IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 5, 2020, accepted August 25, 2020, date of publication September 18, 2020, date of current version October 6, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3025159

Optimal Harmonic Period Assignment With Constrained Number of Distinct Period Values

IVAN PAVIĆ, (Member, IEEE), AND HRVOJE DŽAPO, (Senior Member, IEEE)
Faculty of Electrical Engineering and Computing, University of Zagreb, 10000 Zagreb, Croatia

Corresponding author: Ivan Pavić (ivan.pavic2@fer.hr)

This work was supported by the European Regional Development Fund through the project "System for increased driving safety in public urban rail traffic (SafeTRAM)."

ABSTRACT Harmonic periods have been of great importance in the design of real-time applications due to their high schedulability, predictability, and ease of analysis. Therefore, period assignment is an important part of the design process of many real-time systems. This includes various applications such as radar dwell tasks, robotics, and industrial control applications, where tasks are specified using period ranges and worst-case execution times. In this paper, we study the issue of assigning a fixed number of harmonic periods from period ranges to maximize utilization in real-time systems. In the existing period assignment approaches, the number of different harmonic period values in the solution was not addressed. In this work, we show that, in real-time systems in which the number of available task periods is restricted, such a constraint is crucial for efficient system design. We formally define the problem in the context of existing harmonic period assignment research. We show that this problem is at least weakly NP-hard and devise an optimal algorithm and suboptimal heuristics. Based on an extensive evaluation on synthetically generated task sets, we conclude that our approach is efficient and applicable in a variety of real-world scenarios.

INDEX TERMS Harmonic, period assignment, period optimization, real-time systems.

I. INTRODUCTION

In traditional industrial control systems, timeliness, stability and predictability are very important properties. Moreover, in safety-critical control systems, these properties are *condicio sine qua non* as they are required according to generic safety standards such as IEC 61508 and domain-specific safety standards, e.g., EN 50128 in the railway domain and ISO 26262 in the automotive domain. Efficiency and accuracy of the control algorithm depends to a large extent on the timeliness of underlying embedded computing platforms. Consequently, strict requirements are imposed on the design of operating systems as the controlling procedures have to be prompt and correct. In this context, selecting adequate sample times, i.e., task periods, is crucial for an appropriate behavior of a system.

A. RELATED WORK

Period assignment is a well-studied topic in real-time system design, since the choice of periods in sporadic or periodic task sets has a direct effect on system schedulability, efficiency and utilization. Additionally, harmonic period assignment is of special interest as it is well-known that any harmonic task set with processor utilization less or equal to one is schedulable by a rate-monotonic scheduler [1]. Research in this domain can be divided into three groups with respect to the particular focus of the research.

The first research area includes papers focused on the schedulability of real-time systems with arbitrary period selection, i.e., periods of task sets are not constrained to harmonic values. Early research in this context was done by Seto at al. in [2]. The authors devised algorithms for discovering feasible integer periods in fixed-priority systems with a fixed rate-monotonic and an arbitrary priority assignment. Moreover, in their approach periods are upper-bounded by the slowest task rate required by an application. According to authors in [3], the approach taken in [2] seems to be inefficient due to the combinatorial explosion for larger task sets. On the other hand, in [3], authors precisely formulate the feasibility region in the rate space and devise optimization algorithm for any convex objective. In their approach, periods are not constrained to be integers, while fixed-priority scheduling is assumed. However, there are no additional constraints regarding the period range for tasks in systems.

The associate editor coordinating the review of this manuscript and approving it for publication was Shafiqul Islam.

The second research area includes papers that exploit the harmonic relations between the periods of tasks in systems for determining schedulability. For instance, Han and Tyan in [4] devise a sufficient schedulability bound which is better than the one proposed by Liu and Layland [5]. The approach is based on two previously introduced algorithms Sr and DCT investigated in the context of distance-constrained real-time systems [6]. Similarly, an exact polynomial-time schedulability test for harmonic task sets with any fixed-priority assignment was devised in [7]. It is worth noting that the problem of determining schedulability of sporadic task sets with arbitrary periods is NP-hard [8].

The third research area includes papers which are focused on harmonic period assignment with period ranges. In recent research [9], authors determined that two classical harmonic period optimization problems labeled UHPA (utilization-maximizing harmonic period assignment) and CHPA (cost-minimizing harmonic period assignment) are in the NP-hard complexity class. Additionally, they devised approximation algorithms for the relaxed version of the CHPA problem, i.e., the constraints on period ranges are removed. We formally define these problems later in the paper as our research is focused on a variant of the UHPA problem with additional constraints on period values. In [10], the authors introduce the notion of harmonic projection and devise an exponential time (in size of a task set) algorithm for determining harmonic periods for tasks with period ranges. Additionally, they devise period assignment algorithms such that the resulting utilization of a task set is equal to the lower or the upper bound utilization value. They expand on their work in [11]. We highlight the similarities and the differences with our approach later in the paper. Period selection and assignment were investigated in the context of minimizing the hyperperiod of task set in systems in which periods are closely harmonically related [12]. Similarly, in [13], the authors investigate the minimization of the hyperperiod by non-harmonic period assignment from period ranges.

The common motivation in the period assignment research is the real-time system and optimal controller co-design, in which the problem of selecting adequate sample times is directly linked to the problem of determining optimal periods [14]. In such approaches, the LQG (linear-quadratic-Gaussian) plant model is used [15], [16]. Moreover, the period assignment of harmonic period values is of great importance in many applications such as radar dwell tasks [17], mobile robotics [18], integrated modular avionics [19], and automotive applications [20].

B. MOTIVATION AND NEW CHALLENGES

As an additional motivation and rationale for imposing additional constraints on the classical harmonic period assignment problem, i.e., a variant of the UHPA, we focus on safety-critical software from real-world industrial scenarios as it serves as the primary motivation for this research. Safety-critical embedded software for control applications typically has a modular composition in which each module, task or runnable executes with a predefined period which is determined off-line as a part of the application design. For instance, ANSYS SCADE Suite [21], HIMA SILworX [22] and KONČAR Grap Designer [23] provide automatic code generation based on a set of application modules. The application designer determines a range of periods for every module, i.e., task, in the application. In order to ensure function correctness, tasks have to be executed with periods belonging to their specified range. Moreover, it is in the interest of application and system designers that every task in the system executes with the highest possible frequency, i.e, the lowest possible period as this will ensure a higher quality of service, and consequently increase utilization. Thus, utilization is maximized. The number of tasks in an application can grow and be arbitrarily high. However, in many systems, e.g., the KONČAR Grap [23] operating system, or engine management systems [20] in automotive applications, the number of available periods, i.e., rates, is fixed or bounded and cannot be increased due to the specific architecture of the hardware and the operating system. For instance, the maximum number of different period values may be fixed to 4, or restricted to the interval from 4 to 8. It is worth noting that previous research regarding harmonic period assignment does not address the number of distinct period values in the solution of the period assignment problem. Our approach can be used by system and application designers to determine the optimal choice of task periods, even when the number of available periods in the system is limited.

C. CONTRIBUTIONS AND ORGANIZATION

There are five main contributions in this paper. We define utilization-maximizing harmonic period assignment with a constrained number of distinct period values referred to as UDHPA (i). We show that the already studied UHPA problem is Turing reducible to the UDHPA problem. Additionally, using the complexity results for the UHPA problem from [9], we determine that the UDHPA problem is at least weakly NP-hard by reduction from the well-known partition sum problem (ii). We devise an optimal and heuristic algorithms for the UDHPA problem, provide time-complexity analysis, and show the effectiveness of the approach with extensive evaluation on a large number of synthetically generated instances, which correspond to real-world motivational scenarios (iii). We use our optimal algorithm and heuristics to solve instances of the UHPA problem and compare them to the existing approaches (iv). Moreover, we provide a numerical example that illustrates a real-world period assignment problem (v). The rest of the paper is organized as follows. In section II, we introduce the system model. In section III, firstly we revisit existing harmonic period assignment problems, and then define the UDHPA problem. In section IV, we analyze the complexity of the UDHPA problem. In section V, we devise an optimal algorithm for the UDHPA

problem. In section VI, we evaluate our approach. Finally, in section VII, we state the concluding remarks.

II. SYSTEM MODEL

In this paper, system S is represented as a set $\mathcal{T}$ of n periodic tasks with no initial offset, i.e., a synchronous task set [24]. We use a task model which is common for the period assignment with period ranges proposed in [9], [10], and [11]. Therefore, task τ_i is represented as a tuple $\tau_i = \{C_i, I_i\}$, where C_i is the worst-case execution time (WCET) of τ_i, and I_i is the period range of allowed period values for τ_i. Period range $I_i = [p_i^{min}, p_i^{max}]$ is determined by the minimal p_i^{min} and the maximal p_i^{max} allowed period value. Actual period value of task τ_i is denoted as T_i. This is the value which is assigned to a task by solving a period assignment problem. Additionally, T_i is the deadline of τ_i, i.e., implicit-deadline task sets are considered. We assume that the WCET is a real number, i.e., $C_i \in \mathbb{R}$, and that task periods are integers, i.e., $T_i, p_i^{min}, p_i^{max} \in \mathbb{N}$. Moreover, periods are in harmonic relation, i.e., $\frac{T_i}{T_j} \in \mathbb{N} \vee \frac{T_j}{T_i} \in \mathbb{N}, i, j \in [1, n]$. Definition II.1 determines the correctness of period assignment.

Definition II.1. Correct Period Assignment. Period assignment for task τ_i is correct iff period value T_i assigned to task τ_i is such that $p_i^{min} \leq T_i \leq p_i^{max}$.

It is worth noting that the correctness of period assignment does not guarantee the feasibility of a task set. The feasibility of a harmonic task set is often expressed using the utilization of a task set, i.e., $U = \sum U_i = \sum \frac{C_i}{T_i} \leq 1$.

III. PROBLEM FORMULATION

A. CLASSICAL HARMONIC PERIOD ASSIGNMENT PROBLEMS

First off, we describe existing harmonic period assignment problems which are analyzed and discussed in the literature [9]–[11]. Inputs of these problems are tasks described with worst-case execution times C_i and period ranges $I_i = [p_i^{min}, p_i^{max}]$. The outputs are period values T_i assigned to each task. The utilization-maximizing harmonic period assignment (UHPA) problem is formulated as follows:

$$\begin{aligned} &\text{maximize} \quad U = \sum_{i=1}^{n} U_i \\ &\text{subject to } T_i \in I_i, \quad i \in [1, n] \\ &\qquad \frac{T_i}{T_j} \in \mathbb{N} \quad \text{or} \quad \frac{T_j}{T_i} \in \mathbb{N}, \quad i, j \in [1, n] \\ &\qquad U \leq 1 \end{aligned}$$

As it can be seen, actual periods T_i are allowed to be in range I_i and have to be in harmonic relation. It is worth noting that in the related literature (e.g., [9]) period values T_i are not restricted to the integer values. However, the period ratios of the two consecutive integers $\frac{T_i}{T_j}$ are required to be integers. The second common problem formulation in the literature is the cost-minimizing harmonic period assignment (CHPA) problem which can be formulated as:

$$\begin{aligned} &\text{minimize} \quad \sum_{i=1}^{n} w_i T_i \\ &\text{subject to } T_i \in I_i, \quad i \in [1, n] \\ &\qquad \frac{T_i}{T_j} \in \mathbb{N} \quad \text{or} \quad \frac{T_j}{T_i} \in \mathbb{N}, \quad i, j \in [1, n] \\ &\qquad U \leq 1 \end{aligned}$$

The CHPA problem is common in control co-design applications [15] where the goal function is a linear function of task periods. In the goal function, the weight w_i determines the contribution of each period to the total cost.

