Efficient Feasibility Analysis for Graph-Based Real-Time Task Systems

Jinghao Sun, Rongxiao Shi, Kexuan Wang, Nan Guan, *Member, IEEE*, and Zhishan Guo

***Abstract*—The demand bound function (DBF) is a powerful abstraction to analyze the feasibility/schedulability of real-time tasks. Computing the DBF for expressive system models, such as graph-based tasks, is typically very expensive. In this article, we develop new techniques to drastically improve the DBF computation efficiency for a representative graph-based task model, digraph real-time tasks (DRT). First, we apply the well-known quick processor-demand analysis (QPA) technique, which was originally designed for simple sporadic tasks, to the analysis of DRT. The challenge is that existing analysis techniques of DRT have to compute the demand for each possible interval size, which is contradictory to the idea of QPA that aims to aggressively skip the computation for most interval sizes. To solve this problem, we develop a novel integer linear programming (ILP)-based analysis technique for DRT, to which we can apply QPA to significantly improve the analysis efficiency. Second, we improve the task utilization computation (a major step in DBF computation for DRT) efficiency from pseudo-polynomial complexity to polynomial complexity. Experiments show that our approach can improve the analysis efficiency by dozens of times.**

***Index Terms*—Demand bound function (DBF), digraph real-time tasks (DRT), feasibility, linear program (LP).**

I. Introduction

Real-time systems are often implemented by a number of concurrent tasks sharing hardware resources, in particular the execution processors. Traditionally, a real-time task system is modeled as a collection of periodically or sporadically repeating computation requests [1], [2]. With the increased complexity of real-time embedded software, complex control flow structures, such as mode switches, local loops, and `if-else` branches, cannot be fully captured by these simple periodic and sporadic task models. A natural representation of these complex structures is the *task graph*, where vertices represent different types of jobs, and edges represent the possible flow of control. Each vertex (job) is characterized by its worst-case execution time (WCET) requirement and relative deadline. Each edge is labeled with the minimum separation time between the release of the two vertices (jobs) it connects. The first graph-based real-time task model is the recurring branching task model which is proposed to formulate some restricted forms of conditional real-time process code [3]. Over years, more and more expressive graph-based task models are proposed to precisely describe complex embedded real-time systems [3]–[10], and the digraph real-time (DRT) task model [10] is a representative one among them, which generalizes most known real-time task models.

When scheduling a real-time task system on a target execution platform, we concern with a fundamental analysis problem, called the *feasibility analysis*, i.e., "*how do we determine whether the tasks can be scheduled in such a manner that all jobs complete by their deadlines?*". In this article, we restrict our attention to *pre-emptive scheduling* (i.e., a job executing on the processor can be interrupted at any instant in time, and its execution resumes later with no cost or penalty) of the DRT tasks. The *demand bound function (DBF) analysis* technique is a standard methodology for the feasibility analysis of real-time tasks under pre-emptive scheduling, which is centered upon the idea of the DBF for tasks: this quantifies the maximum amount of processor time that all jobs generated by the task can require in an interval of specified size, and attempt to determine whether there is an interval size for which the DBF summed over all tasks in the system exceeds the processor capacity (i.e., the *overflow* occurs). Briefly, the computation of DBF for DRT tasks is conducted in the following two steps.

Step 1: Compute the utilization (e.g., the amount of processor demand required by a task per time unit) for each task and determine an interval-size bound L by using the utilization summed over all tasks.

Step 2: For each interval size less than L, compute the DBF for each task, and then sum the DBF over all tasks, and check whether the summed DBF (i.e., the maximum processor demand required by all tasks) exceeds this interval size (i.e., the processor capacity).

