Model-based optimization of ARINC-653 partition scheduling

Han, Pujie; Zhai, Zhengjun; Nielsen, Brian; Nyman, Ulrik

Published in:
International Journal on Software Tools for Technology Transfer

DOI (link to publication from Publisher):
10.1007/s10009-020-00597-6

Publication date:
2021

Document Version
Other version

Link to publication from Aalborg University

Citation for published version (APA):
Han, P., Zhai, Z., Nielsen, B., & Nyman, U. (2021). Model-based optimization of ARINC-653 partition scheduling. *International Journal on Software Tools for Technology Transfer*, *23*(5), 721-740. https://doi.org/10.1007/s10009-020-00597-6

Noname manuscript No.
(will be inserted by the editor)

Model-based optimization of ARINC-653 partition scheduling

Pujie Han · Zhengjun Zhai · Brian Nielsen · Ulrik Nyman

Received: date / Accepted: date

Abstract The architecture of ARINC-653 partitioned scheduling has been widely applied to avionics systems owing to its robust temporal isolation among applications. However, this partitioning mechanism causes the problem of how to optimize the partition scheduling of a complex system while guaranteeing its schedulability. In this paper, a model-based optimization approach is proposed. We formulate the problem as a parameter sweep application, which searches for the optimal partition scheduling parameters with respect to minimum processor occupancy via an evolutionary algorithm. An ARINC-653 partitioned scheduling system is modeled as a set of timed automata (TA) in the model checker Uppaal. The optimizer tentatively assigns parameter settings to the TA models and subsequently invokes Uppaal to verify schedulability as well as evaluate promising solutions. The parameter space is explored with an evolutionary algorithm that combines refined genetic operators and the self-adaptation of evolution strategies. The experimental results show the applicability of our optimization method.

Keywords partitioned scheduling · model-based optimization · parameter sweep · evolutionary algorithm · timed automata · Uppaal

1 Introduction

As the performance of embedded processors rapidly increases, there is a growing trend towards integrating multiple real-time applications into a partitioned scheduling system in avionics development. The ARINC 653 standard [1] prescribes a robust temporal partitioning mechanism for Integrated Modular Avionics (IMA) systems, where a global scheduler assigns a fraction of processor time to a temporally isolated partition that contains a set of concurrent tasks. A local scheduler of the partition manages the included tasks. The application of partitioned scheduling is effectively able to prevent failure propagation among partitions. However, it raises the question of how to allocate processor time to partitions in an optimal manner while guaranteeing their time requirements.

In ARINC 653, the time allocation for partitions is executed cyclically according to a static schedule. A schedulable system requires sufficient time allocation for all partitions. The time requirement of a partition is described as a tuple of periodic scheduling parameters $\langle \mathit{period}, \mathit{budget} \rangle$, which can be used for generating the static schedule [1]. Given the set of specific real-time applications in the system, these parameters determine not only the schedulability of the system but also its processor occupancy. In this paper, the question of resource allocation is interpreted as the optimization of ARINC-653 partition scheduling parameters of a schedulable system. The goal is to minimize the processor occupancy of the system, thus making it possible to accommodate more additional workload of applications [34].

The nature of ARINC-653 partition scheduling is a complex non-linear non-convex parameter optimization problem [34]. So far, most investigations [30, 13, 21, 34,

This work was in part funded by Independent Research Fund Denmark under grant number DFF-7017-00348, Compositional Verification of Real-time MULTI-CORE SAFETY Critical Systems

Pujie Han · Zhengjun Zhai
School of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an, 710072, China
E-mail: hanpujie@mail.nwpu.edu.cn, zhaizjun@nwpu.edu.cn

Brian Nielsen · Ulrik Nyman
Department of Computer Science, Aalborg University, Aalborg, 9220, Denmark
E-mail: {bnielsen, ulrik}@cs.aau.dk

22] have been confined to analytical methods, whose rigorous mathematical models build on the worst-case assumptions of a simplified system. In more complex real-time applications, more expressive model-checking (MC) approaches [11,33,10,9,23,8] are extensively being developed to incorporate a great variety of behavioral features including concrete task actions, dependency and communications. They are based on various formal models such as preemptive Time Petri Nets (pTPN), Linear Hybrid Automata (LHA), and Timed Automata (TA). For each promising scheduling scheme, its schedulability can be verified or falsified automatically via state space exploration of the system model.

However, to identify a globally optimal scheduling configuration, the entire combinatorial parameter space must be explored thoroughly. Each of these combinations leads to a single model-checking operation which is in itself a PSPACE-complete problem. Therefore, we use Evolutionary Algorithm (EA) as a heuristic optimization method, thereby avoiding the brute-force search of parameter space.

The model-based methods are also confronted with the state space explosion problem, which makes the exact model checking practically infeasible. There have been several promising techniques that attempt to mitigate the state space explosion of classical MC. Statistical Model Checking (SMC) [28] is a simulation-based method that runs and monitors a number of simulation processes, providing the statistical results of verification with a certain degree of confidence. However, SMC cannot provide any guarantee of schedulability but quick falsification owing to its nature of statistical testing. By contrast, compositional approaches [25] decompose the system into components, check each component separately by classical MC and conclude system properties at a global level, but might offer conservative results due to abstraction of the components. Therefore, it is reasonable to combine the global SMC and compositional MC techniques. Nevertheless, we found no studies that applied such a combination to the optimization of ARINC-653 partition scheduling.

Uppaal [3] is a model-checking toolbox for modeling and verifying real-time systems described as extended TA, which is expressive enough to cover features of an IMA system. There are several branches in the Uppaal family. The classical Uppaal and Uppaal SMC [12] provide the implementation of symbolic MC and SMC respectively. In the previous work [18, 19], we have integrated the global SMC and compositional MC into a Uppaal-based schedulability analysis of IMA systems.

In this paper, we propose a model-based optimization method of ARINC-653 partition scheduling for IMA systems. The core idea is to extend the Uppaal TA model of the system with a parameter sweep application that searches for the optimal schedulable solutions with respect to minimum processor occupancy. Our main contributions include:

- *A model-based optimization method* that addresses the optimal time allocation of partitioned scheduling systems by performing a heuristic search of the objective parameter space of the Uppaal TA model.
- *A* Uppaal*-based modeling and analysis technique* that supports parameter sweep by quickly falsifying non-schedulable solutions and evaluating schedulable ones. An IMA system is modeled as TA models in Uppaal and its schedulability constraints are verified automatically via the integrated method of global SMC and compositional MC analysis.
- *A generator of ARINC-653 partition schedules* that connects the parameter optimizer and the Uppaal TA models of an IMA system to enable the automatic design of IMA partition scheduling.
- *An evolutionary algorithm* that combines refined genetic search operators and the adaptation of evolution strategies, thereby accelerating the process of finding optimal solutions and meanwhile reducing the risk of premature convergence.

The rest of the paper is organized as follows. Section 2 gives the definition of the optimization problem. Section 3 provides a background of the schedulability analysis. Section 4 introduces the parameter optimization method and briefly presents its constituent components. We detail the evolutionary algorithm EA4HS in Section 5. The experiments on sample systems are shown in Section 6. Section 7 gives the related work and Section 8 finally concludes.

2 Optimization Problem Description

In this section, we first outline an IMA partitioned scheduling system, and then give the definition of its parameter optimization problem.

2.1 System Model

We focus on a two-level partitioned scheduling system where partitions are scheduled by a Time Division Multiplexing (TDM) global scheduler and each partition also has a local scheduler based on preemptive Fixed Priority (FP) policy to manage the partition's internal tasks.

The system consists of a set of temporal partitions $\Omega = \{\mathcal{P}_i | i = 1, 2, \ldots, n\}$ running on a single processor.

The TDM global scheduler executes time allocation for partitions according to a static schedule $\mathcal{S}$ cyclically and repeats $\mathcal{S}$ every major time frame M [1]. The partition schedule $\mathcal{S}$ is comprised of a set of partition time windows: $\mathcal{S} = \{W_t | t = 1, 2, \ldots, w\}$. W_t is a time slot $\langle P_t, o_t, d_t \rangle$ belonging to a partition $P_t \in \Omega$, where o_t and d_t denote the offset from the start of M and expected duration respectively. The w time slots are non-overlapping, satisfying that $0 \leq o_1 < o_1 + d_1 < o_2 < o_2 + d_2 < \cdots < o_w < o_w + d_w \leq M$. Partitions are activated only during their partition time windows within M.

Each partition $\mathcal{P}_i$ accommodates a set of tasks $\Gamma_i = \{\tau^i_j | j = 1, 2, \ldots, m_i\}$ which are scheduled by the local scheduler of $\mathcal{P}_i$ in accordance with the preemptive FP policy and executed only when $\mathcal{P}_i$ is activated. A task τ is represented by the tuple $\langle I, T, O, J, D, R, L \rangle$ where I is initial offset, T is release interval, O is offset, J is jitter, $D \leq T$ is deadline, R denotes task priority, and L describes the behavior of τ as a sequential list. Each element of L is an abstract instruction $\langle Cmd, Res, T_{BCET}, T_{WCET} \rangle$. *Cmd* is an operation code in the command set {*Compute*, *Lock*, *Unlock*, *Delay*, *Send*, *Receive*, *End*}. *Res* is an identifier encoding one of the resources such as processor time, locks, and messages. T_{BCET} and T_{WCET} are execution time in the best case and the worst case respectively. In the command set, *Compute* denotes a general computation step, *Lock* and *Unlock* handle locks, *Delay* allows the task to stop running for a certain time, *Send* and *Receive* are used for inter-partition communications, and *End* is the symbol of job termination.

2.2 Schedulability Condition

The schedulability of a partitioned scheduling system can also be divided into conditions at the global and the local level. Figure 1 shows this hierarchical scheduling architecture.

At the global level, the schedulability is that the time supply of the global scheduler satisfies the time requirement of each partition. The partition schedule $\mathcal{S}$ defines the time supply for partitions in the system. According to the ARINC 653 standard, the time requirement of a partition $\mathcal{P}_i$ can be described as a tuple of periodic scheduling parameters $\langle p_i, b_i \rangle$ where p_i is a partition period and b_i is the budget within p_i. Thus the schedulability condition denotes that the budget b_i can be guaranteed by the partition schedule $\mathcal{S}$ during each period p_i. Compared with the variable-length partition schedule, we are more interested in handling the concise parameter tuple $\langle p_i, b_i \rangle$ that is used as an input in determining the partition time windows of $\mathcal{P}_i$ [1].

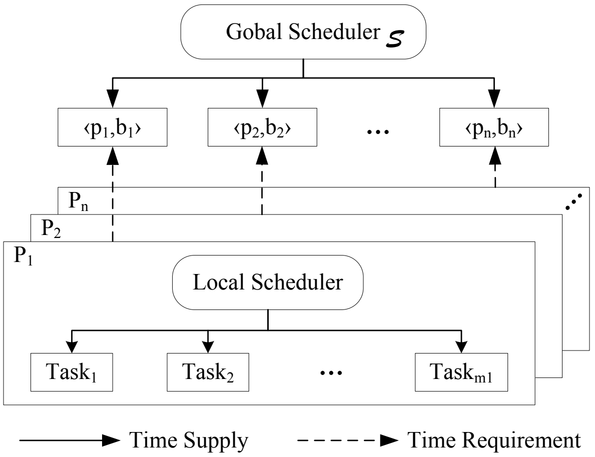


Fig. 1 Hierarchical architecture of partitioned scheduling systems

The schedulability at the local level requires all tasks to meet their deadlines. The tuple of scheduling parameters $\langle p_i, b_i \rangle$ indicates the total periodic time requirement of tasks in $\mathcal{P}_i$. We define two types of tasks:

- A *periodic task* has the kth release time $t_k \in [I + kT + O, I + kT + O + J]$ where $k \in \mathbb{N}$ and T denotes a fixed period. A periodic task meets its deadline iff the task can finish its kth job before the instant $(I + kT + D)$ for any $k \in \mathbb{N}$.
- A *sporadic task* characterized by a minimum separation T between consecutive jobs releases its $(k+1)$th job at $t_{k+1} \in [t_k + T, +\infty)$, and its first release is at $t_0 \in [I, +\infty)$. A sporadic task complies with its deadline iff its kth job can be completed before $(t_k + D)$ for any $k \in \mathbb{N}$.

In addition, the ARINC-653 standard allows tasks to perform two types of communication between them: intra- and inter-partition communication. The type of a communication operation of a task depends on whether the communicating tasks are located in the same partition. The behavior of resource sharing or message communication incurs the task-blocking overheads that could affect the schedulability of partitions at the local level. Hence our model-based method also needs to describe the concrete task behavior including the (intra- and inter-partition) communication precisely in Uppaal models.