B. FORMULATION OF HARMONIC ASSIGNMENT PROBLEM WITH A CONSTRAINED NUMBER OF DISTINCT PERIOD VALUES

In this paper, we solve the utilization-maximizing harmonic period assignment problem with a constrained number of different period values. In the previous problems, the number of different period values in the solution is not constrained. For instance, an optimal solution can have any number of different period values, from only one up to n. As argued in the motivation of the paper, the number of different period values can be smaller than the number of tasks in the system. To address this, we formulate the problem in which the number of different period values is constrained, i.e., fixed. Such an approach enables a more flexible system design as it allows the system designer to regulate the number of distinct period values in the system. This is elaborated further using the numerical example in section VI-D. Now, we proceed to formulate such a problem. Firstly, we introduce the vector $\vec{p}$ which contains m different period values, where $m \leq n$. Secondly, we introduce the period assignment matrix $\mathbf{X}$ which contains mapping of period value p_j to period value T_i of task τ_i. Value x_{ij} of the binary matrix $\mathbf{X}$ is determined as follows:

$$x_{ij} = \begin{cases} 1, & \text{period value } p_j \text{ is assigned to task } \tau_i, \text{ i.e., } T_i \leftarrow p_j \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

In case $m = n$, one period value p_j maps to only one period value T_i, i.e., this is one to one mapping. In case $m \leq n$, period value p_j can be mapped to many tasks, i.e., generally, this is one to many mapping. However, period value p_j has to be mapped to at least one period value T_i. If p_j is not mapped to at least one task, then the number of different period values in the resulting period assignment would not be equal to m, and this is the requirement of the problem. To express this formally, we introduce the constraints given with equations (2) and (3). First, we restrict the assignment of only one period value p_j to period value T_i of task τ_i:

$$\sum_{j=1}^{m} x_{ij} = 1, \quad i \in [1, n] \tag{2}$$

The constraint (2) ensures that the i-th row of matrix $\mathbf{X}$ can contain only one non-zero element, i.e., only one p_j is mapped to T_i. Secondly, every period value p_j has to be assigned to at least one task:

$$\sum_{i=1}^{n} x_{ij} \geq 1, \quad j \in [1, m] \tag{3}$$

The constraint (3) ensures that the j-th column of matrix $\mathbf{X}$ has to contain at least one non-zero element, i.e., every period value p_j has to be assigned to a task. The latter constraints do not ensure the correctness of a period assignment. For instance, a period value which is too high or too low can be assigned to a task. To ensure that a period assignment is correct (Def. II.1), we have to restrict the assignment to period values from interval $[p_i^{min}, p_i^{max}]$. The correctness criteria can be expressed as:

$$x_{ij} \implies p_i^{min} \leq p_j \leq p_i^{max} \tag{4}$$

The latter logical expression can be translated to an arithmetic expression using the translation for logical implication to a linear constraint:

$$X \implies Y \rightarrow x \leq y \tag{5}$$

At this point, we introduce the binary constraint matrix $\mathbf{A}$ with $n \cdot m$ elements a_{ij}. Value a_{ij} is determined as follows:

$$a_{ij} = \begin{cases} 1, & p_i^{min} \leq p_j \leq p_i^{max} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

To ensure the correctness of a period assignment, $x_{ij} \implies a_{ij}$ has to hold. Using the arithmetic counterpart for implication given with (5), the correctness criteria can be expressed as:

$$x_{ij} \leq a_{ij}, \quad \forall i \in [1, n], \forall j \in [1, m] \tag{7}$$

With the latter constraints in place, the problem can be formally expressed as:

$$\begin{aligned} \text{maximize} \quad & U = \sum_{j=1}^{m} \sum_{i=1}^{n} C_i \frac{x_{ij}}{p_j} \\ \text{subject to} \quad & x_{ij} \leq a_{ij}, \quad i \in [1, n],\ j \in [1, m] \\ & \sum_{j=1}^{m} x_{ij} = 1, \quad i \in [1, n] \\ & \sum_{i=1}^{n} x_{ij} \geq 1, \quad j \in [1, m] \\ & p_j = k_j p_{j-1}, \quad k_j \in \mathbb{N}^+ \setminus \{1\},\ j \in [2, m] \\ & U \leq 1 \end{aligned}$$

where k_j is the integer ratio of two consecutive period values p_j and p_{j-1}. We refer to this problem as the utilization-maximizing harmonic period assignment with a constrained number of different period values (UDHPA). The outputs of the UDHPA problem are period vector $\vec{p}$ and period assignment matrix $\mathbf{X}$. These two variables determine period T_i for each task in system. Similarly as in the classical problems, the inputs of the problem are tasks represented with worst-case execution times C_i and period ranges I_i. The additional input in the UDHPA problem is the number of distinct period values in the solution m.

IV. PROBLEM ANALYSIS: TURING REDUCIBILITY AND COMPLEXITY

A. TURING REDUCTION FROM UHPA TO UDHPA

In order to show that the UHPA problem is Turing reducible to the UDHPA problem, we show that an oracle for the UDHPA problem can be used to solve the UHPA problem. Formally, this is stated with the following lemma.

Lemma IV.1. *The UHPA problem can be solved by solving the UDHPA problem n times.*

Proof: If we consider the UHPA problem, an optimal solution has an arbitrary number of distinct period values which can be lower than the number of tasks in the system. In the UDHPA problem, however, the number of distinct period values is fixed. Therefore, to solve the UHPA problem using an algorithm for the UDHPA problem, one has to solve the UDHPA problem for every number m of distinct period values from interval $[1, n]$. To better illustrate this, we provide pseudo-code Alg. 1. □

Algorithm 1 Turing Reduction From UHPA to UDHPA

Input: $S = \mathcal{T}$
Output: U_{max} ▷ maximal utilization

```
function SolveUHPA(T)
    U_max ← 0
    for m in range 1 to n do
        U ← SolveUDHPA(T, m)
        if U > U_max then
            U_max ← U
        end if
    end for
    return U_max
end function
```

Theorem IV.1. *The UHPA problem is Turing reducible to the UDHPA problem.*

Proof: In Lemma IV.1, we can see that the UHPA problem is solved by invoking the oracle for the UDHPA problem in polynomial time, which proves the theorem. □

B. COMPLEXITY ANALYSIS

To derive the complexity of the UDHPA problem, we use the UHPA complexity results from [9]. To show that the UHPA problem is at least weakly NP-hard, the authors provided many-one reduction from the partition sum problem (PART) to the UHPA problem. In other words, they provided a polynomial-time algorithm for reducing any given instance of the PART problem to an instance of the UHPA problem. Their proof can be directly applied to the UDHPA problem. For completeness and clarity, here we reproduce important parts of the proof. For the complete proof, refer

to section 3 in [9]. Firstly, we define the number partitioning problem (PART).

Definition IV.1. The PART Problem. Let $A = \{a_1, \ldots, a_n\}$ be a set of n items with an associated size function $s : A \to \mathbb{N}$ which assigns a positive integer to each item. The problem is to determine whether A can be partitioned into two sets, A_1 and A_2, such that the total size of items in A_1 equals that of A_2. More formally, let S, S_1, and S_2 denote the sum of items for A, A_1, and A_2, respectively. That is,

$$S = \sum_{a_i \in A} s(a_i) \tag{8}$$

$$S_1 = \sum_{a_i \in A_1} s(a_i) \tag{9}$$

$$S_2 = \sum_{a_i \in A_2} s(a_i) \tag{10}$$

Then, the problem is to decide whether A can be partitioned into A_1 and A_2 (i.e., $A_1 \cup A_2 = A$ and $A_1 \cap A_2 = \emptyset$), such that $S_1 = S_2$. An instance of this problem is said to be a *positive* one if such a partitioning exists [9].
The PART problem is known to be NP-complete, but solvable in pseudo-polynomial time [25].

At this point, we reproduce the polynomial-time method for transforming any given instance of the PART problem to an instance of the UHPA problem. To show that the proof is applicable to UDHPA problem as well, we show that the transformation of any PART instance to an instance of the UHPA is a transformation to an instance of the UDHPA problem as well.

Definition IV.2. PART Transformation. For any instance of the PART problem, the corresponding UHPA problem is specified by a set of $n+2$ tasks. The WCET of τ_i is determined as:

$$C_i = \begin{cases} \dfrac{4s(a_i)}{3S+3}, & 1 \le i \le n \\ \dfrac{2}{3S+3}, & n+1 \le i \le n+2 \end{cases} \tag{11}$$

Period ranges for each τ_i are determined as:

$$I_i = \begin{cases} [1,2], & 1 \le i \le n \\ [1,1], & i = n+1 \\ [2,2], & i = n+2 \end{cases} \tag{12}$$

Proposition IV.1. *An UHPA instance obtained using **PART transformation** is an UDHPA instance with $m = 2$.*

Proof: Any instance of the PART problem is transformed to an instance of UHPA problem with period ranges such that the allowed harmonic period values are either 1 or 2. There are always exactly two different period values in the resulting UHPA problem. Therefore, any such instance is an UDHPA instance with two different period values, i.e., $m = 2$. □

Lemma IV.2. *A given instance of the PART problem is positive (i.e., the given set can be partitioned) if and only if the UHPA problem instance obtained from **PART transformation** has a solution in which $U = 1$ [9].*

The latter lemma is proven in [9]. It is worth noting that it applies to the UDHPA problem as well, since we know from Theorem IV.1. that every instance of the UHPA problem obtained using **PART transformation** can be solved with an oracle for the UDHPA problem with $m = 2$, i.e., there is one to one mapping between instances of the UHPA and UDHPA problem with fixed $m = 2$.

Theorem IV.2. *The UDHPA problem is at least weakly NP-hard.*

Proof: Using the **PART transformation** and Lemma IV.2. we can reduce any PART instance to a corresponding UHPA instance. Additionally, using Proposition IV.1 and Theorem IV.1 we see that this transformation is valid for the UDHPA problem as well. Therefore, any algorithm used for solving the UDHPA problem can be used for solving any instance of the PART problem after the **PART transformation**. Therefore, the UDHPA problem is at least hard as the PART problem. Moreover, the UDHPA problem is at least weakly NP-hard. □

V. AN OPTIMAL ALGORITHM FOR THE UDHPA PROBLEM

The UDHPA problem cannot be easily solved by using existing mixed-integer or integer programming solvers. The utilization of the system, i.e., the goal function is a non-linear, i.e., signomial, function. Methods for solving mixed-integer signomial problems exist, but do not guarantee to find a global solution [26].

