization for acceptance testing of cyber physical systems: A multi-objective search-based approach. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'18)*, pages 49–60, 2018.

[159] S. N. Sivanandam and S. N. Deepa. *Introduction to Genetic Algorithms*. Springer-Verlag, 1 edition, 2008.

[160] Dylan Slack, Sorelle A Friedler, Carlos Scheidegger, and Chitradeep Dutta Roy. Assessing the local interpretability of machine learning models. *arXiv preprint arXiv:1902.03501*, 2019.

[161] Douglas R. Smith. The design of divide and conquer algorithms. *Science of Computer Programming*, 5:37–58, 1985.

[162] D. Sofge, K. De Jong, and A. Schultz. A blended population approach to cooperative coevolution for decomposition of complex problems. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02)*, volume 1, pages 413–418, 2002.

[163] John A. Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio C. Buttazzo. *Deadline Scheduling for Real-Time Systems - EDF and Related Algorithms*. Springer, 1998.

[164] Jill C Stoltzfus. Logistic regression: a brief primer. *Academic emergency medicine*, 18(10):1099–1104, 2011.

[165] Youcheng Sun and Marco Di Natale. Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks. *ACM Transactions on Embedded Computing Systems*, 16(5s), 2017.

[166] Youcheng Sun, Romain Soulat, Giuseppe Lipari, Étienne André, and Laurent Fribourg. Parametric schedulability analysis of fixed priority real-time distributed systems. In *Proceedings of the 2nd International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS'13)*, volume 419, pages 212–228, Cham, 2013. Springer.

[167] Yasunari Takagi, Osamu Mizuno, and Tohru Kikuno. An empirical approach to characterizing risky software projects based on logistic regression analysis. *Empirical Software Engineering*, 10(4):495–515, 2005.

[168] Shin Hwei Tan, Hiroaki Yoshida, Mukul R. Prasad, and Abhik Roychoudhury. Anti-patterns in search-based program repair. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'16)*, pages 727–738, 2016.

[169] Henrik Theiling, Christian Ferdinand, and Reinhard Wilhelm. Fast and Precise WCET Prediction by Separated Cache and Path Analyses. *Real-Time Systems*, 18(2):157–179, 2000.

[170] K. W. Tindell, Richard A. Burns, and A.J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, 1994.

[171] Karim Traore, Emmanuel Grolleau, and Francis Cottet. Simpler analysis of serial transactions using reverse transactions. In *Proceedings of the 2006 International Conference on Autonomic and Autonomous Systems (ICAS'06)*, page 11, Silicon Valley, CA, USA, 2006. IEEE.

[172] András Vargha and Harold D. Delaney. A critique and improvement of the CL common language effect size statistics of mcgraw and wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[173] Sébastien Varrette, Pascal Bouvry, Hyacinthe Cartiaux, and Fotis Georgatos. Management of an academic HPC cluster: The UL experience. In *Proceedings of the 2014 International Conference on High Performance Computing & Simulation (HPCS'14)*, pages 959–967, Bologna, Italy, 2014. IEEE.

[174] David A Van Veldhuizen and Gary B Lamont. Multiobjective evolutionary algorithm research : A history and analysis. Technical Report TR–98–03, Air Force Institute of Technology, Wright-Patterson AFB, 1998.

[175] Georg von der Brüggen, Nico Piatkowski, Kuan-Hsun Chen, Jian-Jia Chen, and Katharina Morik. Efficiently approximating the probability of deadline misses in real-time systems. In *Proceedings of the 30th Euromicro Conference on Real-Time Systems (ECRTS'18)*, volume 106, pages 1–22, Dagstuhl, Germany, 2018. Schloss Dagstuhl.

[176] Georg von der Brüggen, Nico Piatkowski, Kuan-Hsun Chen, Jian-Jia Chen, and Katharina Morik. Efficiently Approximating the Probability of Deadline Misses in Real-Time Systems. In *Proceedings of the 30th Euromicro Conference on Real-Time Systems (ECRTS'18)*, volume 106, pages 1–22, Dagstuhl, Germany, 2018. Schloss Dagstuhl.

[177] K. C. Wang. *Embedded and Real-Time Operating Systems*. Springer, 1st edition, 2017.