2.3 Optimization Problem

Consider the aforementioned partitioned scheduling system. Given a set of partitions $\Omega = \{\mathcal{P}_i | i = 1, 2, \ldots, n\}$ and their respective task sets $\{\Gamma_i\}$, the optimization problem is to find a $2n$-dimensional vector $\mathbf{x} = (x_1, x_2, \ldots, x_{2n}) \in \mathbb{R}^{2n}_+$ where the parameter tuple $\langle p_i, b_i \rangle$ of $\mathcal{P}_i$

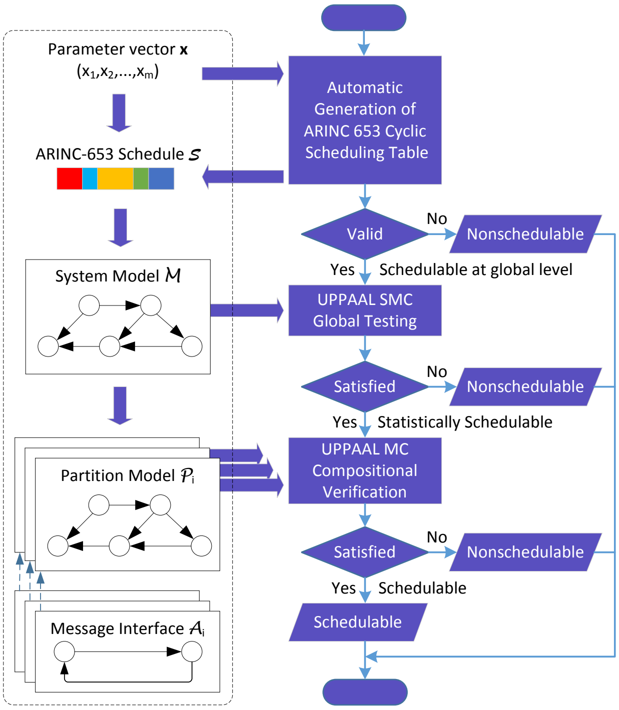

Parameter vector **x**
(x₁,x₂,...,xₘ)
ARINC-653 Schedule $\mathcal{S}$
System Model $\mathcal{M}$
Partition Model $\mathcal{P}_i$
Message Interface $\mathcal{A}_i$
Automatic Generation of ARINC 653 Cyclic Scheduling Table
Valid
No
Nonschedulable
Yes
Schedulable at global level
UPPAAL SMC Global Testing
Satisfied
No
Nonschedulable
Yes
Statistically Schedulable
UPPAAL MC Compositional Verification
Satisfied
No
Nonschedulable
Yes
Schedulable
Schedulable

Fig. 2 Flowchart of schedulability analysis

corresponds to the elements $x_{2i-1} = p_i$ and $x_{2i} = b_i$, such that the system minimizes the processor occupancy U while guaranteeing the schedulability at both the global and local level.

Suppose each release of partitions needs a context switch. The processor occupancy is defined as

$$U = \sum_{i=1}^{n} \frac{c_i \cdot v + b_i}{p_i}, \tag{1}$$

where c_i is the average number of context switching for $\mathcal{P}_i$ during each partition period p_i, and v is the context-switch overhead.

Minimizing the processor occupancy of a partitioned scheduling system makes it possible to accommodate more additional workload of applications. Similar definitions of the processor occupancy function have been proposed and applied in previous papers [13,34], where it was called “processor utilization” or “system utilization”. We found these names counter-intuitive, because we normally chase a higher “utilization” but it should be minimized in this problem. Thus we renamed it processor occupancy in this paper. Equivalently, we also define the remaining processor utilization $U_r = 1 - U$ and find the maximum U_r instead.

3 Background of Schedulability Analysis

In this section, we formulate the schedulability constraints of the optimization problem on the basis of the modeling formalism of Uppaal. The behavior of the partitioned scheduling system presented in section 2.1 is further modeled as a set of Uppaal templates. A template is a generalized object of TA in Uppaal. The automaton structure of a template consists of locations and edges. A template may also have local variables and functions. The templates can be instantiated as a network of TA model instances $\mathcal{M}$ that describe a complete system. For any scheduling parameter vector $\mathbf{x}$, the schedulability of its system model is verified or falsified according to the procedure in Fig. 2, where the right is the flowchart of our model-based analysis and the left dashed-line box contains the data objects of each process.

First, an ARINC-653 partition schedule $\mathcal{S}$ is generated automatically from the input parameter vector $\mathbf{x}$ via an partition scheduling algorithm, which guarantees $\mathcal{S}$ satisfies the time requirement of $\mathbf{x}$, i.e. schedulability at the global level. We refer to $\mathbf{x}$ as a *valid* parameter combination if a partition schedule can be generated from $\mathbf{x}$, then the schedulability analysis will proceed with the following costly steps. Otherwise, it will conclude with the invalidity of $\mathbf{x}$. A partition scheduling algorithm is presented in section 4.2.

Second, the schedulability constraints of the optimization problem are expressed and fast falsified as queries of *hypothesis testing* in Uppaal SMC. We add a boolean array `perror` with the initial value `False` to TA templates for this purpose. Once the schedulability of partition $\mathcal{P}_i$ is violated, the related model will assign the value `True` to `perror[i]` immediately. Thus, the schedulability constraints for $\mathcal{P}_i$ are replaced with the following query ρ_i:

$$\texttt{Pr[<= N](<> perror[i]) <=}\ \theta,\ i = 1, 2, \ldots, n \tag{2}$$

where N is the time bound on the simulations and θ is a very low probability. Uppaal SMC is invoked to estimate whether the system model $\mathcal{M}$ satisfies the conjunction of n queries statistically:

$$\mathcal{M} \models \rho_1 \wedge \rho_2 \wedge \cdots \wedge \rho_n \tag{3}$$

Since Uppaal SMC approximates the answer using simulation based algorithms, we can falsify any nonschedulable solution rapidly but identify schedulable ones only with high probability $(1-\theta)$. Note that the probability distributions used in such models affect the probabilities of events in the overall model. In our case this is not important as we do not evaluate the probability of the events, but only search for a single trace violating the schedulability. Therefore, all schedulable results of SMC testing should be validated by classical MC to confirm the schedulability of the corresponding system.

Finally, in order to alleviate the state-space explosion problem of classical MC, we apply our compositional method presented in [20] to *schedulability validation*, which is comprised of the following four steps:

1. *Decomposition:* The system model $\mathcal{M}$ is first decomposed into a set of communicating partitions models $\mathcal{P}_i,\ i = 1, 2, \dots, n$. The schedulability property is also divided into n TCTL (Timed Computation Tree Logic) safety properties φ_i:

   $$\texttt{A[] not perror[i]},\ i = 1, 2, \dots, n, \tag{4}$$

   each of which belongs to one partition.
2. *Construction of message interfaces:* We define a message interface $\mathcal{A}_i$ as the assumption of the communication environment for each partition $\mathcal{P}_i$. $\mathcal{A}_i$ contains a set of TA models that mimic the requisite message-sending behavior of the other partitions.
3. *Model checking:* We check each partition model $\mathcal{P}_i$ including its environment assumption $\mathcal{A}_i$ individually by verifying the local properties φ_i:

   $$\mathcal{P}_i \| \mathcal{A}_i \models \varphi_i,\ i = 1, 2, \dots, n \tag{5}$$

   where the operator $\|$ denotes composition of two TA models.
4. *Deduction:* According to the assume-guarantee paradigm, we assemble the n local results together to derive conclusions about the schedulability of an entire system $\mathcal{M}$.

The optimization method proposed in the next section builds on the above analysis approach, which guarantees the schedulability constraints in search of the optimal solutions.

4 Parameter Optimization Method

The parameter optimization method presented in this section belongs to a class of random search methods. The optimizer searches for the (nearly) optimal schedulable parameters with respect to minimum processor occupancy U. Each search point in the considered parameter space can be converted into a promising ARINC-653 partition schedule. We finally give a Uppaal template framework that describes an IMA partitioned scheduling system as a network of TA models.

4.1 Parameter Sweep Optimizer

The optimizer is structured as a *Parameter Sweep Application* (PSA) that comprises a set of independent "experiments", each of which is performed by a PSA task with a different set of parameters [17]. These PSA tasks tentatively explore the parameter space of $\langle p_i, b_i\rangle^n$ to find promising search points.

For any search point $\mathbf{x}$, the optimizer creates a PSA task that carries out the following procedure depicted in Fig.3:

(1) A search algorithm first offers a promising parameter vector $\mathbf{x}$ to the PSA task. (2) An ARINC-653 partition schedule is then generated from the parameter setting of $\mathbf{x}$. (3) The PSA task instantiates the Uppaal modeling framework by assigning the partition schedule to the TA models and (4) subsequently invokes Uppaal SMC to execute a fast global schedulability test. (5) If the TA model goes through the SMC test, it should be validated by Uppaal classic via compositional analysis. (6) The schedulability constraints and processor occupancy are evaluated by the objective function. (7) The search algorithm receives feedback on the evaluation of $\mathbf{x}$ to update its candidate solutions and exploration direction. (8) Finally, this PSA task finishes its experiment and waits for the next call from the optimizer. The optimizer will continue the parameter sweep, based upon the results of previous experiments, until the optimization criteria are reached. The best scheduling parameter vector of $\mathbf{x}$ and its partition schedule will be output at the end of the parameter sweep.

Each component of the parameter sweep optimizer copes with a specific issue of the optimization problem.

A *search algorithm* guides the parameter sweep to select search points until an acceptable solution is found. We consider that exhaustive search is mostly infeasible and derivative information also unavailable for complex systems, thus employing an evolutionary algorithm to perform a heuristic search of the parameter space. Since there are no communications or data dependencies among PSA tasks, we adopt parallel search policies that distribute PSA tasks over several computing nodes so as to speed up the parameter sweep. Section 5 details the design of this evolutionary algorithm.

An *ARINC-653 schedule generator* converts the parameter vector $\mathbf{x}$ into an ARINC-653 static partition schedule by using an offline scheduling algorithm, which can make all scheduling decisions prior to run-time. This generator connects the parameter sweep optimizer and the Uppaal TA models of an IMA system to enable the automatic design of ARINC-653 partition scheduling. Section 4.2 gives an implementation of the generator based on the preemptive FP scheduling policy.

A Uppaal *template framework* describes a partitioned scheduling system as a network of TA models. Since Uppaal supports arrays and user-defined types, the ARINC-653 partition schedule is encoded into a structure array `partition_windows` where each element

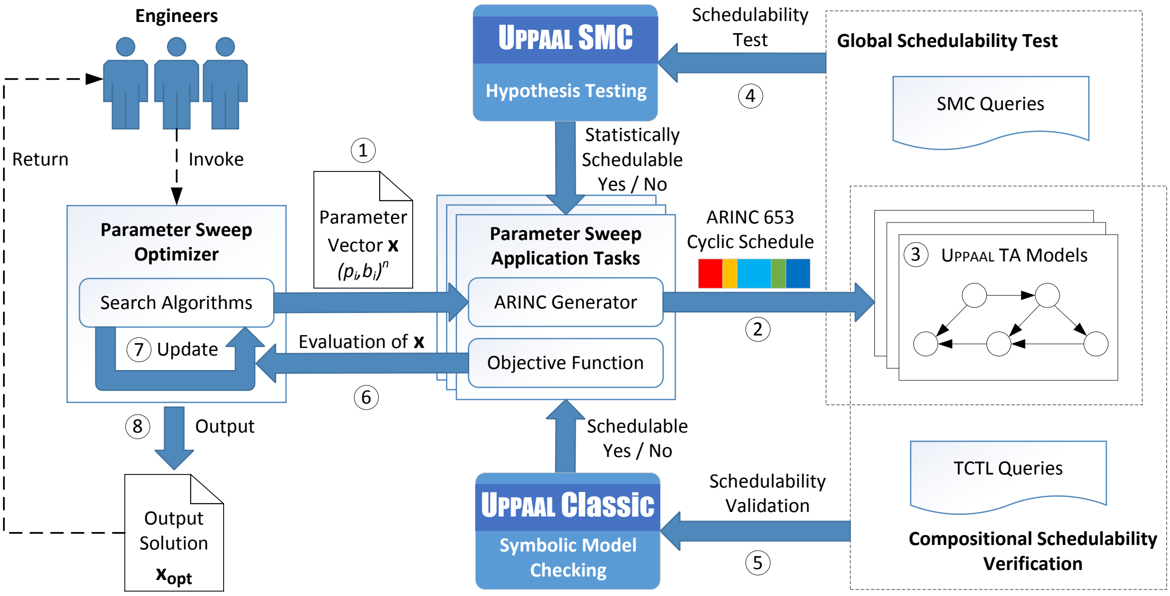

Engineers
Return
Invoke
Parameter Sweep Optimizer
Search Algorithms
7 Update
8 Output
Output Solution $\mathbf{x}_{opt}$
1
Parameter Vector **x** $(p_i,b_i)^n$
Evaluation of **x**
6
UPPAAL SMC
Hypothesis Testing
Schedulability Test
4
Statistically Schedulable Yes / No
Parameter Sweep Application Tasks
ARINC Generator
Objective Function
ARINC 653 Cyclic Schedule
2
Global Schedulability Test
SMC Queries
3 UPPAAL TA Models
Schedulable Yes / No
UPPAAL Classic
Symbolic Model Checking
Schedulability Validation
5
TCTL Queries
Compositional Schedulability Verification

Fig. 3 Architecture of parameter sweep optimizer

corresponds to a partition time window. The global scheduler modeled as a TA template `GS` executes partition scheduling according to the array records. When instantiating the templates, a PSA task should assign the array of its partition schedule to a copy of the UPPAAL model file. The UPPAAL templates are presented in section 4.3.