In our approach, we enumerate possible solutions to find an optimal solution of the problem. We split the UDHPA problem into two independent parts:

1) enumeration of potential harmonic period sets – we refer to this part as period enumeration (PE),
2) assignment of periods from a harmonic period set to tasks - we refer to this part as task assignment (TA).

In the first part, we enumerate the possible harmonic period sets which can be used for task assignment in the system. This is possible as we constrained periods to integer values. Still, in the worst case, enumerating all the possible harmonic period sets can lead to a combinatorial explosion due to the exponential growth in the period search space. Therefore, we introduce several propositions which drastically reduce search space in most use cases. In the second part, we assume that harmonic period set $\vec{p}$ is known. It can be seen that the goal function of the UDHPA problem with known period values is linear, which makes such problem an integer linear program, i.e., zero-one linear program. We refer to the relaxed version of the UDHPA problem, i.e., an assignment problem with a known harmonic period set, as the task assignment (TA) problem. The TA problem can be expressed as follows:

$$\text{maximize } U = \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{C_i}{p_j} x_{ij}$$
$$\text{subject to } x_{ij} \le a_{ij}, \quad i \in [1,n], j \in [1,m]$$

$$\sum_{j=1}^{m} x_{ij} = 1, \quad i \in [1, n]$$
$$\sum_{i=1}^{n} x_{ij} \geq 1, \quad j \in [1, m]$$
$$U \leq 1$$

In the TA problem, only the mapping of period values to tasks has to be determined, since harmonic period set $\vec{p}$ is known in advance. In our approach, we exploit this observation. Firstly, we enumerate possible harmonic period values sets, and then solve the TA problem for each harmonic period set.

A. ENUMERATING PERIOD VALUES

To determine m different period values, we start by choosing the values for the lowest period, i.e., p_1. Subsequent period values are determined by choosing the integer ratios $k_j > 1$ of two consecutive integer values p_j and p_{j-1}. Possible values for the first period depend on two specific values which can be determined from task period ranges. The first value is p_{min}^{min}, which is the minimal lower bound in all period ranges, i.e., $\min p_i^{min}$. The second value is p_{min}^{max}, which is the minimal upper bound in all period ranges, i.e., $\min p_i^{max}$.

Proposition V.1. ***Choice of*** p_1**.** *In any feasible solution of the UDHPA problem, value of* p_1 *is in interval* $[p_{min}^{min}, p_{min}^{max}]$.

Proof: To prove this, we consider cases in which p_1 is not in the proposed interval and show that at least one of the constraints is violated. Firstly, let us assume that p_1 has a value which is lower than p_{min}^{min}. As the period value has to be assigned to at least one task, i.e., $\sum_{i=1}^{n} x_{ij} \geq 1, j \in [1, m]$, there is no feasible solution to the UDHPA problem, since period value p_1 cannot be assigned to task τ_i with the minimal lower bound $p_i^{min} = p_{min}^{min}$, or to any other task. Secondly, if p_1 is greater than the minimal upper bound p_{min}^{max}, it is not possible to assign any period value to task τ_i with $p_i^{max} = p_{min}^{max}$ due to the violation of the period range constraint. Therefore, for any feasible solution to the UDHPA problem, p_1 is within the interval $[p_{min}^{min}, p_{min}^{max}]$. □

Similarly, as we have restricted the possible choices of p_1, we can restrict the choice of any subsequent period values p_j. We introduce another specific value for the given period ranges, p_{max}^{max}, which denotes the maximal upper bound among the upper bounds of all tasks, i.e., $\max p_i^{max}$.

Proposition V.2. ***A Bound on Choice of*** p_j**.** *In any feasible solution, there is no period value such that* $p_j > p_{max}^{max}$.

Proof: If $p_j > p_{max}^{max}$, it is not possible to assign p_j to any task, since the range constraints will be violated. Therefore, in any feasible solution, every period p_j is less than or equal to p_{max}^{max}. □

Using the specific values obtained from period ranges, it is possible to determine the maximum number of distinct period values which can appear in the solution. The following proposition is useful as it restricts the period enumeration search space.

Proposition V.3. ***The Maximum Number of Distinct Period Values.*** *In any feasible solution the maximum number of distinct period values* m *is such that* $m \leq m_{max} = \lfloor \log_2 \frac{p_{max}^{max}}{p_{min}^{min}} + 1 \rfloor$.

Proof: Due to the harmonic relations of period values, it is evident that we obtain the smallest value of the largest period, i.e., p_m, when all consecutive integer ratios are such that $k_j = 2, j \in [1, m-1]$. Then, we have $p_m^{min} = p_{min}^{min} \cdot 2^{m-1}$. From Proposition V.2, we know that $p_m^{min} \leq p_{max}^{max}$, and therefore $p_{min}^{min} \cdot 2^{m-1} \leq p_{max}^{max}$. When we solve the latter inequality for m, we get $m \leq \log_2 \frac{p_{max}^{max}}{p_{min}^{min}} + 1$, which proves the proposition. □

Proposition V.4. ***Maximum Integer Factor*** k_j**.** *In any feasible solution, every integer factor* k_j *is such that* $k_j \leq \lfloor \frac{p_{max}^{max}}{p_{j-1} \cdot 2^{m-j}} \rfloor$.

Proof: Let $p_m \leq p_{max}^{max}$ be the last value of the period vector $\vec{p}$. Period p_m can be calculated as $p_m = p_1 \cdot k_2 \cdot k_3 \cdot \ldots \cdot k_j \cdot \ldots \cdot k_m$. Moreover, $p_m = p_{j-1} \cdot k_j \cdot k_{j+1} \cdot \ldots \cdot k_m$. Therefore, $k_j = \lfloor \frac{p_m}{p_{j-1} \cdot \prod_{i=j+1}^{m} k_i} \rfloor$. Factor k_j is maximal when the product $\prod_{i=j+1}^{m} k_i$ is minimal, i.e., $k_i = 2, i > j$. Therefore, $k_j \leq \lfloor \frac{p_m}{p_{j-1} \cdot 2^{m-j}} \rfloor$, which proves the proposition. □

Using the latter propositions, harmonic period sets can be enumerated in an efficient manner. Alg. 2 depicts a recursive algorithm for the enumeration of period sets. Firstly, the `Period Enumeration` function assigns values determined by Proposition V.1 to the first period p_1 (line 6 in Alg. 2). The subsequent period values are determined according to Proposition V.4 in a recursive manner (line 15 in Alg. 2). In the basic case, when all the period values are set to their respective values, i.e., when $j == m + 1$, the TA problem is solved for the constructed period set. Moreover,

Algorithm 2 Algorithm for Harmonic Period Enumeration

Input: $S = (\mathcal{T})$

1: **function** Period Enumeration($\mathcal{T}, m$)
2: $p_{min}^{min} = \min p_i^{min}, \quad \forall \tau_i \in \mathcal{T}$
3: $p_{min}^{max} = \min p_i^{max}, \quad \forall \tau_i \in \mathcal{T}$
4: $p_{max}^{max} = \max p_i^{max}, \quad \forall \tau_i \in \mathcal{T}$
5: **for** i in range p_{min}^{min} to p_{min}^{max} **do**
6: $p_1 \leftarrow i$ ▷ according to Prop. V.1
7: Period Enumeration Step($\vec{p}$, p_{max}^{max}, 2)
8: **end for**
9: **end function**
10: **function** Period Enumeration Step($\vec{p}$, p_{max}^{max}, j)
11: **if** j ==m+1 **then**
12: Solve Task Assignment($\vec{p}$, $\mathcal{T}$)
13: **return**
14: **end if**
15: **for** k_j in range 2 to $\lfloor \frac{p_{max}^{max}}{p_j \cdot 2^{m-j}} \rfloor$ **do** ▷ according to Prop. V.4
16: $p_j \leftarrow k_j \cdot p_{j-1}$
17: Period Enumeration Step($\vec{p}$, p_{max}^{max}, $j+1$)
18: **end for**
19: **end function**

the number of different harmonic period sets with m distinct period values corresponds to the number in which the basic case is reached. We further analyze the number of solutions with regard to m in section VI in the context of feasibility evaluation. Now, we analyze time complexity of the period enumeration algorithm. Firstly, it is worth noting that the asymptotic analysis with regard to m is not of any interest, since we know from Proposition V.3 that m is bounded and that there are no feasible solutions for higher values of m. Therefore, we analyze time complexity with regard to the highest period in the input p_{max}^{max}, as it is obvious that the number of steps in the algorithm increases when p_{max}^{max} increases (see loop bound in line 15 in Alg. 2). To provide an asymptotic upper bound on the time complexity of Alg. 2, we study a similar enumeration problem referred to as the DIVENUM problem.

Definition V.1. The DIVENUM Problem. The enumeration problem, which we call DIVENUM, is to output all m-tuples $(k_1, \ldots, k_m)$ such that:

$$\Pi_{j=1}^{m} k_j \leq \chi, \quad k_j \in \mathbb{N}^+, \forall j \tag{13}$$

We can see that the DIVENUM problem is in fact very similar to the PE problem, since in both problems we are looking for a set of m factors such that their product is lower than the specified bound, χ and p_{max}^{max}, respectively. In the PE problem, each factor k_j is greater than one. On the other hand, in the DIVENUM problem, k_j is a positive integer including one. Therefore, we know that the number of steps required to enumerate solutions to the DIVENUM problem is always higher than the number of steps in the PE problem. For the sake of completeness and clarity, we provide Alg. 3 that depicts the enumeration for the DIVENUM problem. Moreover, we analyze the time complexity of the Alg. 3, and the obtained result will serve as an upper bound of the time complexity of the PE algorithm (Alg. 2).