The DBF methodology is originally proposed for analyzing the simplest sporadic tasks (e.g., [2]), and further is adapted to analyze the DRT task model [10]. When analyzing the simplest sporadic tasks, the DBF method is quite efficient, i.e., the bound L and the DBF for any specified interval size can be computed within a constant time. The only problem left

Manuscript received April 17, 2020; revised June 17, 2020; accepted July 6, 2020. Date of publication October 2, 2020; date of current version October 27, 2020. This work was supported in part by the National Foundation of Science of China under Grant 61972076, Grant U1808206, and Grant 61672140; in part by the Research Grants Council of Hong Kong under Grant GRF 15204917 and Grant 15213818; and in part by the National Science Foundation under Grant CNS-1850851. This article was presented in the International Conference on Embedded Software 2020 and appears as part of the ESWEEK-TCAD special issue. *(Corresponding author: Nan Guan.)*

Jinghao Sun is with the School Of Computer Science and Technology, Dalian University of Technology, Dalian 116024, China.

Rongxiao Shi and Kexuan Wang are with the School Of Computer Science and Engineering, Northeastern University, Shenyang 110819, China.

Nan Guan is with the Department of Computing, Hong Kong Polytechnic University, Hong Kong (e-mail: nan.guan@polyu.edu.hk).

Zhishan Guo is with the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL 32816 USA.

Digital Object Identifier 10.1109/TCAD.2020.3012174

is that the bound L has a pseudo-polynomial scale, and by following the traditional DBF methodology, it is inefficient to check all interval sizes (less than L). To this end, Zhang and Burns [11] proposed an efficient technique, called the quick processor-demand analysis (QPA), to accelerate the traditional DBF analysis for the simplest sporadic tasks. The main benefit of the QPA technique is that QPA does not require to do analysis for every interval size, but only needs to check few interval sizes for which the overflow may occur. With the help of QPA, the calculation effort (of analyzing the simplest sporadic task set) is exponentially reduced [11].

However, when the DBF method is applied to analyze the DRT task model, it suffers a sharply increased complexity, and it is challenging to directly adapt QPA (for simple sporadic tasks) to accelerate the DBF analysis of DRT tasks, due to the fact that the parameters (e.g., task utilization and DBF) that are frequently calculated during the analysis progress are more complicated to compute, and more specifically, we have the following.

1) *DRT Task's Utilization Is More Complicated to Compute:* In the simplest sporadic task model, the utilization of a task can be computed within $O(1)$ time, but in the DRT task model, computing the task utilization is equivalent to solving the densest cycle of the task graph. The traditional method in [10] solves this problem in pseudo-polynomial time.
2) *DRT Task's DBF Is More Complicated to Compute, and the Traditional Method in [10] Cannot Fit QPA:* The DBF for a simple sporadic task can be represented as a formula, which is computed within $O(1)$ time. However, computing DBF for a DRT task is equivalent to searching an optimal path on the task graph with the maximum workload and bounded length. The traditional method in [10] uses a dynamic programming (DP)-based approach to deal with the exponential path explosion during the graph search progress, which relies on sequentially checking all interval sizes less than L and thus is contradictory with the idea of QPA.

Contributions: In this article, we aim to solve the above challenging problems, and as the main contribution of this article, we propose efficient acceleration techniques for the DBF analysis of the DRT task model. More specifically, we accelerate the DBF analysis in two parts.

1) We integrate the QPA framework into the analysis of DRT tasks. Instead the traditional DP method, we propose an integer linear programming (ILP) model to directly solve the DBF, which does not need to sequentially check all interval sizes, and thus this makes it possible to apply the QPA framework for analyzing DRT tasks. To make our ILP model more efficient to solve, we divide the constraints of our ILP model into two parts: a) easy constraints and b) complex constraints. We develop a row generation algorithm to iteratively solve our ILP model which initially only involves easy constraints, and dynamically adds complex constraints when necessary.
2) We find that the task utilization computation that is solved by traditional method in [10] within pseudo-polynomial time actually has a polynomial time complexity. We develop a linear programming (LP) approach to solve the densest cycle of a task graph, and thus derive the task utilization within polynomial time, which clearly outperforms the traditional method and further accelerates the analysis progress.