The *schedulability constraints* of the optimization problem are expressed as three properties: (1) validity of $\mathbf{x}$, (2) hypotheses of the SMC testing, and (3) TCTL safety properties in the MC compositional analysis. For any $\mathbf{x}$, the schedulability of its corresponding system is verified or falsified in the form of these properties according to the procedure in section 3. The results of this schedulability analysis are transferred from the ARINC-653 schedule generator or UPPAAL to the objective function in the optimizer.

The *objective function* of the optimization problem provides a quality evaluation for any parameter vector $\mathbf{x}$. Since the processor occupancy U of Eq. (1) is only valid for schedulable parameter vectors, we define the objective of the evolutionary search as a fitness function, which evaluates the remaining processor utilization U_r of any $\mathbf{x}$ on the basis of schedulability constraints. The evaluation of $\mathbf{x}$ is to update the state and search direction of the evolutionary algorithm. We give the definition of this fitness function in section 5.2.

4.2 Generation of ARINC-653 Partition Schedules

As depicted in Fig. 4, the ARINC-653 schedule generator takes input of n scheduling parameter tuples $\langle p_i, b_i \rangle, i = 1, 2, \ldots, n$ and produces a partition schedule $\mathcal{S}$ with the major time frame M. The design of the offline scheduling algorithm should prevent a low-criticality application from affecting high-criticality applications. Hence we adopt the preemptive FP scheduling policy to allocate processor time to partitions. A partition is viewed as a periodic execution unit scheduled in a preemptive fixed priority manner prior to the running of the system. For any partition $\mathcal{P}_i$, the execution budget b_i should be provided during each period p_i. We assign a priority r_i to $\mathcal{P}_i$ and use lower numbers for higher priorities. In practice, the priority of a partition is commonly pre-allocated on the basis of its criticality level. Without loss of generality, We assume that $r_i \leq r_j$ iff $i \leq j$.

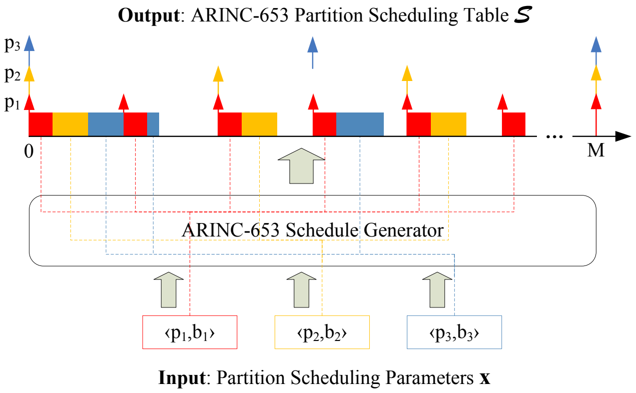

Output: ARINC-653 Partition Scheduling Table $\mathcal{S}$
p3
p2
p1
0
...
M
ARINC-653 Schedule Generator
⟨p1,b1⟩
⟨p2,b2⟩
⟨p3,b3⟩
Input: Partition Scheduling Parameters **x**

Fig. 4 Data flow of an ARINC-653 schedule generator

Algorithm 1 presents the generation process of an ARINC-653 partition schedule. The major time frame M is defined as the least common multiple of all parti-

Algorithm 1 Generation of ARINC-653 partition schedules

```
Input:
    Partition scheduling parameters $\{\langle p_i, b_i\rangle | i = 1, 2, \dots, n\}$
Output:
    Validity of input parameters $SCHED$
    Partition schedule $\mathcal{S} = \{\langle P_t, o_t, d_t\rangle | t = 1, 2, \dots, w\}$
    Major time frame $M$
 1: $SCHED$ :=true
 2: $M := LCM(p_1, p_2, \dots, p_n)$
 3: $\mathcal{S} := \{\langle None, 0, 0\rangle, \langle None, M, 0\rangle\}$
 4: for each partition $\mathcal{P}_i$ from $i = 1$ to $n$ do
 5:   for each period $j$ from $j = 0$ to $M/p_i - 1$ do
 6:     $off := j \times p_i$
 7:     $budg := b_i$
 8:     while $budg > 0$ and $SCHED$ do
 9:       Find two consecutive partition time windows $\langle P_t, o_t, d_t\rangle$ and $\langle P_{t+1}, o_{t+1}, d_{t+1}\rangle \in \mathcal{S}$ where $o_t \leq off < o_{t+1}$
10:       if not found then
11:         $SCHED$ :=false
12:       else
13:         $off := max(off, o_t + d_t)$
14:         $avail := o_{t+1} - off$
15:         Insert a new partition time window of $\mathcal{P}_i$ $\langle \mathcal{P}_i, off, min(avail, budg)\rangle$ into $\mathcal{S}$
16:         $budg := max(budg - avail, 0)$
17:         if $avail > budg$ then
18:           $off := off + budg$
19:         else
20:           $off := o_{t+1} + d_{t+1}$
21:         end if
22:       end if
23:     end while
24:     if $off > (j+1) \times p_i$ then
25:       $SCHED$ :=false
26:     end if
27:   end for
28: end for
29: $\mathcal{S} := \mathcal{S} \setminus \{\langle None, 0, 0\rangle, \langle None, M, 0\rangle\}$
30: return $SCHED, \mathcal{S}, M$
```

tion periods and calculated by the function LCM (line 2). The partition schedule $\mathcal{S}$ is initialized as a set of two auxiliary time slots $\langle None, 0, 0\rangle$ and $\langle None, M, 0\rangle$ that denote the lower and upper bound of partition time windows respectively (line 3). We allocate processor time to partitions from higher priority to lower priority, thus avoiding handling partition preemption. For each partition, we iteratively find gaps between the existing time slots in $\mathcal{S}$ (line 9) and insert new partition time windows into these gaps (line 15).

Algorithm 1 is able to handle any input parameter combinations and offer precise (non-)schedulability conditions (line 10 and 24) at the global level, thereby integrating the parameter sweep optimizer with the Uppaal TA models of ARINC-653 partitioned scheduling systems.

4.3 UPPAAL Template Framework

In the Uppaal template framework, an IMA partitioned scheduling system is modeled as two types of TA: scheduler models and execution models. The TA template of a global scheduler `GS` and a local scheduler `LS` constitute the scheduler models, which control the execution models by using a set of channels as scheduling commands. The execution models consist of two TA templates `PeriodicTask` and `SporadicTask` describing two types of tasks. We present the modeling methods of two major features of partitioned scheduling systems[1].

Two-level Hierarchical Scheduling: The two-level scheduler models `GS` and `LS` realize the hierarchical architecture. Take the local scheduler `LS` shown in Fig.5 for example. A local scheduler belongs to a partition identified by a template parameter `pid`. `LS` receives notification from `GS` through two channels `enter_partition` and `exit_partition` when entering and exiting the partition `pid` respectively, and uses four channels `ready`, `release`, `sched` and `stop` as commands to manage the tasks in `pid`. If there is a task becoming ready to run or relinquishing the processor, the task model will send its `LS` a `ready` or `release` command respectively. `LS` maintains a ready queue `rq` that keeps all the tasks ready and waiting to run, and always allocates the processor to the first task with the highest priority in `rq`. If a new task having a higher priority than any tasks in `rq` get ready, `LS` will insert the task into `rq`, interrupt the currently running task via `stop` and schedule the new selected task via `sched`.

According to whether the current time is inside the partition as well as to the number of the tasks in the ready queue, we create four major locations `NoTask`, `Idle`, `WaitPartition`, and `Occupied`. These four locations cover all situations, where the model must be at one of these locations for any instant. By contrast, the other locations realize conditional branches and atomic action sequences in the model.

Note that this framework has the capability of adopting different local scheduling policies in the system. This can be achieved by instantiating a new template of the local scheduler with a different scheduling policy for the partition. The new template is only required to conform with the same function definition of the channels as before.

Task Behavior: In the templates `PeriodicTask` and `SporadicTask`, we define a set of abstract instructions to describe concrete task behavior. Figure 6 shows the main

[1] A zip file containing the source code for the optimization and all the models can be found at `http://people.cs.aau.dk/~ulrik/submissions/908233/EA_and_models.zip`.

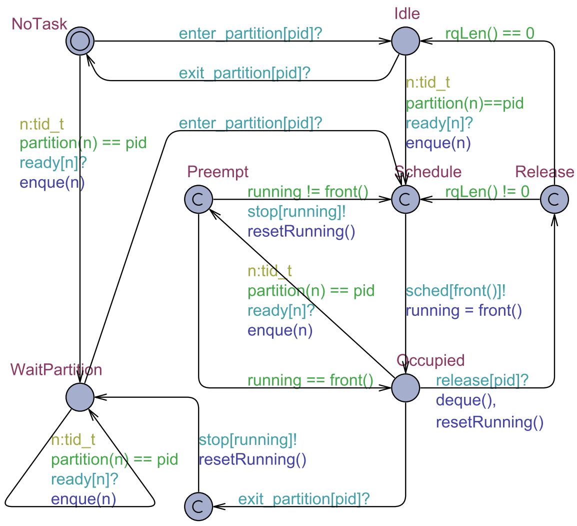


Fig. 5 Local scheduler model

structure of the task templates. A clock `exeTime` measures the processing time during the execution of an abstract instruction, and progresses only when the model is at the location `Running`. Once the task is scheduled by `LS` through the channel `sched`, it will start execution on the processor and move from the location `Ready` to `ReadOp`.

A sequential list of abstract instructions is implemented as the structure array `op`. By using an integer variable `pc` as a program counter, the task can fetch the next abstract instruction from `op[pc]` at the location `ReadOp`. According to the command of this abstract instruction, the task model performs a conditional branch and moves from the location `ReadOp` to one of the different locations that represent different operations.

5 Evolutionary Algorithm EA4HS

Evolutionary algorithms (EA) are an iterative stochastic search method inspired by natural selection and based on the collective learning process within a population of individuals, each of which represents a search point in the solution space of a specific problem [2]. The population evolves from random initial values toward increasingly better solutions by means of three *selection*, *recombination*, and *mutation* operators. The individuals are evaluated and selected according to the value of a *fitness function*. There are several variants of EAs such as Genetic Algorithms (GA), Evolution Strategies (ES), and Evolutionary Programming (EP), which adopt distinctive fitness function, representation of search points, and implementation of operators.

In this section, we present an evolutionary algorithm *EA4HS* for solving the parameter optimization of ARINC-653 hierarchical scheduling systems. This algorithm combines improved operators of the GA and self-adaptation of the ES. We first give the outline of EA4HS. The designs of its fitness function, operators and self-adaptation are then detailed.

5.1 Outline of the Evolutionary Algorithm EA4HS

The goal of EA4HS is to optimize a set of *object parameters* $\mathbf{x} = (x_1, x_2, \ldots, x_m)$, i.e. the unknown $2n$-dimensional vector $\mathbf{x}$ in the optimization problem, regarding an objective function $\mathit{\Omega} : \mathbb{R}^m_+ \to \mathbb{R}$. The EA manipulates *populations* $\beta^{(g)}, g \in \mathbb{N}$ of individuals $\alpha_k^{(g)}$, $k = 1, 2, \ldots, K$ where g is the number of generations and K the size of the population. An *individual* $\alpha_k^{(g)}$ is represented by a tuple $\langle \mathbf{x}_k^{(g)}, \mathbf{s}_k^{(g)} \rangle$ that consists of not only object parameters $\mathbf{x}_k^{(g)} = (x_{k,1}^{(g)}, x_{k,2}^{(g)}, \ldots, x_{k,m}^{(g)})$ but also *strategy parameters* $\mathbf{s}_k^{(g)} = (\sigma_{k,1}^{(g)}, \sigma_{k,2}^{(g)}, \ldots, \sigma_{k,m}^{(g)})$.