Algorithm 3 Algorithm for the DIVENUM Problem

Input: χ, j

```
1: function Divisor Enumeration(χ, j)
2:     if j == 0 then
3:         output tuple (k_1, ..., k_m)
4:         return
5:     end if
6:     for k_{m+1-j} in range 1 to χ do
7:         Divisor Enumeration(χ / k_{m+1-j}, j − 1)
8:     end for
9: end function
```

In Alg. 3, we see that the number of steps $T(\chi, m)$ required to enumerate all m-tuples of positive integers with product less than or equal to χ is given with:

$$T(\chi, m) = \sum_{k=1}^{\chi} T(\lfloor \frac{\chi}{k} \rfloor, m-1) \tag{14}$$

$$T(\chi, 0) = 1 \tag{15}$$

where $T(\chi, 0)$ is the number of elementary operations in the basic case. For any bound x, we know that:

$$T(x, 1) = \sum_{k=1}^{x} T(x, 0) = x \tag{16}$$

Moreover, for $T(x, 2)$ we have the following:

$$\begin{aligned} T(x, 2) &= T(\frac{x}{1}, 1) + T(\frac{x}{2}, 1) + T(\frac{x}{3}, 1) + \ldots + T(\frac{x}{x}, 1) \\ &= \frac{x}{1} + \frac{x}{2} + \frac{x}{3} + \ldots + \frac{x}{x} \end{aligned}$$

We can see that this is in fact a finite partial sum of the harmonic series:

$$T(x, 2) = x \sum_{k=1}^{x} \frac{1}{k} = xH_x \tag{17}$$

where $H_x = \sum_{k=1}^{x} \frac{1}{k}$ is the x-th harmonic number. Now, with $T(x, 2) = xH_x$, $T(x, 3)$ can be expressed as:

$$\begin{aligned} T(x, 3) &= T(\frac{x}{1}, 2) + T(\frac{x}{2}, 2) + T(\frac{x}{3}, 2) + \ldots + T(\frac{x}{x}, 2) \\ &= xH_x + \frac{x}{2}H_{x/2} + \frac{x}{3}H_{x/3} + \ldots + \frac{x}{x}H_1 \end{aligned}$$

Next, we can bound $T(x, 3)$:

$$\begin{aligned} T(x, 3) &= xH_x + \frac{x}{2}H_{x/2} + \frac{x}{3}H_{x/3} + \ldots + \frac{x}{x}H_1 \\ &\leq xH_x(1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{x}) \\ &= xH_x^2 \end{aligned}$$

By induction, we get that $T(\chi, m) = \chi H_\chi^{m-1}$. Moreover, harmonic numbers can be approximated with an integral:

$$H_\chi = \int_1^\chi \frac{1}{t} dt = \ln \chi \tag{18}$$

Therefore, an upper bound on the time complexity for the DIVENUM problem is given with $O(\chi \log^{m-1}(\chi))$. Moreover, the time complexity of the PE algorithm is in $O(p_{max}^{max} \log^{m-1}(p_{max}^{max}))$.

B. SOLVING THE TA PROBLEM

The TA problem is solved by enumeration of all possible period to task assignments with respect to the given harmonic period set. First off, we devise polynomial-time algorithms which yield the lower and upper bounds for the goal function, i.e., the utilization of the system. Then, by using these bounds, we devise an algorithm for the optimal task assignment.

1) BOUND ALGORITHMS

In order to determine the bounds, we relax constraints $\sum_{i=1}^{n} x_{ij} \geq 1, j \in [1, m]$ to allow that some period values remain unused in the solution. We refer to this problem as TA*. In such a scenario, an algorithm that produces a lower bound of the utilization assigns the highest correct period to every task (note: correct with respect to Definition II.1). We refer to this assignment as the HPF (highest period first) assignment, and it is depicted with Alg. 4. Similarly, the LPF

(lowest period first) assignment yields an upper bound of the TA* problem. To obtain the LPF assignment using the Alg. 4, period values $\vec{p}$ have to be iterated from the lowest to the highest period value (line 6 in Alg. 4). Formally, the properties of the HPF assignment are stated in the continuation (Proposition V.5 and V.6). The properties of the LPF assignment are analogous with respect to an upper bound of utilization.

Algorithm 4 HPF Algorithm

Input: $\vec{p}, \mathcal{T}$

Output: (correctness, U, $\mathbf{X}$) - correctness of assignment, utilization and period assignment matrix

```
1:  function HPF Assignment(p⃗, T)
2:      X ← [0], n ← |T|, m ← |p⃗|
3:      U ← 0, correctness ← true
4:      for i in range 1 to n do
5:          assigned ← false
6:          for j in range m down to 1 do
7:              if p_i^min ≤ p_j ≤ p_i^max then
8:                  x_ij ← 1
9:                  U ← U + C_i/p_j
10:                 assigned ← true
11:                 break
12:             end if
13:         end for
14:         if not assigned then
15:             correctness ← false
16:             break
17:         end if
18:     end for
19:     return (correctness, U, X)
20: end function
```

Proposition V.5. *The highest period first (HPF) assignment yields the tight lower bound for the TA* problem.*

Proof: Without loss of generality, we observe one task τ_i from task set $\mathcal{T}$ and two correct periods p_j and p_k such that $p_j < p_k$. Assume that U_0 is utilization of task set $\mathcal{T}$ without τ_i and that to each task a period value is assigned correctly. We observe two possible period assignments for τ_i. In the first case, $T_i \leftarrow p_j$, and system utilization is $U = U_0 + \frac{C_i}{p_j}$. In the second case, $T_i \leftarrow p_k$, and system utilization is $U' = U_0 + \frac{C_i}{p_k}$. As $p_j < p_k$, it follows that $\frac{C_i}{p_k} < \frac{C_i}{p_j}$, and consequently $U' < U$. Therefore, assignment to higher period p_k for each τ_i will yield a lower bound of utilization. Moreover, this lower bound is tight. □

Proposition V.6. *The highest period first (HPF) period assignment yields a lower bound of utilization for the TA problem.*

Proof: Proposition V.5 guarantees that the HPF will yield minimal utilization in the case when there are no restrictions on the number of the distinct period values which have to appear in the solution. The HPF algorithm tries to assign the highest period value to each task, but this is not possible if we have restriction on the number of distinct period values as all of the period values have to be used. Therefore, if the highest correct period value cannot be assigned to the task, the lower period value will be assigned to the task and consequently utilization will increase. Therefore, the HPF assignment yields a lower bound of the utilization for the TA problem. However, in this case, this bound may not be tight. □

Both, the HPF and the LPF, have polynomial-time complexity, which is evident from Alg. 4. For each task in a task set, i.e., in n steps, the highest or the lowest period is chosen in at most m steps. Therefore, the time complexity of these algorithms is $O(n \cdot m)$. Note that these algorithms are not suitable for solving UDHPA instances as they do not guarantee that the number of used values in the solution will be equal to m, i.e., some period values from $\vec{p}$ may remain unused. However, in cases when there is restriction only on the maximal number of period values, i.e., solution can have any number of period values from 1 to m_{max}, usage of these algorithms is appropriate. Moreover, usage of these algorithms is appropriate for UHPA instances as in the UHPA problem there are no constraints on the number of period values. In such cases, we first use period enumeration to find appropriate period sets, and HPF or LPF approach to find corresponding task assignments. Time complexity of the approach is pseudo-polynomial with regard to p_{max}^{max} as time complexity of PE algorithm is in $O(p_{max}^{max} \log^{m-1}(p_{max}^{max}))$, and polynomial regarding n, as time complexity of HPF and LPF is in $O(m \cdot n)$

2) THE OPTIMAL TASK ASSIGNMENT ALGORITHM

An optimal algorithm for enumeration of task assignments is depicted with Alg. 5. We refer to this algorithm as optimal task assignment (OTA). Prior to explaining the optimal task assignment algorithm, we explain how the bounds calculated by the HPF and the LPF algorithm are used. Moreover, we explain how the number of distinct period values is tracked during the enumeration process.

a: BOUNDS OF UTILIZATION

Using the HPF and the LPF assignment, we construct the vector of lower bounds $\vec{b^l}$ and the vector of upper bounds $\vec{b^u}$. These vectors are used to prune infeasible or suboptimal branches in the enumeration of task assignments. These vectors contain upper and lower bounds of subsets of task set $\mathcal{T}$. In this context, the i-th subset of task set $\mathcal{T}$ is set $\mathcal{T}_i = \{\tau_i, \tau_{i+1}, \ldots, \tau_n\}$. Therefore, the i-th value of vectors b^l and $\vec{b^u}$ can be expressed as:

$$b_i^l = U_{HPF}(\mathcal{T}_i), \quad b_i^u = U_{LPF}(\mathcal{T}_i), \; i = 1, \ldots, n \tag{19}$$

$$b_{n+1}^l = 0, \quad b_{n+1}^u = 0 \tag{20}$$

where $U_{HPF}(\mathcal{T}_i)$ corresponds to the utilization obtained by the HPF assignment for task set $\mathcal{T}_i$. Similarly, $U_{LPF}(\mathcal{T}_i)$ corresponds to the utilization obtained by the LPF assignment for task set $\mathcal{T}_i$. It is worth noting that the first values of both

vectors, namely b_1^l and b_1^u, correspond to the lower bound and the upper bound of task set $\mathcal{T}$, i.e., $\mathcal{T} = \mathcal{T}_1$. Values $b_{n+1}^l = 0$ and $b_{n+1}^l = 0$ are introduced for valid comparison in the last step of recursion (see line 28 in Alg. 5).

b: USAGE OF EVERY PERIOD IN THE ASSIGNMENT

To ensure that every period value p_j is used in a task assignment at least once, we have to track the number of used period values when constructing a task assignment. Therefore, we introduce the binary vector $\vec{l}$ with values l_j such that:

$$l_j = \begin{cases} 1, \text{ period value } p_j \text{ is assigned at least once} \\ 0, \text{ otherwise} \end{cases} \tag{21}$$

Additionally, d is the number of currently assigned period values, i.e., $d = \sum_{j=1}^{m} l_j$. As all of the period values have to be used at least once, in a valid task assignment d must be equal to m.

Here follows a detailed explanation of the OTA algorithm depicted with Alg. 5. For brevity and ease of representation, variables U_{max}, $\mathbf{X}_{max}$, $\mathbf{X}$, $\vec{l}_j$, and d, flag `feasible`, and bounds $\vec{b^l}$, $\vec{b^u}$ are assumed to be global. U_{max} is the current maximal value obtained for assignment matrix $\mathbf{X}_{max}$, and flag `feasible` indicates the feasibility of the problem. Global variable $\mathbf{X}$ is current assignment matrix. The local variable u represents utilization at step i.

In the first part of the algorithm, i.e., `Task Assignment` function, we use the HPF assignment to determine the lower bound of utilization prior to enumerating all task assignments (lines 2 to 8 in Alg. 5). If the obtained lower bound is greater than one or the assignment is not correct, there is no need for enumeration of task assignments (line 6 in Alg. 5). In this way, we efficiently prune a lot of period sets for which the task assignment is infeasible.

In the second part of the algorithm, i.e., `Task Assignment Step` function, task assignments are enumerated in a recursive manner. In the basic case (lines 12 to 18), we test if the utilization of the current task assignment u is larger than the current maximal value U_{max}, and update the solution $\mathbf{X}_{max}$ accordingly. In other cases, period values are conditionally assigned to the tasks (lines 21 to 36).

At the beginning of the loop, we check if the value p_j is assigned to any task at the previous recursion steps. If l_j is 0 at step i, value p_j is not assigned to any task τ_k such that $k < i$. On the other hand, if value p_j is used for the first time at step i, we set values of l_j and auxiliary variable ul to 1 (lines 22 to 25). The auxiliary variable ul keeps track of "locking" period value p_j at step i. Therefore, at the end of the loop (lines 32 to 35), if p_j was used for the first time at step i, we have to reset ("release") l_j and ul.