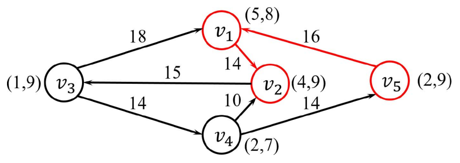

Fig. 1. Example DRT task model.

We conduct experiments with randomly generated task systems to evaluate the efficiency of our proposed method. The experimental results show that our method is much faster than the traditional method in [10].

The motivation for accelerating the DBF analysis is twofold. The first requirement comes from online systems. During the runtime of a real-time system, new tasks may arrive, and need to be added to the current task set. The system must reanalyze feasibility online to decide whether to allow the new tasks to enter into the system or not. Such online admission control gives a stronger requirement on the efficiency of the feasibility test as the decisions have to be made in a very short time. Second, efficient analysis is very useful in design-space exploration of real-time systems, in which many different parameter profiles need to be checked. An automated search may even be undertaken as part of the architectural definition of the system. An efficient but accurate feasibility scheme is therefore needed.

II. Task Model

We consider a task system $\mathcal{T}$ consisting of a set of n independent DRT tasks, i.e., $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$. A DRT task τ_k is represented by a directed graph $G_k = (V_k, E_k)$, where V_k denotes the set of vertices in G_k, and E_k denotes the set of edges in G_k. The vertices in V_k represent the types of all jobs that can be released by τ_k. Each vertex v_i of V_k has the WCET $C_i \in \mathbb{N}$, and the relative deadline $D_i \in \mathbb{N}$. The edges in E_k represent the order in which τ_k's job instances are released. Each edge $(v_i, v_j) \in E_k$ is labeled with $p_{ij} \in \mathbb{N}$ denoting the minimum inter-release separation time (or equally, the period) between vertices v_i and v_j. We assume that the task system $\mathcal{T}$ satisfies the frame separation property [6] by which all jobs' deadlines are constrained to not exceed the inter-release separation times: for all vertices v_i and their outgoing edges (v_i, v_j), we require $D_i \le p_{ij}$. If the task system does not meet the frame separation property, then the techniques proposed in this article will be over-approximate. We discuss this in Section III-A in details. An example DRT task is given in Fig. 1. There are five vertices and seven edges. Each vertex v_i is labeled by a pair of integers, where the first integer represents the execution time of the job released by v_i, and the second integer represents the deadline of the job released by v_i. Each edge (v_i, v_j) is labeled by an integer, representing its period.

Execution Semantics: The semantic of a DRT task system $\mathcal{T}$ is defined as the set of *job sequences* it may generate. In the following, we introduce the job sequences generated by $\mathcal{T}$ in more details. Before going into details, we first give some useful notations as follows. A *job instance* is represented by a tuple (r, e, d) consisting of an absolute release time r, an execution time e, and an absolute deadline d. The job sequence $\sigma_k = [(r_0, e_0, d_0), (r_1, e_1, d_1), \ldots]$ is a sequence of job instances generated by τ_k, which corresponds to a *path* $\pi = (v_0, v_1, \ldots)$ in G_k such that the xth tuple (r_x, e_x, d_x) of σ_k corresponds to the xth vertex v_x of π, and thus, the following equalities hold, i.e., $r_{x+1} - r_x \geq p_{x,x+1}$, $d_x = r_x + D_x$, and $e_x \leq C_x$, for all $x \geq 0$. Combining the job sequences σ_k of individual tasks $\tau_k \in \mathcal{T}$ results in a job sequence $\Sigma_{\mathcal{T}}$ of the whole system $\mathcal{T}$, i.e., $\Sigma_{\mathcal{T}} = \{\sigma_k | \tau_k \in \mathcal{T}\}$.