The strategy parameters come from evolution strategies to control statistical properties of the genetic operators [6]. These strategy parameters can evolve together with object parameters during the evolution process. For any individual $\alpha_k^{(g)}$, there are $2n$ strategy parameters in $\mathbf{s}_k^{(g)}$ where the evolution of $x_{2i-1,k}^{(g)}$ and $x_{2i,k}^{(g)}$ with $i \in \{1, 2, \ldots, n\}$ (i.e. the unknown parameters p_i and b_i in the optimization problem) is guided by the combination of $\sigma_{2i-1,k}^{(g)}$ and $\sigma_{2i,k}^{(g)}$.

Let I be the range of individuals. The fitness function $f : I \to \mathbb{R}$ realizes the objective function $\mathit{\Omega}$ by mapping each individual to a fitness value. In general, the better an individual fits, the higher is the probability of its being selected in the next generation. Moreover, the EA adopts the mechanism of *elitism* that many of the fittest individuals are copied directly to the next generation, and E is the number of elitist individuals in each generation.

The outline of EA4HS is given in Algorithm 2.

The object parameters in the first population $\beta^{(0)}$ are initialized as a set of independent random numbers from a uniform distribution $\mathcal{U}(x_{imin}, x_{imax})$ where the interval $[x_{imin}, x_{imax}]$ indicates the search range of the optimal solutions. By contrast, all the strategy parameters are set to user-defined values at the first generation according to the definition of the mutation operator of object parameters. Then we evaluate the fitness value of each individual in $\beta^{(0)}$ (line 3). After initialization we enter and execute the main loop of the evolution process until a termination condition is satisfied (lines 4-19).

The main loop produces a descendant population $\beta^{(g+1)}$ from the parent population $\beta^{(g)}$ at any generation g. First, E elitist individuals are copied into the set

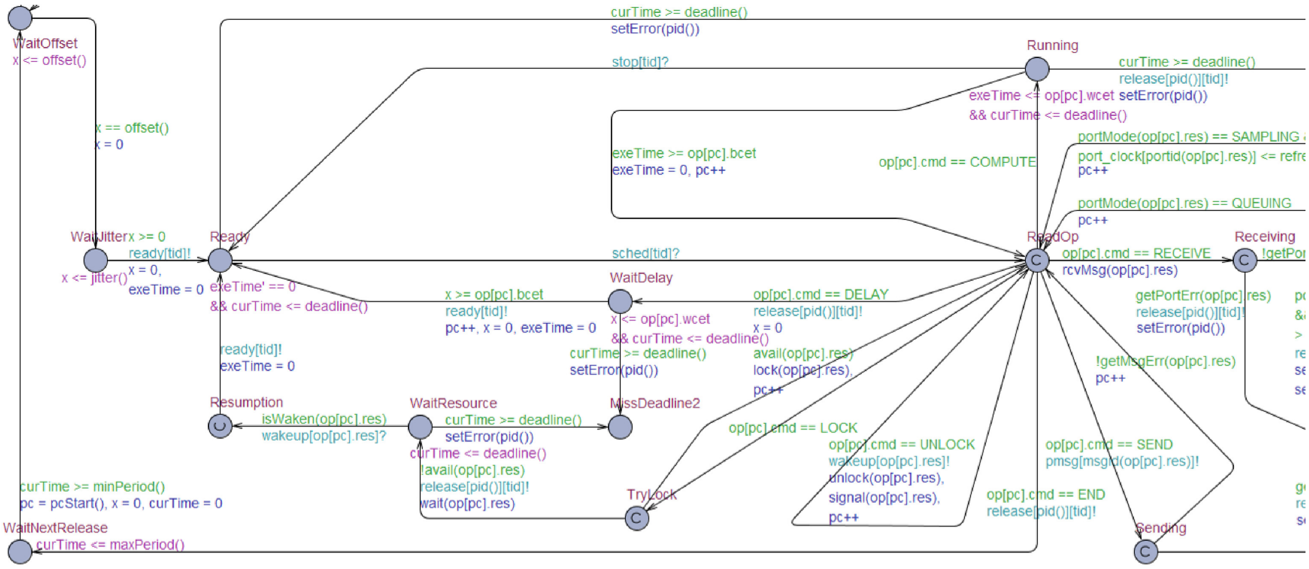


Fig. 6 Main structure of a task model

Algorithm 2 Outline of EA4HS

```
Input:
    Configuration of the evolutionary algorithm
Output:
    Scheduling parameters $\mathbf{x} = (x_1, x_2, \ldots, x_m)$
1:  $g := 0$
2:  initialize $\beta^{(0)} : \{\alpha_k^{(0)} | \alpha_k^{(0)} = \langle \mathbf{x}_k^{(0)}, \mathbf{s}_k^{(0)} \rangle, k = 1, 2, \ldots, K\}$
3:  evaluate $\beta^{(0)} : \{f(\alpha_1^{(0)}), f(\alpha_2^{(0)}), \ldots, f(\alpha_K^{(0)})\}$
4:  repeat
5:      $e := elitist(\beta^{(g)}, E)$
6:      $\beta^{(g)\prime} := selection(\beta^{(g)})$
7:      for $k = 1$ to $K - E$ do
8:          repeat
9:              $\bar{\mathbf{s}} := recombination_s(\beta^{(g)\prime})$
10:             $\bar{\mathbf{x}} := recombination_x(\beta^{(g)\prime})$
11:             $\mathbf{s}_k^{(g+1)} := mutation_s(\bar{\mathbf{s}})$
12:             $\mathbf{x}_k^{(g+1)} := mutation_x(\bar{\mathbf{x}}, \mathbf{s}_k^{(g+1)})$
13:         until $\mathbf{x}_k^{(g+1)}$ is valid or $R_{max}$ iterations are done
14:     end for
15:     $\beta^{(g+1)} := e \bigcup \{\alpha_k^{(g+1)} | \alpha_k^{(g+1)} = \langle \mathbf{x}_k^{(g+1)}, \mathbf{s}_k^{(g+1)} \rangle, k = 1, 2, \ldots, K - E\}$
16:     evaluate $\beta^{(g+1)} : \{f(\alpha_1^{(g+1)}), f(\alpha_2^{(g+1)}), \ldots, f(\alpha_K^{(g+1)})\}$
17:     update $\mathbf{x}$
18:     $g := g + 1$
19: until termination condition
20: return $\mathbf{x}$
```

e (line 5). According to the fitness values of $\beta^{(g)}$, we execute the selection operator that chooses $(K-E)$ pairs of parents separately from the population $\beta^{(g)}$ and writes these parental individuals into the set $\beta^{(g)\prime}$ (line 6). Then the algorithm enters an inner loop (lines 7-14) where a new individual is born during each iteration.

In this inner loop, reproduction should be repeated until a valid object parameter combination is produced or the maximum number R_{max} of iterations is reached (lines 8-13). Otherwise the new generation would be drowning in invalid parameters and starved of information. Based on the selected parental pairs in $\beta^{(g)\prime}$, the recombination and mutation of object parameters are performed (lines 10 and 12), generating the object parameter vector $\mathbf{x}_k^{(g+1)}$ of the kth new offspring. Meanwhile, the strategy parameters originating from $\beta^{(g)}$ also undergo recombination (line 9) and mutation (line 11) independently to control the mutation operator of object parameters that achieves mutative self-adaptation. The resulting object parameters $\mathbf{x}_k^{(g+1)}$ and strategy parameters $\mathbf{s}_k^{(g+1)}$ constitute a new individual $\alpha_k^{(g+1)}$.

We obtain the descendant population $\beta^{(g+1)}$ by composing E elitist individuals e and $(K-E)$ new offspring $\{\alpha_k^{(g+1)}\}$ (line 15). The fitness of $\beta^{(g+1)}$ is evaluated (line 16) to update the current optimal scheduling parameters $\mathbf{x}$ (line 17). Finally, the evolution process returns $\mathbf{x}$ as an optimal solution (line 20).

5.2 Definition of the Fitness Function

The fitness function provides a measure for any individual $\alpha = \langle \mathbf{x}, \mathbf{s} \rangle$ to determine which individuals should have a higher probability of being selected to produce the population at next generation.

The motivation for designing this fitness function stems from two aspects: First, the fitness value should reflect not only the goal of processor occupancy but also the potential for schedulability satisfaction. Such a fitness function evaluates the processor occupancy on the basis of assessment of the schedulability constraints in such a way that we select better individuals without breaching the constraints of the optimization problem. Second, it is necessary to speed up the fitness calculation due to a costly model-based schedulability analysis. An integration of global SMC testing and compositional MC verification should provide a fast strict assessment of schedulability properties for any individual.

Accordingly, the fitness function $f : I \rightarrow \mathbb{R}$ extracts the object parameters $\mathbf{x} = (x_1, x_2, \ldots, x_{2n})$ from their individual α and evaluates the fitness value of $\mathbf{x}$ in accordance with the following principles:

- Invalid parameter combinations, which cannot generate a valid partition schedule, are assigned to the lowest fitness.
- For any valid parameter vector $\mathbf{x}$, the MC verification should not be invoked to confirm the strict schedulability until the entire system of $\mathbf{x}$ is proved statistically schedulable by the SMC tests.
- Higher fitness values should be assigned to statistically schedulable parameter vectors than non-schedulable ones, and to strictly schedulable parameter vectors than only statistically schedulable ones.
- For any valid parameter vector $\mathbf{x}$, if more schedulable partitions are found in the SMC tests or MC verification, a higher fitness should be assigned to $\mathbf{x}$.
- For any two valid parameter vectors, if they are equal in the number of schedulable partitions, we will assign a higher fitness to the vector whose schedulable partitions occupy less processor time.
- For any strictly schedulable parameter vector, a lower processor occupancy U means a higher fitness.

We define the fitness function as the following piecewise formula:

$$f(\alpha) = \begin{cases} -\zeta\Big(1 + \dfrac{\sum_{i=1}^{n} g(x_{2i-1}, x_{2i})}{\sum_{i=1}^{n} x_{2i}}\Big), & \gamma_1 \\ -\zeta\Big(1 - \dfrac{1}{\sum_{i=1}^{n}(x_{2i}/x_{2i-1})}\Big), & \gamma_2 \\ 0, & \gamma_3 \\ \zeta\Big(1 + \sum\limits_{i=1}^{n} \rho(i)(1 - \dfrac{x_{2i}}{x_{2i-1}})\Big), & \gamma_4 \\ \zeta\Big(n + \dfrac{1}{n}(1 + \sum\limits_{i=1}^{n} \varphi(i)(1 - \dfrac{x_{2i}}{x_{2i-1}}))\Big), & \gamma_5 \\ \zeta\Big(1 + n + U_r(\mathbf{x})\Big), & \gamma_6 \end{cases} \tag{6}$$

where the conditions consist of

- $\gamma_1 : \ \exists i, x_{2i-1} < x_{2i}$
- $\gamma_2 : \ \neg\gamma_1 \wedge \sum\limits_{i=1}^{n} \dfrac{x_{2i}}{x_{2i-1}} > 1$
- $\gamma_3 : \ \neg\gamma_1 \wedge \neg\gamma_2 \wedge \neg valid(\mathbf{x})$
- $\gamma_4 : \ valid(\mathbf{x}) \wedge \sum\limits_{i=1}^{n} \rho(i) < n$
- $\gamma_5 : \ valid(\mathbf{x}) \wedge \sum\limits_{i=1}^{n} \rho(i) = n \wedge \sum\limits_{i=1}^{n} \varphi(i) < n$
- $\gamma_6 : \ valid(\mathbf{x}) \wedge \sum\limits_{i=1}^{n} \rho(i) = n \wedge \sum\limits_{i=1}^{n} \varphi(i) = n,$

ζ is a scale factor, $g(p, b) = \begin{cases} b - p, & p < d \\ 0, & p \geq d \end{cases}$ provides the excess budget for the period p and execution budget b, $\rho(i) = \begin{cases} 1, & \text{if SMC query } \rho_i \text{ is satisfied} \\ 0, & \text{if SMC query } \rho_i \text{ is not satisfied} \end{cases}$ returns the results of the SMC schedulability testing, Similarly $\varphi(i) = \begin{cases} 1, & \text{if TCTL property } \varphi_i \text{ is satisfied} \\ 0, & \text{if TCTL property } \varphi_i \text{ is not satisfied} \end{cases}$ provides the results of the compositional MC schedulability verification, $U_r(\mathbf{x})$ gives the remaining processor utilization, and $valid(\mathbf{x})$ fetches the validity of $\mathbf{x}$ after invoking Algorithm 1. The condition $\sum\limits_{i=1}^{n} \rho(i) = n$ and $\sum\limits_{i=1}^{n} \varphi(i) = n$ imply the statistically and strictly schedulability respectively, for all n partitions of the system conclude with positive results.