Next, we break down two groups of conditions in their respective `if` statements (line 26 and 28). The first `if` statement checks validity of assignment. The first condition, i.e., $p_i^{min} \leq p_j \leq p_i^{max}$ corresponds to the correctness criteria (Def. II.1). In the second condition, $m - d$ is the number of unused periods from the input period set $\vec{p}$, and $n - i$ is the number of tasks to which the period is not assigned. The condition requires that the number of unused periods is less than or equal to the number of tasks to which a period value is not assigned. In other words, if it is not possible to assign every period value in the next recursion steps, p_j cannot be assigned to τ_i. In line 27, period p_j is assigned to task τ_i, i.e., utilization for the next recursion step is incremented by $\frac{C_i}{p_j}$.

The second `if` statement (line 28) serves to test the feasibility and the bounds of the assignment. Condition $u' \leq 1$ ensures feasibility of the assignment. The second condition, i.e., $u' + b_{i+1}^l \leq 1$, checks if the sum of the current utilization and minimal utilization of task subset $\mathcal{T}_{i+1}$ is less than or equal to one. If this condition is false, we know that there is no assignment for which the final utilization will be less than one, because b_{i+1}^l is a lower bound. Similarly, the third condition, i.e., $u' + b_{i+1}^u \geq U_{max}$, checks if the sum of the current utilization and maximal utilization of task subset $\mathcal{T}_{i+1}$ is greater than or equal to current maximal value U_{max}. If this condition is not true, we know that there is no assignment for which the final utilization is greater than the current maximal value U_{max}, because b_{i+1}^u is a upper bound.

The time complexity of the OTA algorithm is evidently exponential in the number of tasks n. In the worst-case, when bounds are ineffective, one of the m period values will be assigned to each task. Therefore, the time complexity of the OTA algorithm is in $O(m^n)$. To find an optimal solution to an UDHPA instance, we have to use the OTA algorithm for each period set obtained using the PE algorithm. Complexity of such an approach is again pseudo-polynomial with regard to p_{max}^{max}, since the time complexity of the PE algorithm is in $O(p_{max}^{max} \log^{m-1}(p_{max}^{max}))$, and exponential with regard to n, since the time complexity of the OTA algorithm is in $O(m^n)$.

VI. EVALUATION

To further investigate the UDHPA problem and our approach, we perform an extensive evaluation of the developed algorithms on synthetically generated task sets. Firstly, we show how the difficulty of the problem changes with regard to utilization, the number of distinct period values, the width of period ranges and the number of tasks in a task set. Furthermore, we show how our approach can be used for UHPA problem instances and compare it with existing approaches. As the parameters used for synthetically generating task sets correspond to the parameters of task sets in motivational scenarios, we show that our approach can be efficiently used in plenty of real-world scenarios. Moreover, we provide a small numerical example that illustrates the benefits of our approach in motivational real-world scenarios.

A. TASK SET GENERATION

We generate task sets using the UUnifast algorithm [27], which is commonly used in measuring the performance of algorithms in real-time systems. Using the UUnifast algorithm we generate utilizations of task sets with regard to p_i^{max} of tasks in a set. Therefore, the target utilization for

Algorithm 5 Optimal Algorithm for the TA Problem (OTA)

Input: $\vec{p}, \mathcal{T}$
Output: (feasible, U_{max}, $\mathbf{X}_{max}$)

function Task Assignment($\vec{p}$, $\mathcal{T}$)
  (correct, U, $\mathbf{X}$) $\leftarrow$ HPF Assignment($\vec{p}$, $\mathcal{T}$)
  feasible $\leftarrow$ false
  $u \leftarrow 0$
  $U_{max} \leftarrow 0$
  if not correct $\vee$ $U > 1$ **then**
    return
  end if
  Task Assignment Step($\vec{p}$, 1, 0)
end function
function Task Assignment Step($\vec{p}$, i, u)
  if $i == n + 1$ **then**
    if $u > U_{max}$ **then**
      feasible $\leftarrow$ true
      $U_{max} \leftarrow u$
      $\mathbf{X}_{max} \leftarrow \mathbf{X}$
    end if
    return
  end if
  $ul \leftarrow 0$
  for j in range 1 to m **do**
    if $l_j == 0$ **then**
      $l_j \leftarrow 1$
      $ul \leftarrow 1$
    end if
    if $p_i^{min} \leq p_j \leq p_i^{max} \wedge m - d \leq n - i$ **then**
      $u' \leftarrow u + \frac{C_i}{p_j}$ ▷ equivalent to $T_i \leftarrow p_j$ or $x_{ij} \leftarrow 1$
      if $u' \leq 1 \wedge u' + b_{i+1}^l \leq 1 \wedge u' + b_{i+1}^u \geq U_{max}$ **then**
        Task Assignment Step($\vec{p}$, $i + 1$, u')
      end if
    end if
    if $ul == 1$ **then**
      $l_j \leftarrow 0$
      $ul \leftarrow 0$
    end if
  end for
end function

UUnifast method corresponds to the lowest utilization of a task set. We refer to this utilization as $U_{min} = \sum_i^n \frac{C_i}{p_i^{max}}$. After the utilizations are generated, we first choose p_i^{max} from the interval $[p_{down}, p_{up}]$ with uniform distribution. Then, based on the parameter σ, we determine the lower bound of the period range for task τ_i as $p_i^{min} = \lceil p_i^{max}\sigma \rceil$. Increasing σ decreases the width of the period range for tasks.

We generate task sets with utilization U_{min} from interval [0.2, 0.9] with an increment of 0.025. Moreover, $p_{down} = 1$, $p_{up} = 2048$, and $\sigma = 0.4$. We generate 1000 task sets per utilization factor, i.e., a total of $29 \cdot 1000 = 29000$ task sets.

TABLE 1. Computing platform specifications.

Processor	Intel(R) Core(TM) i7-7700HQ CPU 2.80GHz
RAM	8.00 GB
Operating system	Linux (64-bit)

Every task set consists of $n = 20$ tasks. These are the default task set generation parameters unless noted otherwise. When it comes to algorithm runtime measurements, it is worth noting that our implementations of algorithms are written in C++. Additionally, the specifications of the computing platform are given in Table 1. Furthermore, we terminated an algorithm if we obtained the utilization value in the interval $[1 - 10^{-7}, 1]$.

B. EVALUATION ON UDHPA INSTANCES

In this section, we evaluate our optimal approach which consists of the PE algorithm (Alg. 2) and the OTA algorithm (Alg. 5) on UDHPA instances with regard to different problem parameters m, σ and n.

Firstly, we evaluate our approach with regard to different number of distinct period values in the solution m. Fig. 1 shows the number of feasible systems for different utilization factors and for a different number of distinct period values in the solution. It can be seen that, for $m = 5$, we obtain the highest number of feasible solutions. For higher values of m, the number drops, and we know from Proposition V.3 that the maximum value of distinct period values for the system to be feasible equals $\log_2 \frac{2048}{1} + 1 = 12$. Since the number of feasible systems for $m > 8, m < 3$ is lower than for $m = 8$, we did not include these graphs in Fig. 1. Moreover, Fig. 2 shows the average number of enumerated period sets with regard to m. It can be seen that, for higher values of different period values m, the number of harmonic period sets is reduced. Thus, increasing the number of different period values reduces the number of potentially feasible systems. On the other hand, although the number of enumerated period sets is higher for lower m in Fig. 2, feasibility is reduced for lower m as it is more difficult to find a lower number of harmonic period values that satisfy the correctness criteria (Def. II.1) for each task in the system.

Fig. 3 shows the average resulting utilization for each utilization factor and a different number of distinct period values. In this particular evaluation, we set the resulting utilization of infeasible systems to zero. In this way, we do not lose information about the overall feasibility of systems. Again, it can be seen that, for $m = 5$, we have the best result. Dashed lines in the figure denote the utilization lower bound obtained by the HPF assignment for the corresponding number of distinct period values.

Fig. 4 shows the average utilization of feasible systems. Here, we averaged the resulting utilization of feasible systems for every m in interval [3, 8]. Additionally, Fig. 4 shows the lower bound and the upper bound obtained by the HPF and the LPF assignment, respectively. It can be seen that the resulting utilization of our optimal approach is very close to the upper

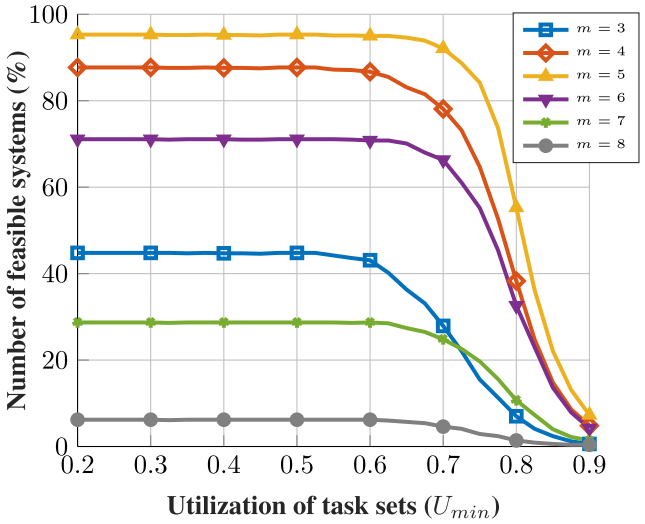


FIGURE 1. Number of feasible systems for different number of distinct period values *m*.

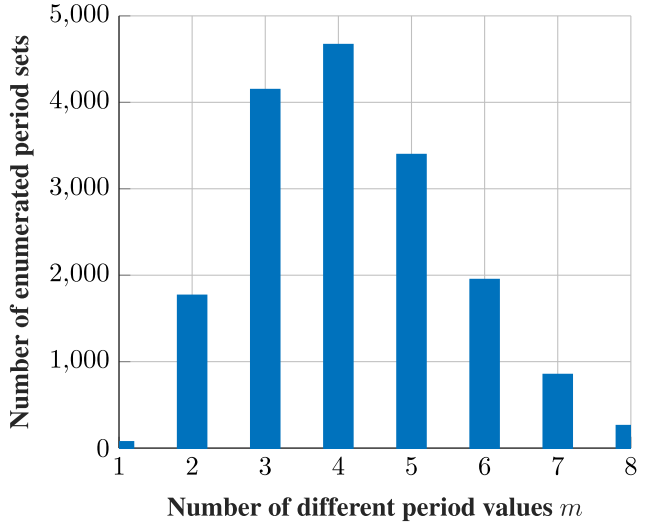


FIGURE 2. Number of period sets obtained in period enumeration w.r.t. *m*.

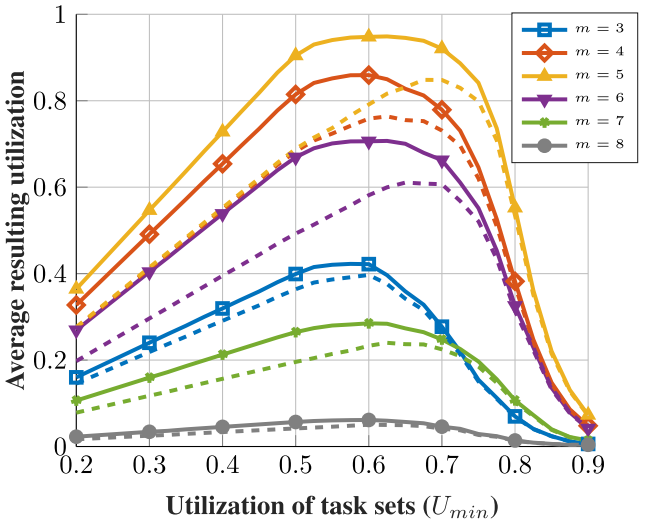


FIGURE 3. Average resulting utilization of task set for different number of distinct period values *m*.