Fig. 1 shows an example to illustrate the semantics of DRT tasks. The system can start at an arbitrary vertex of G_k, i.e., the task τ_k releases its first job instance of any job type. The released job sequence corresponds to a path through G_k. We consider the job sequence $\sigma_k = [(5, 2, 14), (23, 5, 31), (37, 4, 46)]$ which corresponds to path $\pi = (v_5, v_1, v_2)$ in G_k (the edges in π are marked in red). Note that not every job instance in σ_k is released as early as possible, e.g., the second job instance (associated with v_1) is released 2 time units later than its earliest release time.

III. Feasibility Analysis

The main focus of this work on the DRT task model is to solve the associated feasibility problem defined as follows.

Definition 1 (Feasibility): A task set $\mathcal{T}$ is pre-emptive uniprocessor feasible, if and only if all job sequences $\sum_{\mathcal{T}}$ generated by $\mathcal{T}$ can be executed on a pre-emptive uniprocessor platform such that all jobs meet their deadlines.

In particular, we say a job (r, e, d) is successfully scheduled to meet its deadline, if there is an accumulated duration of e time units where the job executes exclusively on the processor within the time interval $[r, d)$. It is known that the earliest deadline first (EDF) is an optimal algorithm for scheduling real-time tasks on a pre-emptive uniprocessor. Thus, the feasibility problem is equivalent to EDF schedulability.

In the following, we first introduce a general feasible-analysis methodology. After that, we give a brief introduction to the relevant prior work about such a methodology.

A. Demand Bound Function Methodology

As stated in Section I, the DBF technique is a standard methodology for the feasibility analysis of real-time tasks. For DRT tasks, the notion of DBF expresses accumulated execution time that a task set can *demand* from the processor within any time interval of a specified size. Formally, we have the following definition.

Definition 2 (DBF): For any task $\tau_k \in \mathcal{T}$ (with the frame separation property) and an interval size t, $\text{DBF}_k(t)$ denotes the maximum cumulative execution requirement of tasks with both release time and deadline within an interval of the specified interval size t, over all task sequences generated by τ_k. More precisely

$$\text{DBF}_k(t) = \max\{e(\pi) | \pi \in G_k \wedge l(\pi) \leq t\} \tag{1}$$

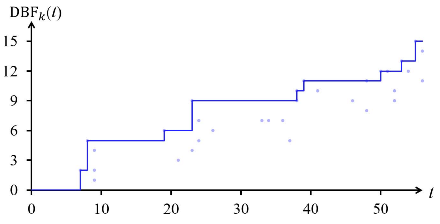

Fig. 2. DBF $\text{DBF}_k(t)$ calculated for the DRT task τ_k in Fig. 1. The dots depict the demand pairs of that task.

where for any path $\pi = (v_1, v_2, \ldots, v_h)$ of G_k, v_x is the xth vertex visited along the path π, $l(\pi)$ is the *length* of path π, i.e., $l(\pi) = \sum_{x=1}^{h-1} p_{x,x+1} + D_h$, and $e(\pi)$ is the *workload* of path π, i.e., $e(\pi) = \sum_{x=1}^{h} C_x$. Further, for a task system $\mathcal{T}$ and a specified interval size t, the DBF $\text{DBF}(t)$ of $\mathcal{T}$ is defined as

$$\text{DBF}(t) = \sum_{k:\tau_k \in \mathcal{T}} \text{DBF}_k(t). \tag{2}$$

The DBF computation formula in (1) may be over-approximate if the frame separation property does not hold. The reason is as follows. If a digraph G_k is not required to meet the frame separation property, the path π of G_k with length less than t may contain a vertex v_i such that the job released by v_i has relative deadline larger than t. In this case, the workload of the job released by v_i should not contribute to $\text{DBF}_k(t)$, but it is accounted for in $\text{DBF}_k(t)$ according to (1). Therefore, $\text{DBF}_k(t)$ calculated by (1) may be larger than the exact one if the frame separation property does not hold. Formula (1) plays a very important role in developing our schedulability analysis methods (see Section III), and this is why we require the frame separation property in this article. Moreover, the definition of $\text{DBF}(t)$ as the sum of $\text{DBF}_k(t)$ of all tasks τ_k relies on their independence of each other. The definition of DBF is *tight* since for each interval size t, there is a job sequence $\Sigma_{\mathcal{T}}$ generated by task set $\mathcal{T}$ in which some jobs actually require a processor demand of $\text{DBF}(t)$ within an interval of t time units. Since we assume discrete time, i.e., DBF $\text{DBF}(t)$ is defined for all $t \in \mathbb{N}$, changes clearly only occur at integers.