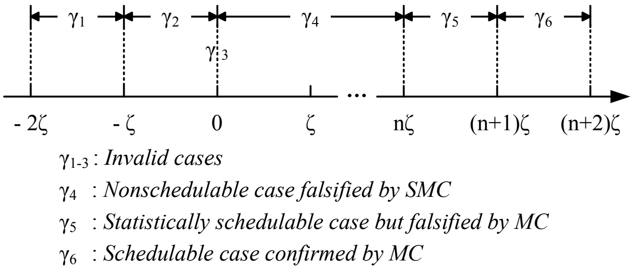

γ1
γ2
γ3
γ4
γ5
γ6
- 2ζ
- ζ
0
ζ
...
nζ
(n+1)ζ
(n+2)ζ
γ1-3 : Invalid cases
γ4 : Nonschedulable case falsified by SMC
γ5 : Statistically schedulable case but falsified by MC
γ6 : Schedulable case confirmed by MC

Fig. 7 Allocation of fitness values on the number axis

There are six cases in the definition of our fitness function. As shown in Fig.7, we allocate different ranges on the number axis to these cases.

The first three cases handle invalid parameter combinations that are indicated by negative or zero fitness values. In the first case γ_1, there exists a partition $\mathcal{P}_i$ whose execution budget $b_i = x_{2i}$ is greater than its period $p_i = x_{2i-1}$. Obviously, such a combination does not make sense. Thus we compute the normalized sum of all the excess budgets and shift it to a low interval $[-2\zeta, -\zeta)$. The second case γ_2, where the total utilization ratio $\sum_{i=1}^{n} x_{2i}/x_{2i-1}$ is greater than 1, overspends all available budgets. Similarly, the excess ratio is mapped into the interval $(-\zeta, 0)$. The rest of invalid parameter vectors should be reported by the ARINC-653 schedule generator due to the non-schedulability at the global level. They are classified as the third case γ_3

and assigned zero fitness. Note that the model-based schedulability testing or verification is not required in these invalid cases.

On the contrary, the fitness of valid object parameters is evaluated on the basis of the results of SMC tests and MC verification. After fast testing the schedulability of each partition in Uppaal SMC, we calculate a fitness value according to the number of statistically schedulable partitions $n_s = \sum_{i=1}^{n} \rho(i)$. The fitness value is mapped into the interval $[n_s\zeta, (n_s+1)\zeta)$ by adding ζn_s and the normalized remaining utilization ratio of statistically schedulable partitions $\zeta(1 - \sum_{i=1}^{n} \rho(i) x_{2i}/x_{2i-1})$ (i.e. case γ_4).

Not until all n partitions go through the SMC tests will the costly compositional MC method be invoked to verify the schedulability of the system. Once this property is confirmed (i.e. case γ_6), the fitness function will extend the remaining processor utilization U_r by an offset $(n+1)\zeta$, thus obtaining the highest fitness within $[(n+1)\zeta, (n+2)\zeta)$. If the schedulability of the system is falsified by the MC verification (i.e. case γ_5), we will map the sum of the number of strict schedulable partitions $\sum_{i=1}^{n} \varphi(i)$ and their remaining utilization ratio $(1-\sum_{i=1}^{n} \varphi(i) x_{2i}/x_{2i-1})$ into the interval $[n\zeta, (n+1)\zeta)$.

5.3 Selection Operator

In the evolution process, there is a high probability of producing low-fitness object parameters such as the invalid combinations where an execution budget is greater than its partition period. Since each generation contains many bad and only very few good individuals, we prefer *exponential ranking selection* operator that is able to give higher selective pressure, i.e. the tendency to select better individuals from a population [32], while guaranteeing certain standard deviation of the fitness distribution of the population after a selection operation [27].

Exponential ranking selection is implemented as two steps: (1) K individuals in a population are ranked in order of fitness from worst 1 to best K. (2) The ith individual is selected according to the exponentially weighted probability

$$p_i = \frac{c^{K-i}}{\sum_{j=1}^{K} c^{K-j}} \tag{7}$$

where the base of exponent $c \in (0,1)$ is used to control the selective pressure of the operator. A smaller c will lead to a higher selective pressure, which means best-fitness individuals are more likely to be selected. The selection operation is repeated until $(K-E)$ pairs of individuals are obtained.

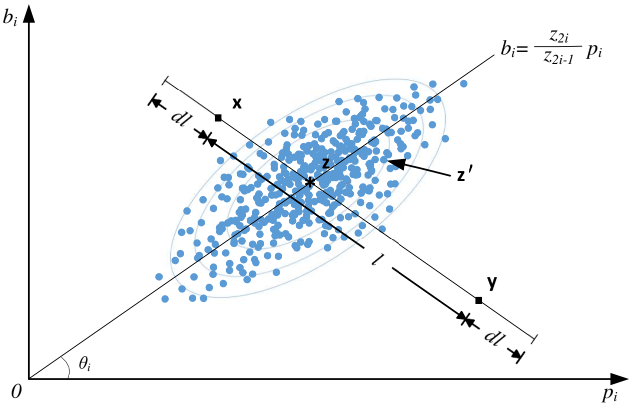


Fig. 8 An example of recombination and mutation operations

5.4 Recombination Operator

There is a widely accepted design principle that recombination operators mainly extract the similarities from selected parents [5]. In our optimization problem, the similarities between individuals originate not only from the independent values of partition periods and budgets but from the processor usage of each partition. Accordingly, we design a *local line recombination* operator for the EA4HS. For any partition $\mathcal{P}_i$, the recombination operator mixes information from parents about the period p_i and budget b_i of $\mathcal{P}_i$, and extracts the similarities in terms of the utilization ratio of b_i to p_i, which indicates the processor usage of $\mathcal{P}_i$.

Let $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_m)$ be the object parameters of two parents. The local line recombination computes an offspring $\mathbf{z} = (z_1, z_2, \ldots, z_m)$ by

$$z_j = x_j + \xi_i(y_j - x_j) \quad j = 1, 2, \ldots, m \quad i = \lfloor \frac{j+1}{2} \rfloor \tag{8}$$

where the weighting ξ_i is randomly generated by a uniform distribution $\mathcal{U}(-d, 1+d)$ and $d \in [0.25, 0.5]$ is the constraint value on the line extension. For any offspring $\mathbf{z}$, two consecutive parameters $p_i = z_{2i-1}$ and $b_i = z_{2i}$ belonging to one partition share a common factor ξ_i. In doing so, the recombination produces the offspring parameters of each partition independently on a common line segment through both of the parents.

We consider three types of genetic information: (1) the period p_i, (2) the budget b_i, and (3) the utilization ratio b_i/p_i of the ith partition. As depicted in Fig.8, the offspring $\mathbf{z}$ can be chosen uniformly at random from the line $\mathbf{xy}$, where the recombination operator mixes these three types of genetic information simultaneously from parents. Obviously, all three types of genetic information are kept in the offspring $\mathbf{z}$ and similar to those in its parents $\mathbf{x}$ and $\mathbf{y}$.

5.5 Mutation Operator

Compared with recombination, mutation operators do not only provide a source of genetic variation but also maintain degree of population diversity, whose insufficiency is one of the major cause of premature convergence [31]. However, generic mutation operators cannot utilize the correlations between the period p_i and budget b_i in individuals to acquire promising processor usage of partitions, causing the mutants to be always eliminated after selection in all probability. This extremely low survival rate increases the risk of premature convergence. Thus we propose a *rotated Gaussian mutation* operator to help the EA4HS converge to a global optimum effectively.

The mutation operator has two input parameters including the set of object parameters $\mathbf{z} = (z_1, z_2, \ldots, z_m)$ after recombination and of strategy parameters $\mathbf{s} = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ that control mutation strength. Each pair of the object parameters (z_{2i-1}, z_{2i}) is mutated as an independent vector $\tilde{z}_i$. The mutation operator transforms $\mathbf{z}$ into a new offspring $\mathbf{z}' = (z'_1, z'_2, \ldots, z'_m)$. Let $\tilde{z}'_i$ stand for (z'_{2i-1}, z'_{2i}). We have

$$\tilde{z}'_i = \tilde{z}_i + \varDelta_i \quad i = 1, 2, \ldots, n \tag{9}$$

where $\varDelta_i$ is a random sample from a bivariate normal distribution $\mathcal{N}(\boldsymbol{\mu}_i, \mathbf{\Sigma}_i)$.

The covariance matrix $\mathbf{\Sigma}_i \in \mathbb{R}^{2\times 2}$ can be geometrically interpreted as a set of ellipses, each of which is a density contour of $\mathcal{N}(\boldsymbol{\mu}_i, \mathbf{\Sigma}_i)$. Consider the fact that a parent $\mathbf{z}$ with valid parameters has a high probability of producing a valid offspring $\mathbf{z}'$ if each of the new utilization ratios z'_{2i}/z'_{2i-1} is close to the parental z_{2i}/z_{2i-1}. We define the covariance matrix $\mathbf{\Sigma}_i$ as the set of ellipses whose major axes are parallel with the lines of equal ratio z_{2i}/z_{2i-1} shown in Fig.8. Thus $\mathcal{N}(\boldsymbol{\mu}_i, \mathbf{\Sigma}_i)$ is obtained by rotating a bivariate normal distribution $\mathcal{N}(\boldsymbol{\mu}_i, \mathbf{D}_i)$ counterclockwise through an angle θ_i:

$$\mathbf{\Sigma}_i = \mathbf{R}_i \mathbf{D}_i \mathbf{R}_i^T \tag{10}$$

where $\mathbf{D}_i = diag(\sigma_{2i-1}, \sigma_{2i})$ derives two strategy parameters σ_{2i-1} and σ_{2i} from $\mathbf{s}$, $\mathbf{R}_i = \begin{pmatrix} cos\theta_i & -sin\theta_i \\ sin\theta_i & cos\theta_i \end{pmatrix}$ is a rotation matrix, and

$$sin\theta_i = \frac{z_{2i}}{\sqrt{z_{2i-1}^2 + z_{2i}^2}}, \; cos\theta_i = \frac{z_{2i-1}}{\sqrt{z_{2i-1}^2 + z_{2i}^2}}. \tag{11}$$

Strategy parameters σ_{2i-1} and σ_{2i} indicate the standard deviations of $\mathcal{N}(\boldsymbol{\mu}_i, \mathbf{\Sigma}_i)$ along the major and minor axes respectively. To perform such an ellipses-parallel mutation, we initialize each σ_{2i-1} of the strategy parameters with a greater value than σ_{2i}, thereby adapting the mutation distribution $\varDelta_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \mathbf{\Sigma}_i)$ to the fitness landscape.

The mean $\boldsymbol{\mu}_i \in \mathbb{R}^2$ of the normal distribution is defined as

$$\boldsymbol{\mu}_i = \begin{cases} \mathbf{0}, & \|\tilde{z}_i\|_2 \geq 2\sigma_{2i-1} \\ 2\sigma_{2i-1}(cos\theta_i, sin\theta_i), & \|\tilde{z}_i\|_2 < 2\sigma_{2i-1} \end{cases} \tag{12}$$

where $\|\tilde{z}_i\|_2 = \sqrt{z_{2i-1}^2 + z_{2i}^2}$ is the Euclidean norm of the vector $\tilde{z}_i$. In most cases, the mean $\boldsymbol{\mu}_i$ is assigned $\mathbf{0}$ and hence the mutants $\mathbf{z}'$ will center around the input parameters $\mathbf{z}$. However, the zero mean $\boldsymbol{\mu}_i = \mathbf{0}$ may cause the mutations to generate a large number of invalid minus parameters, especially when the input points $\tilde{z}_i$ are close to the origin but their mutations receive large standard deviations σ_{2i-1}. According to the empirical rule in statistics (i.e. 95% of the values in a normal distribution lie within two standard deviations of the mean), we will add a $2\sigma_{2i-1}$ offset along the major axis of $\mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_i)$ to $\tilde{z}_i$ if the Euclidean norm of $\tilde{z}_i$ is less than a distance of $2\sigma_{2i-1}$, effectively reducing the probability of producing minus parameters.

Subsequently, the EA4HS sets strategy parameters adaptively to direct the search during the evolution process.

5.6 Self-adaptation of Strategy Parameters

The strategy parameters are encoded, selected and inherited together with the object parameters of individuals. They also undergo recombination and mutation operations to control the statistical properties of the mutation operator of object parameters adaptively.

Since the considerable fluctuations of strategy parameters normally degrade the performance of EAs [6], we provide a *weighted intermediate recombination* operator for strategy parameters in order to mitigate these fluctuations as well as extract the similarities. The recombinant $\bar{\mathbf{s}}$ is a weighted average of all the K vectors $\mathbf{s}_k^{(g)} = (\sigma_{k,1}^{(g)}, \sigma_{k,2}^{(g)}, \ldots, \sigma_{k,m}^{(g)})$ of the strategy parameters in a population $\beta^{(g)} = \{\alpha_1^{(g)}, \alpha_2^{(g)}, \ldots, \alpha_K^{(g)}\}$:

$$\bar{\mathbf{s}} = \frac{1}{K} \sum_{k=1}^{K} (1 - \tau_r)^{\lambda_k} \mathbf{s}_k^{(g)} \tag{13}$$

where $\tau_r \in [0, 1]$ is a user-defined learning rate and λ_k is the number of times an individual $\alpha_k^{(g)}$ appears continuously in the elitist set ε. The recombination also assigns $\bar{\mathbf{s}}$ to all the individuals in ε.