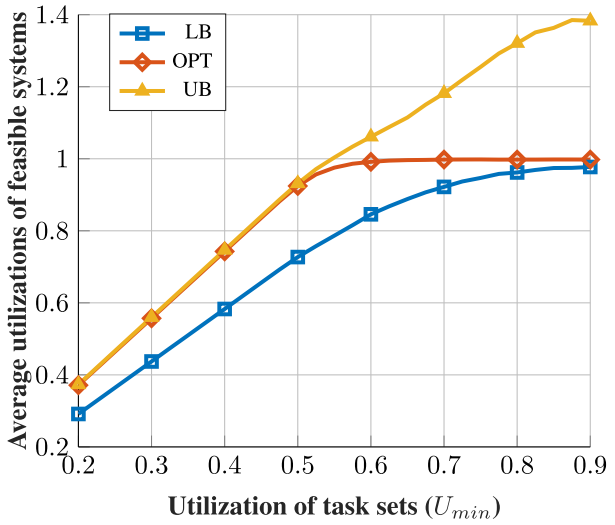


FIGURE 4. Average resulting utilization of the optimal assignment with corresponding bounds.

bound for lower utilization values. When the upper bound is higher than 1, our optimal approach yields the highest possible utilization values lower or equal to 1.

Fig. 5 shows the average runtime per task set of our implementation of the algorithm for each utilization factor and a different number of distinct period values. It can be seen that, on average, the algorithm for larger numbers of different period values has a higher runtime. This is mostly due to a higher number of potential task assignments in solving the TA problem. Additionally, it is worth noting that the average runtime for $m = 8$ is lower than for $m = 7$ and $m = 6$. For $m = 8$, there is a smaller number of enumerated period sets, and therefore fewer TA problem instances have to be solved.

Figs. 6-8 show the effect of the period width σ on the overall feasibility, utilization and average runtime. We use the same parameters in task set generation as in the previous evaluation. However, in this case, the number of distinct period values is fixed, i.e., $m = 5$, and the period range width values σ from interval [0.2, 0.7] are used. Additionally, 100 task sets for each utilization factor and for each σ are generated, which totals to $29 \cdot 6 \cdot 100 = 17400$ task sets.

Fig. 6 shows the feasibility for each utilization factor and for different period widths. It can be seen that, for higher σ, i.e., lower period range width, feasibility drops significantly. This is to be expected as the correctness constraints are more strict and there is less chance of finding a potentially feasible harmonic period set.

In Fig. 7, we again show the average resulting utilization per utilization factor assuming that the utilization of infeasible system equals zero. Larger period range width increases the number of feasible enumerated period sets, and consequently utilization is higher as the task assignments with high utilization can be discovered.

In Fig. 8, we see, that for a larger period range width, i.e., $\sigma = 0.2$, runtime is of an order of magnitude higher

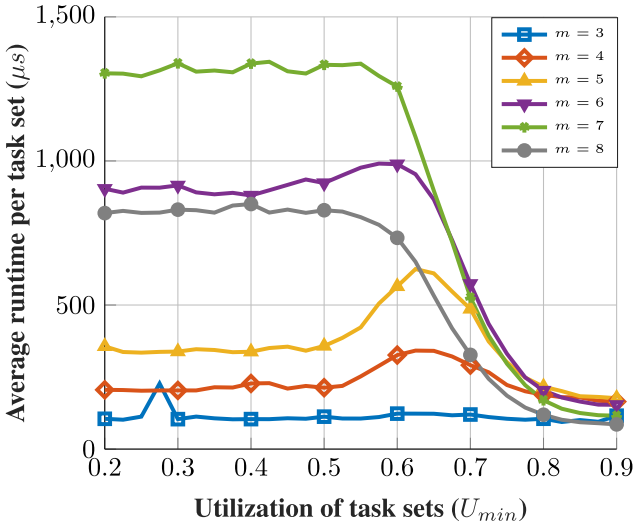


FIGURE 5. Average runtime of the optimal assignment for different number of distinct period values m.

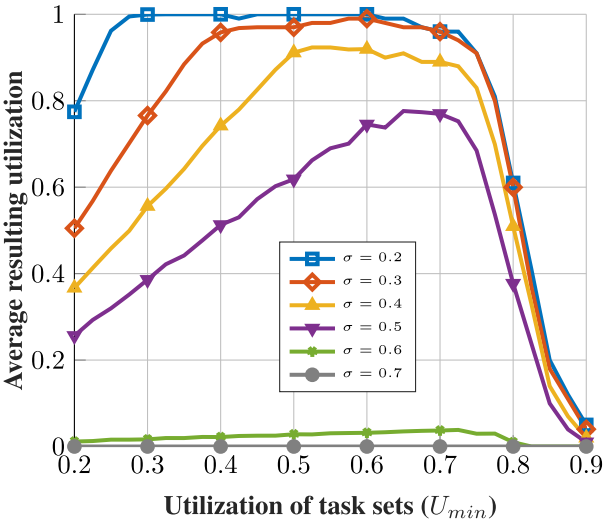


FIGURE 7. Average resulting utilization for different period range width σ.

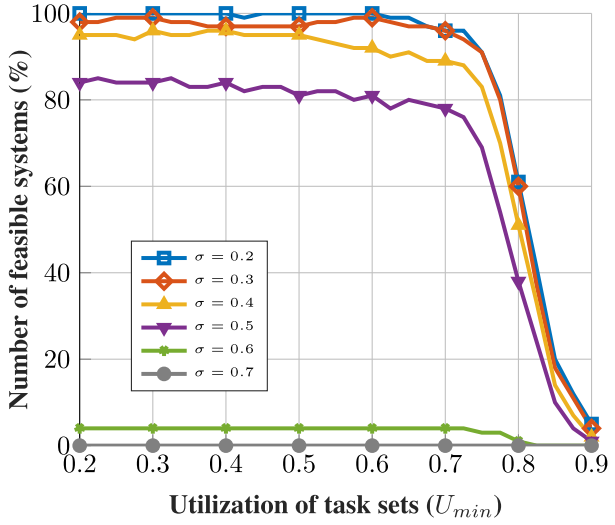


FIGURE 6. Number of feasible systems for different period ranges width σ.

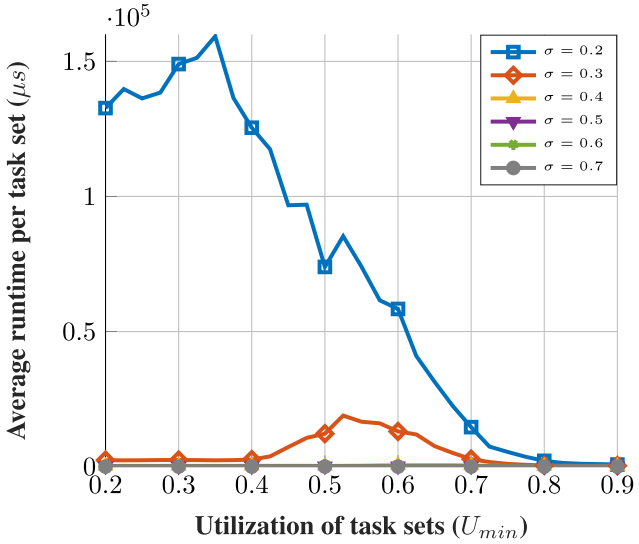


FIGURE 8. Average runtime of the optimal assignment for different period range width σ.

than a smaller period range width. As we already mentioned, the number of feasible solutions is higher for larger period range width and more enumerated period sets have to be explored. Thus, the runtime is increased. It is worth noting that runtime graphs for $\sigma \in [0.4, 0.7]$ cannot be distinguished, as they are much lower than the average runtime for $\sigma = 0.2$.

Finally, we perform the evaluation of our optimal algorithm with regard to the size of a task set. In the previous evaluation, we have fixed the number of tasks in a set to 20. For the purpose of this evaluation, we have fixed the period range width, i.e., $\sigma = 0.4$ and the number of distinct period values, i.e., $m = 5$. For Figs. 9-10, we have generated 200 task sets for each utilization factor and for each $n \in [20, 30, 40, 50]$, which totals to $29 \cdot 4 \cdot 200 = 23200$ task sets. Fig. 9 shows the number of feasible task sets with regard to n. It can be seen that, by increasing n, feasibility drops. This is the effect of adding more period range constraints in the systems for each task. Intuitively, it will be more difficult to find an appropriate period assignment with a higher number of constraints. For the same reason, the total average utilization is reduced when n is increased as depicted in Fig. 10. For the runtime evaluation with regard to n, we have set the utilization factor to $U_{min} = 0.6$ and generated 200 task sets for every fifth number of tasks in range [20, 100], i.e., a total of $200 \cdot 17 = 3400$ task sets. In Fig. 11, the PE + OTA graph corresponds to our optimal approach. The PE + EXH graph corresponds to an approach which consists of the PE algorithm (Alg. 2) and an exhaustive search of task assignments, which does not employ utilization bounds and pruning rules devised in section V-B2. We can see that the runtime of exhaustive search rises exponentially with the number of tasks, which is an expected behavior, since, at worst, time complexity is in $O(m^n)$ (evaluation is not performed for $n \geq 70$). On the other hand, we see that the runtime of the PE + OTA approach has

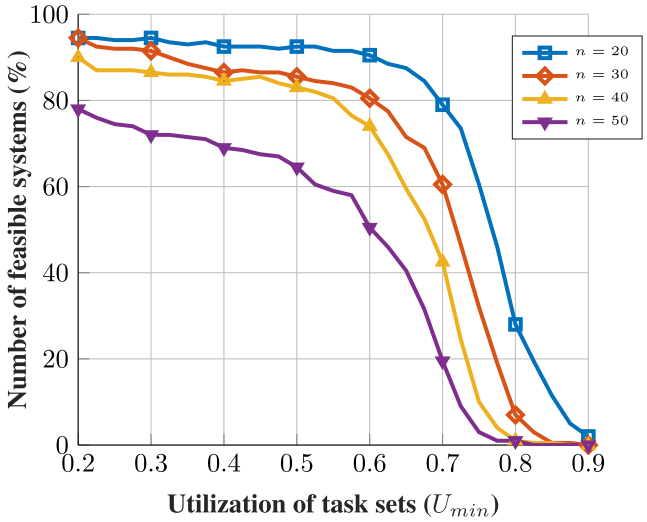


FIGURE 9. Number of feasible systems for different number of task in system *n*.