[178] Penghong Wang, Jianrou Huang, Zhihua Cui, Liping Xie, and Jinjun Chen. A gaussian error correction multi-objective positioning model with NSGA-II. *Concurrency and Computation: Practice and Experience*, 32(5):1–16, 2020.

[179] Joachim Wegener and Matthias Grochtmann. Verifying timing constraints of real-time systems by means of evolutionary testing. *Real-Time Systems*, 15(3):275–298, 1998.

[180] Joachim Wegener, Harmen Sthamer, Bryan F. Jones, and David E. Eyres. Testing real-time systems using genetic algorithms. *Software Quality Journal*, 6(2):127–135, 1997.

[181] Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest. Automatically finding patches using genetic programming. In *Proceedings of the 31st International Conference on Software Engineering (ICSE'09)*, pages 364–374, 2009.

[182] Sanford Weisberg. *Applied linear regression*, volume 528. John Wiley & Sons, 2005.

[183] I. Wenzel, R. Kirner, B. Rieder, and P. Puschner. Measurement-based worst-case execution time analysis. In *Proceedings of the Third IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS'05)*, pages 7–10, 2005.

[184] Darrell Whitley and Joan Kauth. GENITOR: A different genetic algorithm. In *Proceedings of the 1988 Rocky Mountain Conference on Artificial Intelligence*, pages 118–130, 1988.

[185] Josh L. Wilkerson and Daniel Tauritz. Coevolutionary automated software correction. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO'10)*, pages 1391–1392, 2010.

[186] Josh L. Wilkerson, Daniel R. Tauritz, and James M. Bridges. Multi-objective coevolutionary automated software correction. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation (GECCO'12)*, pages 1229–1236, 2012.

[187] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., 3rd edition, 2011.

[188] Changjiu Xian, Yung-Hsiang Lu, and Zhiyuan Li. Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time. In *Proceedings of the 2007 44th ACM/IEEE Design Automation Conference (DAC'07)*, pages 664–669, San Diego, CA, USA, 2007. IEEE.

[189] Wenbo Xu, Zain Alabedin Haj Hammadeh, Alexander Kröller, Rolf Ernst, and Sophie Quinton. Improved deadline miss models for real-time systems using typical worst-case analysis. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS'15)*, pages 247–256, Lund, Sweden, 2015. IEEE.

[190] Beyazit Yalcinkaya, Mitra Nasri, and Björn B. Brandenburg. An exact schedulability test for non-preemptive self-suspending real-time tasks. In *Proceedings of the 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE'19)*, pages 1228–1233, Florence, Italy, 2019. IEEE.

[191] Toshie Yamashita, Keizo Yamashita, and Ryotaro Kamimura. A stepwise aic method for variable selection in linear regression. *Communications in Statistics - Theory and Methods*, 36(13):2395–2403, 2007.

[192] Fei Yu, Guoqiang Li, and Naixue Xiong. Schedulability analysis of multi-processor real-time systems using UPPAAL. In *Proceedings of the 2nd International Conference on Information Science and Engineering (ICISE'10)*, pages 1–6, Hangzhou, China, 2010. IEEE.

[193] Justyna Zander. *Model-based Testing of Real-Time Embedded Systems in the Automotive Domain*. PhD thesis, Fraunhofer FOKUS, 2008.

[194] Haibo Zeng, Marco Di Natale, and Qi Zhu. Minimizing stack and communication memory usage in real-time embedded applications. *ACM Transactions on Embedded Computing Systems*, 13(5s):1–25, Dec 2014.

[195] Fengxiang Zhang and Alan Burns. Schedulability analysis for real-time systems with EDF scheduling. *IEEE Transactions on Computers*, 58(9):1250–1258, 2009.

[196] Zhongheng Zhang. Variable selection with stepwise and best subset approaches. *Annals of Translational Medicine*, 4(7):1–6, 2016.

[197] Yecheng Zhao and Haibo Zeng. The virtual deadline based optimization algorithm for priority assignment in fixed-priority scheduling. In *Proceedings of the 2017 IEEE Real-Time Systems Symposium (RTSS'17)*, pages 116–127, 2017.

[198] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. Technical report, 2001.

[199] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.