A *log-normal* operator [6] is applied to the mutation of strategy parameters, providing the primary source of their genetic variation. This log-normal mutation ensures positiveness of the strategy parameters that serve as standard deviation of a normal distribution. The recombinant $\bar{\mathbf{s}} = (\bar{\sigma}_1, \bar{\sigma}_2, \ldots, \bar{\sigma}_m)$ mutates into the strategy parameters $\mathbf{s}_k^{(g+1)} = (\sigma_{k,1}^{(g+1)}, \sigma_{k,2}^{(g+1)}, \ldots, \sigma_{k,m}^{(g+1)})$ at next generation by

$$\sigma_{k,j}^{(g+1)} = \bar{\sigma}_j e^{\tau_u \mathcal{N}_j(0,1)} \tag{14}$$

where τ_u is also an input learning rate and $\mathcal{N}_j(0,1)$ denotes a random sample from the standard normal distribution. The learning rate $\tau_u = 1/\sqrt{m}$ is recommended according to [6].

6 Experiments

This section presents the experiments on two avionics systems to demonstrate the applicability of our optimization method. In the experiments, our parameter sweep method shows the capability of converging to a global optimum. We also evaluate the performance of search algorithms by comparing the proposed EA4HS with exhaustive search and two popular genetic algorithms.

All the experiments in this section were executed on the cluster that consists of 4 computer nodes with 1 TB memory. Each node has 64 cores of 4 AMD Opteron 6376 processors. The schedulability tests and validation were performed on Uppaal 4.1.19 64-bit version. We assign the timebound $N = 1.0 \times 10^4$ time units and the probability threshold $\theta = 0.05$ for Eq.(2).

6.1 Experiment on Simple Periodic Task Sets

We first perform the experiments on a simple periodic task set taken from [13]. The task set comprises two identical partitions with different priorities, thus making their partition priority ordering irrelevant. Each partition contains multiple independent periodic tasks, whose period, deadline and priority are encoded into Uppaal declarations. The behavior of a task is described as a pure *Compute* instruction with a Worst Case Execution Time (WCET).

The task set is shown in Table 1 where the column “PID” and “TID” identify the partitions and tasks respectively, “PR” gives partition priorities, “T” is task periods, “E” is the WCET, “D” is deadline, and “R” is task priorities. We define the time unit as a microsecond in the table and the context switch overhead as 2 time units.

Table 1 Task set of Experiment 1 [13] (Times in μs)

PID	PR	Task				
		TID	T	E	D	R
$\mathcal{P}_1$	1	Tsk_1^1	160	8	100	1
		Tsk_2^1	240	12	200	2
		Tsk_3^1	320	16	300	3
		Tsk_4^1	480	24	400	4
$\mathcal{P}_2$	2	Tsk_1^2	160	8	100	1
		Tsk_2^2	240	12	200	2
		Tsk_3^2	320	16	300	3
		Tsk_4^2	480	24	400	4

Experiment 1

The parameter optimization of the above avionics workload was carried out by three following methods:

– *Exhaustive search:* An analytical condition for schedulable parameters is derived from a response time analysis [13]. This method scans all possible integer combinations of the partition period through a potential interval $[4, 200]$. For each period combination, it uses a binary search together with the schedulability condition to find the minimum execution budget from the highest priority to the lowest one. This exhaustive search is able to produce a global optimal solution, but only applicable to such a simple system.

– *Parameter sweep with GAs:* Two popular GAs, the classic and the breeder genetic algorithm [26], are first applied to parameter sweep for comparison. Table 2 shows their operator combinations and denotes them by “GA 1” and “GA 2” respectively. Individuals are binary encoded in both of the GAs. In GA 1, an exchange of each bit in parents takes place with a probability $p_e = 0.5$, and the probability of bit mutation is $p_u = 0.2$. In GA 2, the percentage $T\%$ of truncation selection is set to 50%, the weighting constraint of intermediate recombination is $d_r = 0.5$, the standard deviation of Gaussian mutation is $\sigma_u = 10$, and the mutation probability is $p_u = 0.2$.

– *Parameter sweep with EA4HS:* In the EA4HS, individuals are also binary encoded, the base c of exponential ranking selection is 0.8, the value d of local line recombination is 0.5, and we initialize the strategy parameters $\sigma_{2i-1} = 50$ and $\sigma_{2i} = 5$ for $i \in \{1, 2\}$ in an individual. We define the learning rates $\tau_r = 0.7$ and $\tau_u = 1/\sqrt{4} = 0.5$. The operator combination is also shown in Table 2.

Both the GAs and EA4HS adopt the search range $[4, 200]$ for all partition periods, population size $K = 64$, elitism size $E = 4$, and maximum generation $G = 300$. For each

Table 2 Operator combinations of the EAs

EA	Selection operator	Recombination operator	Mutation operator
GA 1	Roulette wheel selection	Uniform crossover	Bit-flip mutation
GA 2	Truncation selection	Intermediate recombination	Gaussian mutation
EA4HS	Exponential ranking selection	Local-line recombination	Rotated Gaussian mutation

individual, we calculate its fitness value and store them in a hash table. Once the same individual reappears in the following generations, the fitness value will be fetched from the hash table directly, thus avoiding the costly redundant fitness calculation.

Table 3 shows the optimization result of Experiment 1. Both the exhaustive search and parameter sweep with the EA4HS reached the same global optimal solution $\mathbf{x}_{opt} = (160, 34, 160, 34)$, which gives a minimum processor occupancy $U = 45\%$. Unfortunately, two GAs only offer two local optimal solutions with much higher processor occupancy 61.67% and 58.73%.

Table 3 Optimization result of Experiment 1 (Times in μs)

Method	Solution	Occupancy	Optimal
GA 1	$(120, 26, 180, 66)$	61.67%	No
GA 2	$(126, 35, 126, 35)$	58.73%	No
EA4HS	$(160, 34, 160, 34)$	45%	Yes

Figure 9 presents the evolution of minimum processor occupancy and cumulative processing time of the GAs and EA4HS in Experiment 1. Since duplicate fitness calculation is replaced with reading the hash table, the convergence of the evolution means a synchronous slowdown in the variation of minimum processor occupancy and cumulative processing time. Obviously, both GA 1 and GA 2 fell into a premature convergence on local optimal solutions after 30 generations.

In contrast, the EA4HS adjusts strategy parameters adaptively to control the average search area of mutation operations. When there was a convergence trend during the generations of $[30, 70)$ and $[160, 200)$, the self-adaptation of strategy parameters expanded the search areas to improve population diversity and subsequently made the search concentrated in smaller areas to find better solutions within few generations, thereby leading to a fast decrease in the minimum processor occupancy over the subsequent generations $[100, 160)$ and $[270, 300)$ shown in Fig. 9(a). The repeated adjustments of the EA4HS reduce the risk of premature convergence on a local optimal area, producing two “steps” of its processing time curve in Fig. 9(b).

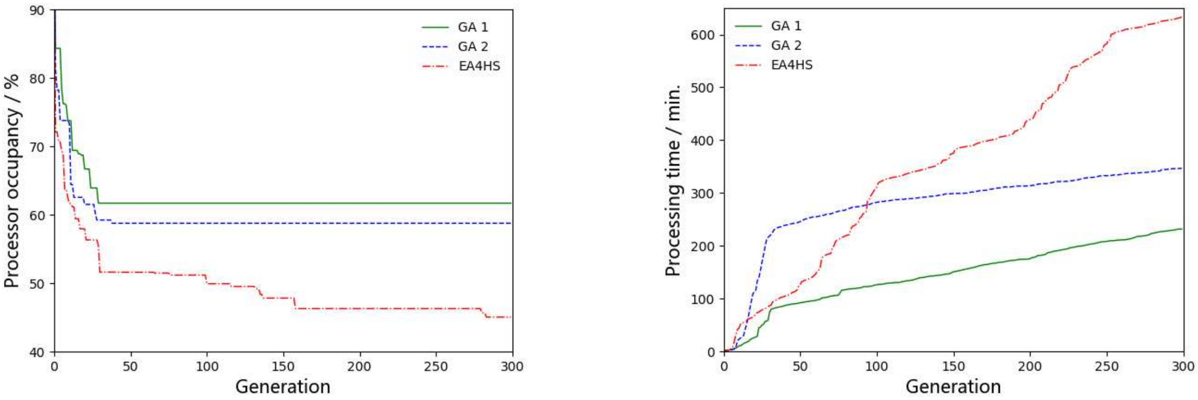
Experiment 2

Considering that the premature convergence may affect the result of these two GAs, we continue the comparison experiment on the same task set but adopt different configuration for the EAs to defer their convergence during the evolution. Experiment 2 repeats the same procedure for Experiment 1, using the same search range $[4, 200]$, population size $K = 64$, elitism size $E = 4$, maximum generation $G = 300$, and the following detailed configuration:

- *GA 1:* More frequent bit-flip mutation in GA 1 will produce new individuals more randomly, thus possibly raising the degree of population diversity to prevent premature convergence. Hence we keep the bit-exchange probability $p_e = 0.5$ but use a double bit-mutation probability $p_u = 0.4$.
- *GA 2:* We increase both the probability and strength of the variable mutation to delay the convergence in GA 2, using the new standard deviation $\sigma_u = 20$ of Gaussian mutation and its larger mutation probability $p_u = 0.4$. We still keep the percentage $T = 50\%$ of truncation selection and the weighting constraint $d_r = 0.5$ of intermediate recombination.
- *EA4HS:* In the EA4HS, a lower learning rate τ_r will slow down the convergence by adjusting the average strategy parameters of populations. Hence we invoke the EA4HS with a smaller learning rate $\tau_r = 0.4$ and retain the other configuration including the base $c = 0.8$ of exponential ranking selection, the weighting constraint $d = 0.5$ of local line recombination, the learning rate $\tau_u = 0.5$, and the initial strategy parameters $\sigma_{2i-1} = 50$ and $\sigma_{2i} = 5, i \in \{1, 2\}$.

In Table 4, the results of Experiment 2 show that our EA4HS gets accustomed to this new configuration, for the algorithm acquired the global optimal solution $\mathbf{x}_{opt} = (160, 34, 160, 34)$ with the minimum processor occupancy $U = 45\%$. Unfortunately, two GAs still deviated from the global optima, but they generated better solutions compared with their optimization result in Experiment 1.

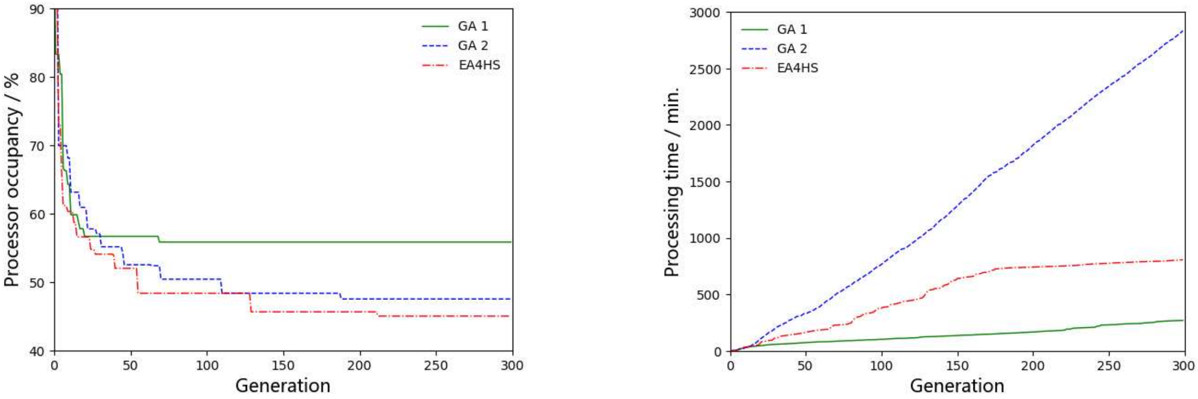
Figure 10 depicts the evolution of minimum processor occupancy and cumulative processing time in Experiment 2. Although GA 1 obtained a lower processor occupancy than that in Experiment 1, it fell into



(a) Minimum processor occupancy

(b) Cumulative processing time

Fig. 9 Evolution of minimum processor occupancy and cumulative processing time in Experiment 1



(a) Minimum processor occupancy

(b) Cumulative processing time

Fig. 10 Evolution of minimum processor occupancy and cumulative processing time in Experiment 2

Table 4 Optimization result of Experiment 2 (Times in μs)

Method	Solution	Occupancy	Optimal
GA 1	$(60, 14, 120, 33)$	55.83%	No
GA 2	$(120, 27, 120, 26)$	47.5%	No
EA4HS	$(160, 34, 160, 34)$	45%	Yes

the premature convergence again at around the 30th generation. The new configuration of GA 2 successfully avoided its convergence to find more better individuals but significantly increased the processing time from $347min$ to $2833min$.