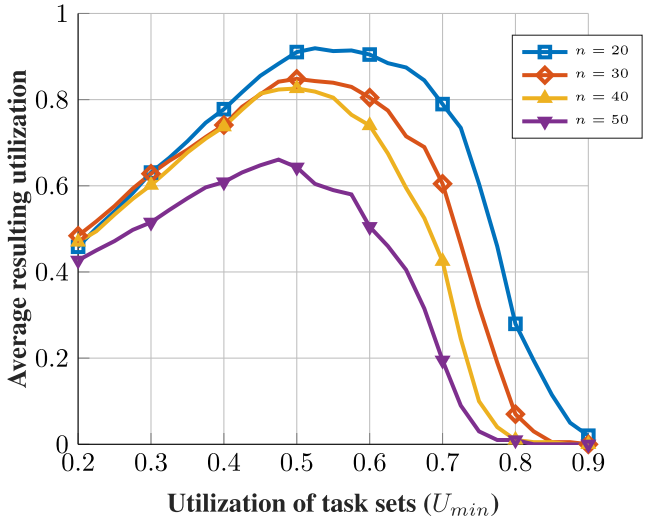


FIGURE 10. Average resulting utilization of task set for different number of tasks in system *n*.

a reduced growth rate due to the usage of devised utilization bounds and pruning rules.

C. EVALUATION IN THE CONTEXT OF EXISTING UHPA APPROACHES

On the basis of Theorem IV.1, we know that the UHPA problem is Turing reducible to the UDHPA problem, and therefore we can simply employ our optimal algorithm for UHPA instances. To solve an UHPA instance, we need to solve the corresponding UDHPA instances for every possible number of distinct period values m, which is given with Proposition V.3. To optimally solve an UHPA instance, we use the PE algorithm in combination with the OTA algorithm for each m. In the UHPA problem, there is no restriction on the number of distinct period values. Thus, we can use the PE algorithm with the HPF algorithm to obtain the period assignment for each m. It is worth noting that it is possible that, while using the HPF algorithm, some values

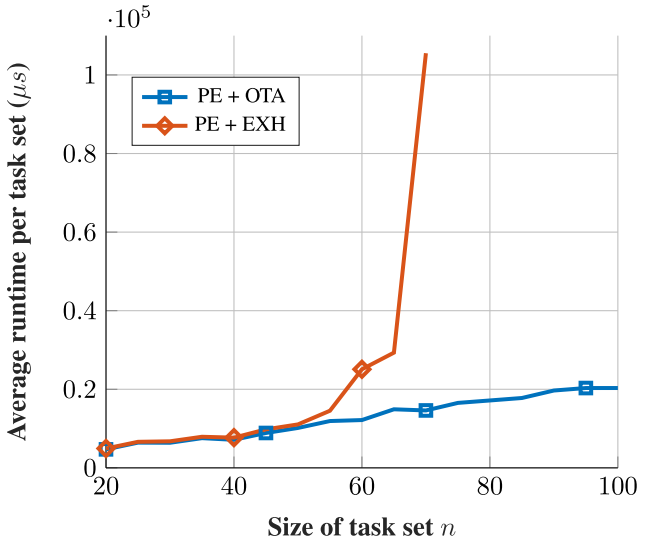


FIGURE 11. Average runtime of the optimal assignment for different size of task set *n*.

of the enumerated period set may remain unused. However, this is not a problem in the context of UHPA instances, since there are no restrictions on the number of different period values. We compare our approach with existing UHPA approaches in the literature, which are based on finding harmonic projections for given task period ranges [10], [11]. The algorithms employed in these approaches are generally of pseudo-polynomial time complexity, but in specific cases complexity can be reduced to linear or polynomial time. The approach in [10], referred to as *forward search*, consists of two parts, namely the graph construction algorithm (GCA), and a greedy heuristic for period assignment, which can yield low utilization (LU) or high utilization (HU). Here, we employ the LU heuristic as it increases the chance that the resulting harmonic period assignment will be feasible. The GCA part is analogous to our period enumeration part of the algorithm. Similarly, the HPF and OTA algorithms are counterparts to the LU heuristic. The approach from [11], referred to as *backward search*, is based on the harmonic period existence test and suboptimal heuristic period assignment. Figs. 12-16 show the performance of our approach in comparison with approaches from [10], [11]. We have generated task sets using the default task set generation parameters from the beginning of this section.

Fig. 12 shows the number of feasible systems for different period assignment approaches. It can be seen that the number of feasible task sets is higher when our optimal approach (PE + OTA) and heuristic approach (PE + HPF) are used, than when *forward search* or *backward search* are used. The number of feasible systems when the PE + HPF or PE + OTA approaches are used is the same because both algorithms are optimal regarding feasibility. However, the HPF algorithm yields the solution with the lowest utilization. More precisely, it yields the lowest utilization for the TA* problem. As explained in section V-B1, the TA* problem does not restrict the number of different period values in the

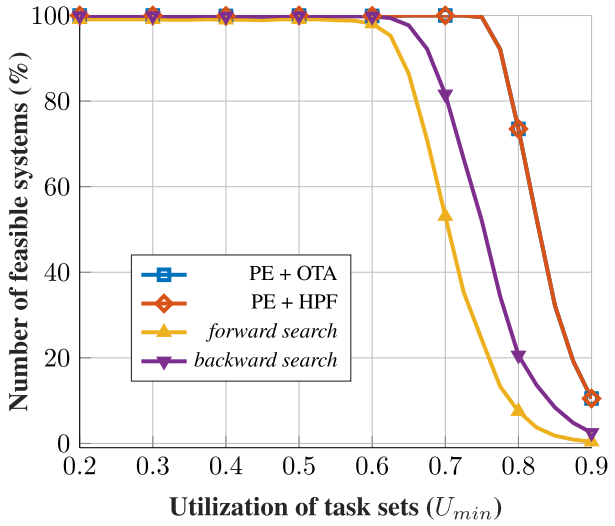


FIGURE 12. Number of feasible systems for UHPA instances for different period assignment approaches.

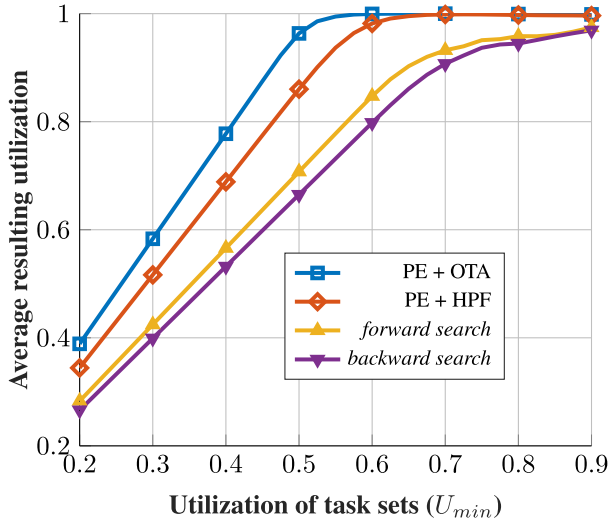


FIGURE 13. Average resulting utilization for different period assignment approaches.

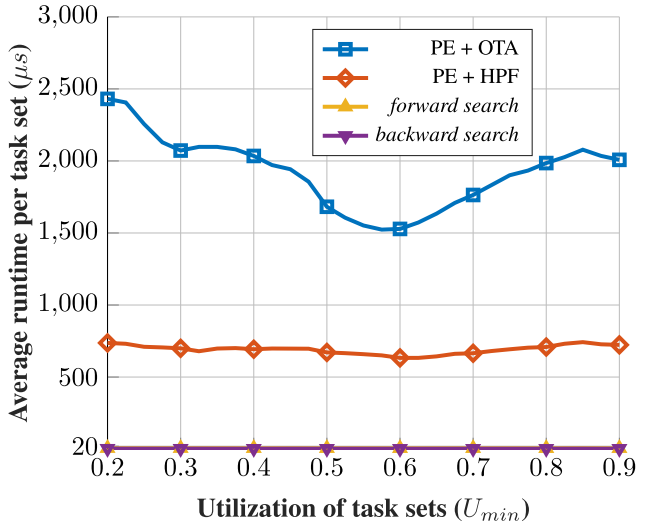


FIGURE 14. Average runtime for different period assignment approaches w.r.t. utilization.

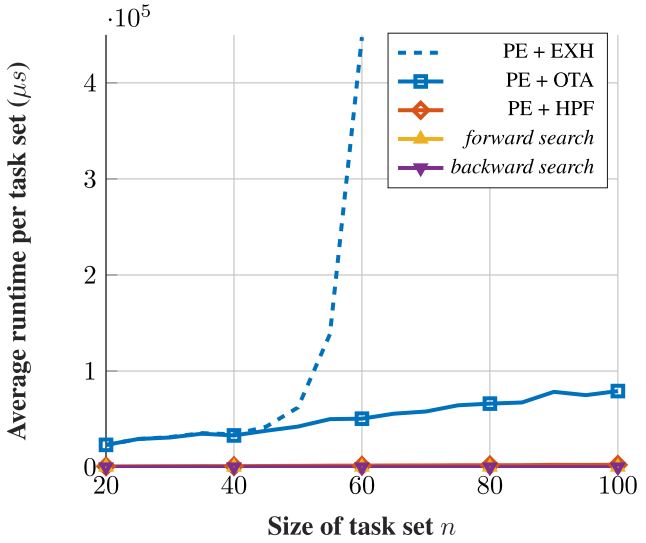


FIGURE 15. Average runtime for different period assignment approaches w.r.t. size of task set.

solution, and therefore the utilization obtained using the HPF assignment is minimal. Fig. 13 shows the resulting utilization for different period assignment approaches. As expected, our optimal approach (PE + OTA) yields the highest utilization. Moreover, the period enumeration with the HPF assignment (PE + HPF) dominates *forward search* and *backward search* as well.

Finally, we present the runtime results in Figs. 14-16. In Fig. 14, it can be seen that the runtime for *forward search* and *backward search* is at least an order of magnitude lower than in our approach. Since the period ranges are relatively wide, i.e., $\sigma = 0.4$, both *forward* and *backward search* are time-efficient. Moreover, in Fig. 15, we can see that when the number of tasks in a set is increasing, runtime is higher for exhaustive search (PE + EXH) and our optimal approach (PE + OTA) than for the other approaches which cannot be distinguished in the figure. The PE + HPF approach is efficient when the number of tasks is increasing, since the time complexity of the HPF algorithm is polynomial $O(n \cdot m)$ and the time complexity of the PE algorithm does not depend on the number of tasks in the system. However, in Fig. 16, we can see that the runtime of approaches which employ PE algorithm increases when the maximum period in the system p_{max}^{max} is increased. It is worth noting that although runtime can be significantly higher when our optimal approach (PE + OTA) and the heuristic approach (PE + HPF) are used, it is still relatively low, i.e., several milliseconds per task set. Since period assignment in practice is typically done off-line during the application design, this is more than acceptable.