Example 1: Consider again job sequence $\sigma_k = [(5, 2, 14), (23, 5, 31), (37, 4, 46)]$ which corresponds to path $\pi = (v_5, v_1, v_2)$, generated by task τ_k from Fig. 1. This sequence σ_k shows that in a time interval $t = 41$, task τ may generate a demand of 11 on the processor as follows. The first job is released at $t_1 = 5$ and the third job has its deadline at $d_3 = 46$. Thus, all three jobs of the sequence have both their release time and deadline within time interval [5, 46] of interval size 41. Together, their execution time is $2+5+4 = 11$. In fact, there are no job sequences generated by τ_k with a higher demand within an interval of length 41. We can conclude that the DBF $\text{DBF}_k(41) = 11$ for our example task τ_k. The DBF calculated for τ_k in Fig. 1 is given in Fig. 2.

The following theorem (proposed in [10]) shows how the DBF can be used in the feasibility analysis (or equally, the EDF-schedulability test) of the system $\mathcal{T}$.

Theorem 1 [10]: Task system $\mathcal{T}$ is EDF-schedulability upon pre-emptive uniprocessor if and only if $\forall t \geq 0$, $\text{DBF}(t) \leq t$.

Algorithm 1: DBF Methodology

```
1 for each interval size $\hat{t} = 1, 2, 3, \ldots, L$ do
2     if $\text{DBF}(\hat{t}) > \hat{t}$ then
3         return "infeasible" // according to Thm. 1
4 return "feasible"
```

According to Theorem 1, the EDF-schedulability of $\mathcal{T}$ can be checked by determining whether there is a t_f violating $\text{DBF}(t_f) \le t_f$. Such a t_f can be bounded by a constant L whose value depends upon the exact characteristics of DRT tasks in the system. The feasibility analysis framework for the task system $\mathcal{T}$ based upon the DBF is described in Algorithm 1. For each interval size $\hat{t} \in [1, L]$, we compute the DBF $\text{DBF}(\hat{t})$ by (1), and check the schedulability condition in line 2.

1) If overflow occurs, i.e., $\hat{t} < \text{DBF}(\hat{t})$, then it returns "infeasible" according to Theorem 1.
2) Otherwise, the computation and checking procedure continues for the next iteration, and finally, it returns "feasible" if $\text{DBF}(\hat{t}) \le \hat{t}$ holds for all interval sizes $\hat{t} \in [1, L]$.

The analysis framework of Algorithm 1 applies the DBF computation as a point-to-point checking procedure: as shown in lines 1–3, it needs to compute the DBF $\text{DBF}(\hat{t})$ and to check the schedulability condition for all interval sizes $\hat{t}$ ranging from 1 to L. As shown in [10], the interval-size bound L may have a pseudo-polynomial scale, which is a very large number especially when the total utilization summed over all tasks gets close to 1 (see in Section III-B1). The point-to-point checking procedure usually is time-consuming, and there is a lot of space for optimization and improvement. For the simplest sporadic task model, Zhang and Burns [11] designed a very efficient framework to accelerate the traditional DBF analysis, and we will review Zhang's framework in Section III-B2, and discuss how to adapt their technique to accelerate the analysis of DRT tasks.

Algorithm 1 only gives the basic analysis framework, but leaves two main computation problems as follows.

1) How do we compute the interval-size bound L?
2) How do we compute $\text{DBF}(\hat{t})$ for a specified interval size $\hat{t}$?