Compared with the configuration of EA4HS in Experiment 1, a lower learning rate $\tau_r = 0.4$ avoids a sharp drop in strategy parameters and frequent adjustments during the evolution. Thus the EA4HS generated smoother curves of minimum processor occupancy (Fig. 10(a)) and cumulative processing time (Fig. 10(b)) in Experiment 2. The evolution had not entered a convergence until it found a nearly optimal solution at around the 200th generation, finally reaching the global optimum at the 212nd generation.

The experiments reveal distinct superiority of our EA4HS over the GAs. First, GA 1 is not applicable to this optimization problem. We find the bit-based reproduction of GA 1 cannot produce more better individuals steadily, making GA 1 very prone to premature convergence. By contrast, GA 2 can overcome the problem of premature convergence by adjusting the mutation configuration. However, GA 2 has a low search efficiency. In both of the experiments, GA 2 concentrate search on the local optimal area where all the periods are centered around $120\mu s$ and far from the best $160\mu s$. For this purpose, our EA4HS is provided with new recombination and mutation operator which can produce descendant individuals on the basis of the processor usage of parents. Hence it is more likely to climb up a higher processor occupancy even if our current individuals are far from the optimal area.

Table 5 Workload of the avionics system [11,16](Times in milliseconds)

No.	Task	Release	Offset	Jitter	Deadline	Priority	Execution Chunks			
							Time	Mutex	Output	Input
P_1	Tsk_1^1	[25,25]	2	0	25	2	[0.8,1.3] [0.1,0.2]	- -	- -	- -
	Tsk_2^1	[50,50]	3	0	50	3	[0.2,0.4]	-	Msg_1	-
	Tsk_3^1	[50,50]	3	0	50	4	[2.7,4.2]	-	-	-
	Tsk_4^1	[50,50]	0	0	50	5	[0.1,0.2]	Mux_1^1	-	-
	Tsk_5^1	[120,∞)	0	0	120	6	[0.6,0.9] [0.1,0.2]	- Mux_1^1	- -	- -
P_2	Tsk_1^2	[50,50]	0	0.5	50	2	[1.9,3.0]	-	-	-
	Tsk_2^2	[50,50]	2	0	50	3	[0.7,1.1]	-	Msg_2	-
	Tsk_3^2	[100,100]	0	0	100	4	[0.1,0.2]	Mux_1^2	-	-
	Tsk_4^2	[100,∞)	10	0	100	5	[0.8,1.3] [0.2,0.3]	- Mux_1^2	- -	- -
P_3	Tsk_1^3	[25,25]	0	0.5	25	2	[0.5,0.8]	-	-	Msg_1
	Tsk_2^3	[50,50]	0	0	50	3	[0.7,1.1]	-	-	Msg_2
	Tsk_3^3	[50,50]	0	0	50	4	[1.0,1.6]	-	-	Msg_3
	Tsk_4^3	[100,∞)	11	0	100	5	[0.7,1.0] [0.1,0.3]	- -	- -	- -
P_4	Tsk_1^4	[25,25]	3	0.2	25	2	[0.7,1.2]	-	-	-
	Tsk_2^4	[50,50]	5	0	50	3	[1.2,1.9]	-	Msg_3	Msg_1
	Tsk_3^4	[50,50]	25	0	50	4	[0.1,0.2]	-	-	Msg_4
	Tsk_4^4	[100,100]	11	0	100	5	[0.7,1.1]	-	-	-
	Tsk_5^4	[200,200]	13	0	200	6	[3.7,5.8]	-	-	-
P_5	Tsk_1^5	[50,50]	0	0.3	50	1	[0.7,1.1]	-	-	Msg_1
	Tsk_2^5	[50,50]	2	0	50	2	[1.2,1.9]	-	Msg_4	Msg_2
	Tsk_3^5	[200,200]	0	0	200	3	[0.4,0.6] [0.2,0.3]	- Mux_1^5	- -	- -
	Tsk_4^5	[200,∞)	14	0	200	4	[1.4,2.2] [0.1,0.2]	- Mux_1^5	- -	- -

6.2 Experiment on a Concrete Avionics System

We undertake the third experiment on a much larger and more complex IMA partitioned scheduling system including multiple task types, task dependency, and inter-partition communication [11,16]. As shown in Table 5, the system consists of 5 partitions that contain a total of 18 periodic tasks and 4 sporadic tasks. The type of a task depends on its *release* interval. A periodic task has a fixed period, whereas a sporadic task satisfies a minimum separation between consecutive release. The execution of a task is characterized as a sequence of *chunks*. Each chunk has a lower and upper bound on *execution time*, a set of potentially required resources and message-passing operations. There are 3 intra-partition locks, as shown in column *mutex*, and 4 inter-partition message types in the task set. The columns *output* and *input* indicate transfer direction of messages. According to the resources required by chunks, we convert each chunk into a subsequence of the abstraction instruction sequence (*Receive*, *Lock*, *Compute*, *Unlock*, *Send*, *End*) in the UPPAAL execution models. We assume the context switch overhead to be 0.2 milliseconds in the experiment.

In this IMA system, the features such as task dependency and communication render the analytical bounds in [13] non-applicable. Moreover, the immense parameter space makes it impossible to complete a brute-force search for a global optimal solution. Hence Experiment 3 compares the EA4HS optimization with the empirical scheduling scheme given in [11]. Their detailed configuration is listed as follows:

- *Empirical scheduling:* In this scheme, all the partitions have a unique period $p = 25ms$, which is the minimum and a harmonic of task periods. Each partition is allocated to a time slot of the same length $5ms$ within every partition period. We first create an ARINC-653 partition schedule according to this empirical scheme. The common partition period is used as the major time frame $M = 25ms$. Within every M, the five time slots of the partitions are arranged in order of priority. A context switch overhead $0.2ms$ is inserted into the start of the time slots, each of which thus shrinks to the size of $4.8ms$. Subsequently, we analyze the schedulability of this empirical scheme by using the compositional approach given in Section 3.
- *EA4HS:* Considering the larger parameter space and longer processing time for each generation, we apply a new population size $K = 256$, elitism size $E = 16$ and maximum generation $G = 200$ but keep the rest of the configuration of Experiment 2, which has been proved applicable to the simple system with similar quantities of time. We set the base of exponential ranking selection $c = 0.8$ and the value of local line recombination $d = 0.5$. In the first generation, the strategy parameters of an individual are initialized as $\sigma_{2i-1} = 50$ and $\sigma_{2i} = 5$ for $i \in \{1, 2, \ldots, 5\}$. We define the learning rates $\tau_r = 0.4$ and $\tau_u = 1/\sqrt{10} \approx 0.3$.

Table 6 presents the optimization results of Experiment 3. Owing to the much larger unknown parameter space, it is more difficult to find a schedulable solution than Experiment 1 and 2. The empirical scheme even failed to conclude with a schedulable solution. In the schedulability analysis, a counterexample generated by UPPAAL demonstrates that Tsk_{3}^{1} misses its deadline at the instant $t = 50.2ms$. Although the original $5ms$ allocation is sufficient for the execution of Tsk_{3}^{1}, the additional overhead of context switches makes Tsk_{3}^{1} go over budget. Moreover, this empirical scheme takes up all the processor time, increasing the integration cost of additional avionics workload.

In contrast to the unsatisfactory results of the empirical method, our EA4HS acquired a schedulable solution $\mathbf{x} = (25, 4.9, 25, 4.7, 25, 3.4, 25, 4.5, 50, 4.5)$ with a lower processor occupancy 82.6%. Its schedulability is not only tested statistically by UPPAAL SMC but also validated rigorously by UPPAAL classic MC. Even though its global optimality cannot be confirmed, engineers can still benefit from such schedulable results that have acceptable processor occupancy.

Figure 11(a) illustrates the best fitness value and processing time of each generation in the EA4HS optimization. The fitness value offers the quality evaluation of any parameter combination regardless of its schedulability. According to the definition of fitness function in section 5.2, the coordinate plane can be divided into three areas that correspond to different fitness intervals: (1) $[0, 500)$ where the generations contain no schedulable solution, for all the individuals are fast falsified by SMC. (2) $[500, 600)$ where all five partitions of the best individual are proved statistically schedulable by SMC but its schedulability is finally excluded by MC. (3) $[600, 700)$ where the schedulability of the best individual is strictly confirmed by MC.

As shown in Fig. 11(a), there was no schedulable individual in the initial population. During the generations of $[1, 105)$, the best fitness value and processing time increased gradually as more partitions of individuals were proved statistically schedulable by SMC. At the 105th generation, we found the first statistically schedulable individual with the fitness value 501 and started the MC compositional analyses. Although its schedulability was excluded by MC, there were a growing number of higher-fitness individuals that went through the SMC tests at the following generations. Since most of the MC compositional analyses were much more time-consuming than the SMC tests, the average processing time for each generation rose from around $10min$ to more than $40min$ after 105 generations. Finally, we acquired the first schedulable individual at the 161st generation and found the best solution with the lowest processor occupancy 82.6% at the 179th generation within the cumulative time of 62 hours.

Figure 11(b) shows the composition of populations during the evolution. A population consists of the following four types of individuals: (1) Invalid individuals that cannot generate ARINC-653 schedules. (2) SMC falsified individuals that turned out to be non-schedulable in the SMC tests. (3) MC falsified individuals that were proved statistically schedulable by SMC but eliminated in the MC compositional analyses. (4) schedulable individuals.

The evolution of the population composition demonstrates improvements in the efficiency of our EA4HS optimization. First, the EA4HS avoids the populations drowning in invalid individuals via repeated reproduction (lines 8-13 of Alg. 2). As shown in Fig. 11(b), invalid individuals accounted for a quarter of the initial population that was generated randomly, but the EA4HS kept their proportion falling sharply until they vanished after the 17th generation. Second, the application of SMC fast falsification speeds up the optimization. For each population in Fig. 11(b), most of the individuals underwent the SMC tests rather than the costly MC compositional analyses. Third, the EA4HS adaptively keeps a steady growth in the number of higher-

Table 6 Optimization result of Experiment 3 (Times in milliseconds)

Method	Solution	Schedulability	Occupancy	Optimal
Empirical	$(25, 4.8, 25, 4.8, 25, 4.8, 25, 4.8, 25, 4.8)$	No	100%	No
EA4HS	$(25, 4.9, 25, 4.7, 25, 3.4, 25, 4.5, 50, 4.5)$	Yes	82.6%	Unknown

fitness individuals but does not concentrate rapidly on the localities of dominant solutions, thus reducing the risk of premature convergence. In Fig. 11(b), the proportion of schedulable individuals increased gradually until they filled the elitist list. At the following generations, the newly-produced individuals did not converge on a few dominant solutions but maintained a degree of population diversity. Finally, there was a steady proportion of schedulable individuals in the populations at around the 200th generation.

7 Related Work

A few approaches to optimizing the partition scheduling of avionics systems have been presented in the literature, applying analytical or formal methods from either a global or compositional viewpoint on the hierarchical scheduling architecture.

Compositional analytical methods introducing the abstraction and composition of constituent partitions optimize each partition locally for the whole system. The authors of [29,15,30] adopted different resource models to characterize the time demand and supply of partitions, presented the schedulability conditions under EDF (Earliest Deadline First) and RM (Rate Monotonic) policy, and gave utilization bounds of these resource models. In [16], they extended this compositional framework into ARINC-653 avionics systems, providing a task model to deal with the behaviors like communication latencies and blocking/preemption overheads within partitions. In [24], the authors proposed a similar analytical method for applications consisting of periodic or sporadic tasks scheduled by FP policy to find the best scheduling parameter pairs of partitions. To improve the runtime performance, Dewan and Fisher [14] proposed a polynomial-time approximation algorithm for minimizing the interface bandwidth of sporadic task systems.

However, the combination of local optimality of each partition does not necessarily lead to the globally optimal solution, because the parameters chosen for one partition may affect the choice for other partitions [13]. From a global viewpoint, Davis and Burns [13] formulated the optimization problem as a holistic selection of partition parameters, providing a set of search algorithms to find the best parameter combination. Nevertheless, the optimal solution can only be determined by an exhaustive search in the case of small systems. Yoon et al. [34] showed the non-convexity of multiple partition optimization and solved this non-linear non-convex problem with Geometric Programming (GP). Kim et al. [22] formulated a linear programming problem for the utilization bound of a schedulable periodic task set scheduled by RM policy in a given ARINC-653 partition. Blikstad et al. [7] simplified the two-level hierarchical scheduling into the pre-runtime scheduling of non-overlapping periodic tasks, adopting a Mixed Integer Programming (MIP) formulation to generate the optimal schedule.