D. NUMERICAL REAL-WORLD PERIOD ASSIGNMENT PROBLEM

To further emphasize and explain the benefits of our approach, we provide a numerical example of a small

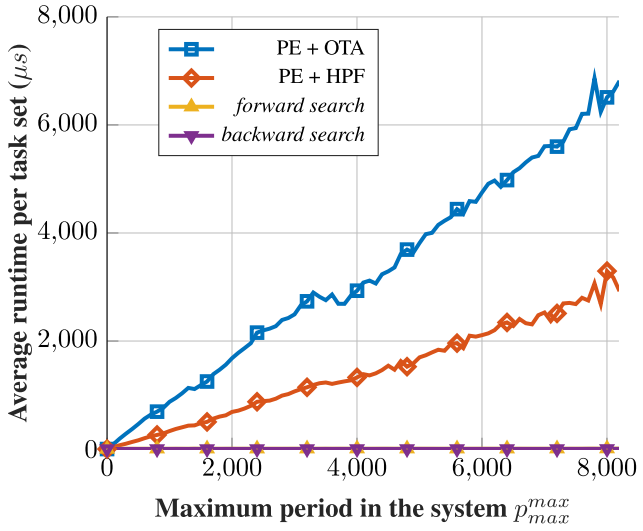


FIGURE 16. Average runtime for different period assignment approaches w.r.t. maximum period in the system p_{max}^{max}.

TABLE 2. Task parameters for the application task set.

Task	C_i	p_i^{min}	p_i^{max}
τ_1	1	2	5
τ_2	2	5	16
τ_3	2	13	42
τ_4	1	21	68
τ_5	13	36	118
τ_6	3	38	124

real-world period assignment problem. As we have already stated in the introduction, the structure of safety-critical control applications is modular and often each module, i.e., task, is developed by a different application designer. In the development process, based on the specific application requirements, application designers provide implementations of tasks with suggested execution rates, which are in this paper and related literature modelled with period ranges. It is in the interest of every application designer that their module executes with the highest possible execution rate, i.e., the smallest period, in order to achieve a higher quality of service for a particular part of the application. Based on the input from the application designers, the system designer has to determine periods which shall be used in the system to achieve the highest utilization, i.e., quality of service. Thus, utilization is maximized. Table 2 shows the task set with task parameters. Such a table is an input to the system designer. However, due to the specific architecture of the system, i.e., the operating system and the underlying hardware, the system designer is restricted regarding the number of distinct period values which can be used in the solution. In this example, the maximum number of different period values is 4. Therefore, any number of different period values smaller or equal to $m_{max} = 4$ can be used. Table 3 shows the period values assigned to each task in the input. We can see that period enumeration with the optimal task assignment (PE + OTA) yields the most satisfying result, since it produces the maximal utilization and uses no more than 4 period values. In order

TABLE 3. Assigned periods, the number of distinct period values and resulting utilization per period assignment approach.

	Assigned periods T_i per approach			
Task	PE + OTA	PE + HPF	*forward*	*backward*
τ_1	2	5	3	5
τ_2	14	5	15	15
τ_3	14	20	30	30
τ_4	42	60	60	60
τ_5	84	60	60	60
τ_6	84	60	120	120
m	**4**	**3**	5	5
U	**1.000**	**0.983**	0.791	0.658

to achieve this, the system designer has to solve UDHPA instances using the PE + OTA approach for m in the interval $[1, m_{max} = 4]$. We can see that period enumeration using the HPF algorithm (PE + HPF) also yields a satisfying result as the number of the distinct period values is lower than m_{max}. However, the utilization is lower than the value obtained using the OTA algorithm. The *forward* and the *backward search* do not yield satisfying results, since they do not restrict the number of distinct period values in the solution. Moreover, utilization factors are significantly lower than when using both the PE + OTA and the PE + HPF approaches.

VII. CONCLUSION

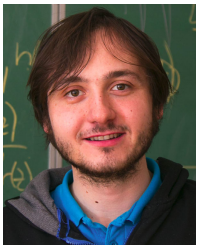
In this paper, we have defined the utilization-maximizing harmonic period assignment problem with a constrained number of distinct period values (UDHPA). The motivation for this problem arises from the observation of industrial control systems in which the number of different period values is either fixed or restricted. We have put the problem in the context of the already studied UHPA problem and show that the UHPA problem is Turing reducible to the UDHPA problem. Additionally, we show that UDHPA problem is at least weakly NP-hard. We have devised an optimal algorithm for the UDHPA problem and showed its efficiency on a large number of synthetically generated task sets. Moreover, we have used the developed algorithm for solving UHPA instances and explained the differences between our approach and the existing approaches. The key benefit of our approach lies in the fact that, for a large variety of synthetically generated UDHPA and UHPA instances, our algorithm is time-efficient and optimal, and therefore more than suitable for use in real-world system design. Furthermore, as we have shown on a numerical example, the existing approaches are not applicable in systems with a restricted number of different period values. Our future work will include the application of our algorithm to real-world scenarios and a further theoretical analysis of the relation between the optimal number of distinct period values and period ranges of tasks.

REFERENCES

[1] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Norwell, MA, USA: Kluwer, 1997.

[2] D. Seto, J. P. Lehoczky, and L. Sha, "Task period selection and schedulability in real-time systems," in *Proc. 19th IEEE Real-Time Syst. Symp.*, Dec. 1998, pp. 188–198.

[3] E. Bini and M. Di Natale, "Optimal task rate selection in fixed priority systems," in *Proc. 26th IEEE Int. Real-Time Syst. Symp. (RTSS)*, Dec. 2005, p. 11.
[4] C.-C. Han and H.-Y. Tyan, "A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms," in *Proc. Real-Time Syst. Symp.*, 1997, pp. 36–45.
[5] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a Hard-Real-Time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.
[6] C.-C. Han, K.-J. Lin, and C.-J. Hou, "Distance-constrained scheduling and its applications to real-time systems," *IEEE Trans. Comput.*, vol. 45, no. 7, pp. 814–826, Jul. 1996.
[7] V. Bonifaci, A. Marchetti-Spaccamela, N. Megow, and A. Wiese, "Polynomial-time exact schedulability tests for harmonic real-time tasks," in *Proc. IEEE 34th Real-Time Syst. Symp.*, Dec. 2013, pp. 236–245.
[8] P. Ekberg and W. Yi, "Fixed-priority schedulability of sporadic tasks on uniprocessors is NP-hard," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2017, pp. 139–146.
[9] M. Mohaqeqi, M. Nasri, Y. Xu, A. Cervin, and K.-E. Årzén, "Optimal harmonic period assignment: Complexity results and approximation algorithms," *Real-Time Syst.*, vol. 54, no. 4, pp. 830–860, Oct. 2018.
[10] M. Nasri, G. Fohler, and M. Kargahi, "A framework to construct customized harmonic periods for real-time systems," in *Proc. 26th Euromicro Conf. Real-Time Syst.*, Jul. 2014, pp. 211–220.
[11] M. Nasri and G. Fohler, "An efficient method for assigning harmonic periods to hard real-time tasks with period ranges," in *Proc. 27th Euromicro Conf. Real-Time Syst.*, Jul. 2015, pp. 149–159.
[12] J. Xu, "A method for adjusting the periods of periodic processes to reduce the least common multiple of the period lengths in real-time embedded systems," in *Proc. IEEE/ASME Int. Conf. Mech. Embedded Syst. Appl.*, Jul. 2010, pp. 288–294.
[13] I. Ripoll and R. Ballester-Ripoll, "Period selection for minimal hyperperiod in periodic task systems," *IEEE Trans. Comput.*, vol. 62, no. 9, pp. 1813–1822, Sep. 2013.
[14] Y. Xu, A. Cervin, and K.-E. Arzen, "Harmonic scheduling and control codesign," in *Proc. IEEE 22nd Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2016, pp. 182–187.
[15] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén, "Feedback–feedforward scheduling of control tasks," *Real-Time Syst.*, vol. 23, pp. 25–53, Jul. 2002.
[16] E. Bini and A. Cervin, "Delay-aware period assignment in control systems," in *Proc. Real-Time Syst. Symp.*, Nov. 2008, pp. 291–300.
[17] C.-S. Shih, S. Gopalakrishnan, P. Ganti, M. Caccamo, and L. Sha, "Scheduling real-time dwells using tasks with synthetic periods," in *Proc. Int. Symp. Syst.-on-Chip*, 2003, pp. 210–219.
[18] H. Li, J. Sweeney, K. Ramamritham, R. Grupen, and P. Shenoy, "Real-time support for mobile robotics," in *Proc. 9th IEEE Real-Time Embedded Technol. Appl. Symp.*, 2003, pp. 10–18.
[19] A. Easwaran, I. Lee, O. Sokolsky, and S. Vestal, "A compositional scheduling framework for digital avionics systems," in *Proc. 15th IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2009, pp. 371–380.
[20] S. Anssi, S. Kuntz, S. Gérard, and F. Terrier, "On the gap between schedulability tests and an automotive task model," *J. Syst. Archit.*, vol. 59, no. 6, pp. 341–350, Jun. 2013.
[21] (2019). *Ansys Scade Suite: Model-Based Development*. Accessed: May 25, 2019. [Online]. Available: https://www.ansys.com/products/embedded-software/ansys-scade-suite
[22] (2019). *Silworx: Simplify Safety Engineering*. Accessed: Jun. 17, 2019. [Online]. Available: https://www.hima.com/en/products-services/silworx
[23] (2019). *Grap-Ide-Powerful Graphical Programming, Service and Diagnostic Tool*. Accessed: May 25, 2019. [Online]. Available: https://www.koncar-institut.hr/en/?solution_group=programming-tools
[24] I. Lee, J. Y.-T. Leung, and S. H. Son, *Handbook of Real-Time and Embedded Systems*, 1st ed. London, U.K.: Chapman & Hall, 2007.
[25] I. P. Gent and T. Walsh, "Analysis of heuristics for number partitioning," *Comput. Intell.*, vol. 14, no. 3, pp. 430–451, Aug. 1998. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/0824-7935.00069
[26] A. Lundell and T. Westerlund, "Global optimization of mixed-integer signomial programming problems," in *Mixed Integer Nonlinear Programming*, J. Lee and S. Leyffer, Eds. New York, NY, USA: Springer, 2012, pp. 349–369.
[27] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Syst.*, vol. 30, nos. 1–2, pp. 129–154, May 2005.



IVAN PAVIĆ (Member, IEEE) received the B.S. and M.S. degrees in electrical and computer engineering from the Faculty of Electrical Engineering and Computing, University of Zagreb, in 2015 and 2017, respectively, where he is currently pursuing the Ph.D. degree. He also works as a Young Researcher with the Department of Electronic Systems and Information Processing, Faculty of Electrical Engineering and Computing, University of Zagreb. His research interests include real-time systems and embedded system design.

HRVOJE DŽAPO (Senior Member, IEEE) received the Dipl.-Eng. and Ph.D. degrees in electrical engineering from the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia, in 1999, and 2007, respectively. He is currently an Associate Professor with the Department of Electronic Systems and Information Processing, Faculty of Electrical Engineering and Computing, University of Zagreb. His research interests include the fields of electronic instrumentation, sensors, embedded systems, and computational electromagnetics.