Stigge *et al.* [10] proposed a dynamical programming-based approach to solve the above problems, and we will briefly introduce their work in Section III-B1, exploring why Zhang's acceleration technique (for the simplest sporadic task model) is difficult to be integrated into Stigge's method for DRT tasks.

B. Relevant Prior Research

We now review some prior work that studies how the processor demand criteria methodology in Algorithm 1 is applied to analyze the DRT task model, and which studies the possible techniques to accelerate the methodology in Algorithm 1. Some notations proposed in the prior work are also used in the methods we derive in this article.

1) Stigge's Method for DRT Tasks: Stigge *et al.* [10] applied the DBF methodology in Algorithm 1 to analyze the DRT task model. Their main contribution is that they propose a DP-based approach to solve the DBF computation problem and derive a pseudo-polynomial-scale bound L. We will introduce their work in these two aspects.

DBF Computation: For a task τ_k and a specified interval size t, computing the DBF $\text{DBF}_k(t)$ is equivalent to finding an optimal path π^* such that π^* has the maximum workload and the length $l(\pi^*)$ of π^* is no more than t, according to (1). Stigge *et al.* [10] developed a DP algorithm to compute $\text{DBF}_k(t)$ by solving the optimal path problem on task graph G_k. The main idea of Stigge's method is to use the notion of demand triples as an abstraction of concrete paths through the task graph, preventing exponential path explosion. More specifically, for any vertex v_i, the *demand triple* of vertex v_i is denoted as (e, d, v_i), representing a finite path π such that π ends at v_i and has the workload equal e, i.e., $e(\pi) = e$, and the length of π is no more than d, i.e., $l(\pi) \le d$. Using all demand triples (e, d, v_i), we can calculate $\text{DBF}_k(t)$ as follows:

$$\text{DBF}_k(t) = \max\{e | (e, d, v_i) \text{ demand triple with } d \le t\}. \quad (3)$$

Now the remaining problem is how to compute all demand triples. In [10], the demand triple computation is applied as an iterative procedure as follows.

1) *Base Case:* Initially, the demand triples corresponding to all zero-length paths are computed and stored.
2) *Iterative Case.* For any stored demand triple (e, d, v_i), consider all successors v_j of v_i. For each such v_j, one can use the demand triple (e, d, v_i) to compute a new demand triple (e', d', v_j) corresponding to a path that has been extended by v_j with a heavier workload e' and a longer length d'. Each newly computed demand triple (e', d', v_j) is stored if it is not stored yet and $d' \le t$.

The above step is repeated until there are no new demand triples. Note that under the iterative procedure, the demand triples with shorter length are used to compute the demand triples with longer length. Moreover, during the runtime of the DBF analysis of Algorithm 1, for any interval sizes t and t' (with $t < t'$), the demand triples that are used to derive $\text{DBF}_k(t')$ must rely on the demand triples that are used to derive $\text{DBF}_k(t)$. *This restricts the DBF analysis to be a forward checking procedure.*

Bound Computation: We now introduce the pseudo-polynomial interval-size bound L proposed by Stigge *et al.* [10]. Before going into details, we first introduce some useful notations.

Definition 3 (Density): For any cycle $\pi = (v_1, \ldots, v_h, v_1)$, its *density* $u(\pi)$ is calculated as

$$u(\pi) = \frac{\sum_{i=1}^{h} C_i}{\sum_{i=1}^{h} p_{i,i+1}}. \quad (4)$$

For example, in Fig. 1, the density of cycle (v_1, v_2, v_3, v_1) is (10/47), and the density of cycle (v_2, v_3, v_4, v_2) is (7/39).

Definition 4 (Utilization): For any DRT task τ_k, its utilization u_k is calculated as follows:

$$u_k = \max\{u(\pi) | \pi \text{ is a cycle in } G_k\} \quad (5)$$

where $u(\pi)$ is the density of the cycle π. Moreover, we say the cycle in G_k that has the density u_k is the densest cycle.