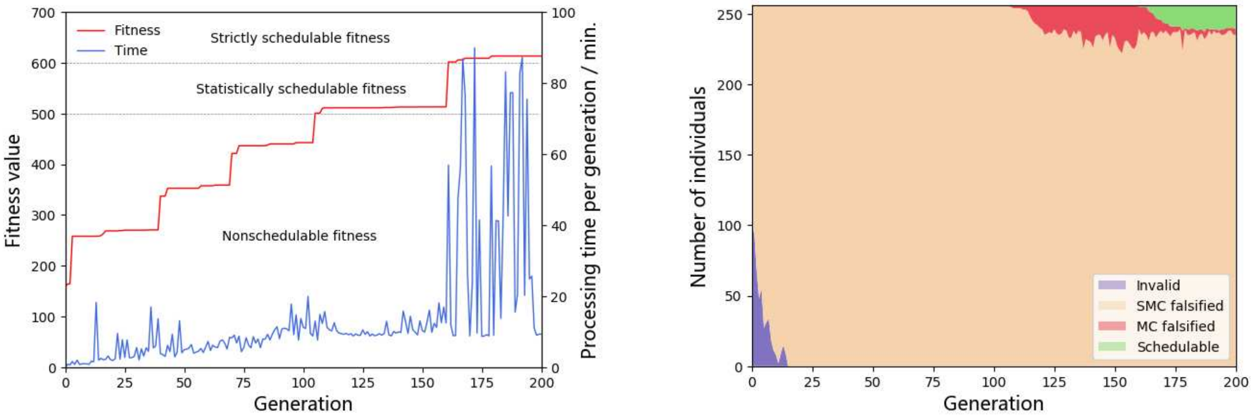
Unfortunately, both of the analytical methods introduce a certain degree of pessimism due to the oversimplification of their optimization policies or system models:

The pessimism of the *compositional methods* mainly originates from the oversimplification of optimization policies. The compositional methods employ the "Divide and Conquer" strategy, finding the optimal solution for each partition independently and assembling all the local results as a complete solution of the system. For each partition, such solvers always search for its optimal solution on the worst-case assumption of the rest of the partitions. This policy reduces the complexity of optimization solving but ignores the possible coordination between partitions. Hence the final solution of an ARINC-653 partitioned system is only the combination of local optima and not globally optimal.

The pessimism of the *global optimization methods* is introduced by the oversimplification of system models. Since the nature of ARINC-653 partition scheduling is a complex non-linear non-convex optimization problem [34], these global methods simplify the system models by linearizing the analytical equations of the schedulability constraints and formulate it as a classical optimization problem like MIP. This simplification leads to a conservative solution of partition scheduling. The degree of pessimism depends on the approximation precision of the simplified system model.

By contrast, model-based methods provide rigorous formal models to describe more concrete behaviors of avionics systems in a readable and understandable way.

Beji et al. [4] expressed the constraints of distributed IMA architecture as SMT (Satisfiability Modulo Theory) logic formulas and used the SMT solver YICES to



(a) Best fitness values and processing time

(b) Composition of populations

Fig. 11 Evolution of best fitness values, processing time and population composition in Experiment 3

find automatically feasible scheduling parameters that minimize the integration cost. However, modeling a detailed system requires a large number of lengthy logic formulas. Their SMT model only covering a flat rather than hierarchical structured partition scheduling system did not give any specification of the tasks in partitions.

Sun et al. [33] proposed a component-based schedulability analysis of hierarchical scheduling systems encoded into linear hybrid automata, thus enabling the optimization of partition parameters. All the integer values of partition parameters were exhaustively tested for schedulability to minimize the processor utilization. Obviously, this exhaustive search is not feasible for a large high-dimensional parameter space.

The authors of [23, 8] applied Uppaal to the compositional optimization of partition parameters. Given a specific partition, they used a lightweight SMC method for a fast design exploration of objective parameters, assuring the schedulability of the corresponding TA models with a high confidence. Once a promising parameter tuple had been found, it could be proved schedulable using the costly MC method. This approach mitigating the state-space explosion coped with each partition up to 6 tasks. However, as remarked above, this compositional optimization does not necessarily lead to the globally optimal solution. Moreover, they also ignored concrete task behaviors in the TA models.

In summary, the analytical methods build on a rigorous mathematical deduction under the worst-case assumptions of a simplified system, thereby fast solving the optimization problem at a low cost. By contrast, the model-based methods are more expressive to describe a concrete avionics system but their optimizers face the challenge of the complexity problem. In this paper, our model-based approach adopts a global evolutionary search to explore the objective solution space effectively and uses the integrated method of simulation-based tests and compositional verification to make the costly schedulability analysis feasible.

8 Conclusion

The model-based method presented in this paper addresses the optimization problem of ARINC-653 partition scheduling in a complex IMA system. We conclude that our model-based approach is applicable to this optimization problem, where an IMA system is modeled as a network of timed automata in Uppaal. Compared with widely-used analytic optimization, the timed automata model of our method is more expressive to describe complex features of IMA systems. We formulate the problem as a global search for the optimal partition scheduling parameters that achieve the minimum processor occupancy and meet the schedulability constraints. A parameter sweep optimizer explores the solution space via evolutionary algorithm while guaranteeing the schedulability by model checking. The evolutionary algorithm EA4HS is promising for reaching the optimal parameters quickly as well as avoids exhaustive exploration of the solution space. The combination of global SMC testing and compositional model-checking verification alleviates the state space explosion of classical model checking. The experiments demonstrate that our optimizer is able to identify the global optimum solutions for simple task sets and find acceptable ones effectively for complex systems.

The design of the model-based parameter sweep also introduces limitations into this optimization method. First, the evolutionary search may not reach a global optimal solution within a finite number of generations, especially when handling a complex IMA system with a large number of partitions. However, our method is still capable of producing better high-quality candidates than purely empirical scheduling schemes. Engineers can benefit from the optimization results of our method in integrating a set of complex applications. Second, the model-based method involves frequent time-consuming schedulability analyses, leading to a long processing time from a few hours to days. But we believe that such a processing time is negligible in the development life cycle of an IMA system. Moreover, we can speed up the optimization process by running on more powerful clusters. As future work, we plan to add more features such as multi-core processor support and more local scheduling policies to the system, further generalizing the proposed method to more complex ARINC-653 scheduling systems.

References

1. AEEC: Avionics application software standard interface: Part 1 - required services. ARINC specification 653P1-4, Aeronautical Radio Inc. (2015)
2. Bäck, T., Schwefel, H.P.: An overview of evolutionary algorithms for parameter optimization. Evolutionary computation **1**(1), 1–23 (1993)
3. Behrmann, G., David, A., Larsen, K.G.: A tutorial on uppaal. In: Formal methods for the design of real-time systems, pp. 200–236. Springer (2004)
4. Beji, S., Hamadou, S., Gherbi, A., Mullins, J.: Smt-based cost optimization approach for the integration of avionic functions in ima and ttethernet architectures. In: Proceedings of the 2014 IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications, pp. 165–174. IEEE Computer Society (2014)
5. Beyer, H.G.: An alternative explanation for the manner in which genetic algorithms operate. BioSystems **41**(1), 1–15 (1997)
6. Beyer, H.G., Schwefel, H.P.: Evolution strategies–a comprehensive introduction. Natural computing **1**(1), 3–52 (2002)
7. Blikstad, M., Karlsson, E., Lööw, T., Rönnberg, E.: An optimisation approach for pre-runtime scheduling of tasks and communication in an integrated modular avionic system. Optimization and Engineering **19**(4), 977–1004 (2018)
8. Boudjadar, A., David, A., Kim, J.H., Larsen, K.G., Mikučionis, M., Nyman, U., Skou, A.: Statistical and exact schedulability analysis of hierarchical scheduling systems. Science of Computer Programming **127**, 103–130 (2016)
9. Boudjadar, J., David, A., Kim, J.H., Larsen, K.G., Nyman, U., Skou, A.: Schedulability and energy efficiency for multi-core hierarchical scheduling systems. Embedded Real Time Systems and Software pp. 1–4 (2014)
10. Boudjadar, J., Larsen, K.G., Kim, J.H., Nyman, U.: Compositional schedulability analysis of an avionics system using UPPAAL. In: AASE 2014
11. Carnevali, L., Pinzuti, A., Vicario, E.: Compositional verification for hierarchical scheduling of real-time systems. IEEE Transactions on Software Engineering **39**(5), 638–657 (2013)
12. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B.: Uppaal smc tutorial. International Journal on Software Tools for Technology Transfer **17**(4), 397–415 (2015)
13. Davis, R., Burns, A.: An investigation into server parameter selection for hierarchical fixed priority pre-emptive systems. In: 16th International Conference on Real-Time and Network Systems (RTNS 2008) (2008)
14. Dewan, F., Fisher, N.: Approximate bandwidth allocation for fixed-priority-scheduled periodic resources. In: 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 247–256. IEEE (2010)
15. Easwaran, A., Anand, M., Lee, I.: Compositional analysis framework using edp resource models. In: 28th IEEE International Real-Time Systems Symposium (RTSS 2007), pp. 129–138. IEEE (2007)
16. Easwaran, A., Lee, I., Sokolsky, O., Vestal, S.: A compositional scheduling framework for digital avionics systems. In: ERCSA 2009
17. Freiberg, P.S., Krag, J.M., Villumsen, B.: Distributed parameter sweep for uppaal models. Ph.D. thesis, Aalborg University (2011)
18. Han, P., Zhai, Z., Nielsen, B., Nyman, U.: A modeling framework for schedulability analysis of distributed avionics systems. In: 3rd Workshop on Models for Formal Analysis of Real Systems and 6th International Workshop on Verification and Program Transformation, MARSVPT 2018, pp. 150–168. EPTCS (2018)
19. Han, P., Zhai, Z., Nielsen, B., Nyman, U., Kristjansen, M.: Schedulability analysis of distributed multicore avionics systems with uppaal. Journal of Aerospace Information Systems **16**(11), 473–499 (2019)
20. Han, P., Zhai, Z., Nielsen, B., Nyman, U.M.: A compositional approach for schedulability analysis of distributed avionics systems. In: International Workshop on Methods and Tools for Rigorous System Design, pp. 39–51 (2018)
21. Kelly, O.R., Aydin, H., Zhao, B.: On partitioned scheduling of fixed-priority mixed-criticality task sets. In: Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on, pp. 1051–1059. IEEE (2011)
22. Kim, J.E., Abdelzaher, T., Sha, L.: Schedulability bound for integrated modular avionics partitions. In: Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, pp. 37–42. EDA Consortium (2015)
23. Kim, J.H., Legay, A., Traonouez, L.M., Boudjadar, A., Nyman, U., Larsen, K.G., Lee, I., Choi, J.Y.: Optimizing the resource requirements of hierarchical scheduling systems. Acm Sigbed Review **13**(3), 41–48 (2016)
24. Lipari, G., Bini, E.: A methodology for designing hierarchical scheduling systems. Journal of Embedded Computing **1**(2), 257–269 (2005)
25. Mendoza, L.E., Capel, M.I., Pérez, M., Benghazi, K.: Compositional model-checking verification of critical systems. In: International Conference on Enterprise Information Systems, pp. 213–225. Springer (2008)
26. Mühlenbein, H., Schlierkamp-Voosen, D.: Predictive models for the breeder genetic algorithm i. continuous parameter optimization. Evolutionary computation **1**(1), 25–49 (1993)
27. Mühlenbein, H., Voigt, H.M.: Gene pool recombination in genetic algorithms. In: Meta-Heuristics, pp. 53–62. Springer (1996)

28. Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: CAV, vol. 3114, pp. 202–215. Springer (2004)
29. Shin, I., Lee, I.: Compositional real-time scheduling framework. In: Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International, pp. 57–67. IEEE (2004)
30. Shin, I., Lee, I.: Compositional real-time scheduling framework with periodic model. ACM Transactions on Embedded Computing Systems (TECS) **7**(3), 30 (2008)
31. Shukla, A., Pandey, H.M., Mehrotra, D.: Comparative review of selection techniques in genetic algorithm. In: Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), 2015 International Conference on, pp. 515–519. IEEE (2015)
32. Sumathi, S., Hamsapriya, T., Surekha, P.: Evolutionary intelligence: an introduction to theory and applications with Matlab. Springer Science & Business Media (2008)
33. Sun, Y., Lipari, G., Soulat, R., Fribourg, L., Markey, N.: Component-based analysis of hierarchical scheduling using linear hybrid automata. In: ERCSA 2014
34. Yoon, M.K., Kim, J.E., Bradford, R., Sha, L.: Holistic design parameter optimization of multiple periodic resources in hierarchical scheduling. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 1313–1318. EDA Consortium (2013)