Algorithm 2: QPA

```
1  $t := \max\{d_k | d_k < L\}$
2  while $h(t) \leq t \wedge h(t) > d_{\min}$ do
3      if $h(t) < t$ then
4          $t := h(t)$
5      else
6          $t := \max\{d_k | d_k < t\}$
7  if $h(t) \leq d_{\min}$ then
8      the task set is schedulable
9  else
10     the task set is not schedulable
```

For example, in Fig. 1, the densest cycle is (v_1, v_2, v_3, v_1), and thus, the utilization is (10/47).

The following lemma (proposed in [10]) gives the upper bound of the interval size t_f that violates $\text{DBF}(t_f) \leq t_f$.

Theorem 2 [10]: If task system $\mathcal{T}$ is not EDF-schedulability upon pre-emptive uniprocessor, there is a t_f with $\text{DBF}(t_f) > t_f$ such that

$$t_f < L = \frac{\sum_{k=1}^{n} \omega_k}{1 - \sum_{k=1}^{n} u_k} \tag{6}$$

where ω_k is the total workload of τ_k, i.e., $\omega_k = \sum_{v_i \in G_k} C_i$, and u_k is the utilization of τ_k.

From Theorem 2, the interval-size bound L used in Algorithm 1 is derived by (6), and the key problem for solving L is to calculate the task utilization u_k for each task τ_k. Stigge *et al.* [10] pointed out that the task utilization u_k can be solved by enumerating all *simple* cycles of the task graph G_k. Since explicit cycle enumeration is still an exponential procedure, Stigge *et al.* [10] reused their path abstraction framework which was already used to reduce the complexity of the DBF computation, and they state that *their method clearly has a pseudo-polynomial time complexity*, e.g., $O(P^2|V_k|)$, where $|V_k|$ is the number of vertices in G_k, and $P = \sum_{(v_i,v_j) \in G_k} p_{ij}$ is the summation of the periods of all edges in G_k (please refer to [10, p. 346]).

2) Quick Processor-Demand Algorithm: As shown in Algorithm 1, the traditional DBF methodology is usually implemented as a point-to-point checking procedure, such that all interval sizes in $[1, L]$ should be checked one by one. Note that the interval-size bound L usually is a very large number. Zhang and Burns [11] designed an efficient framework, called QPA, for accelerating the DBF analysis. Zhang's work is only available for the simplest sporadic task system consisting of n tasks (where each sporadic task τ_k has a WCET C_k, a relative deadline D_k, and a period T_k. The utilization u_k of τ_k equals (C_k/T_k), and we let $U = \sum_{k=1}^{n} u_k$), and their result is summarized by the following theorem.

Theorem 3 [11]: A task set is EDF-schedulable if and only if $U \leq 1$, and the result of the following iterative algorithm is $h(t) \leq d_{\min}$, where $d_{\min} = \min\{D_i\}$, and $h(t)$ is the processor demand function (or equally, the DBF) of the task set.

Here, d_k is the absolute deadline of a job of τ_k. The processor bound function $h(t)$ of the task set is calculated as follows:

$$h(t) = \sum_{k=1}^{n} \max\left\{0, 1 + \left\lfloor \frac{t - D_k}{T_k} \right\rfloor\right\} C_k \tag{7}$$

and the interval-size bound L is calculated as

$$L = \max\left\{D_1, \ldots, D_n, \frac{\sum_{k=1}^{n} (T_k - D_k) u_k}{1 - U}\right\}. \tag{8}$$

Different from the traditional analysis framework of Algorithm 1, QPA applies the DBF analysis by using a backward checking procedure as shown in lines 1–6 of Algorithm 2. Most importantly, Algorithm 2 does not check all interval sizes less than L, but only checks few interval sizes when necessary (lines 4 and 6). This is the key to sharply reduce the computational complexity. Moreover, the parameters [e.g., processor demand function $h(t)$ and the interval-size bound L] can be computed by (7) and (8), which have a very low complexity. QPA reduces the calculation effort exponentially in practice [11].

Difficulties to Adapting QPA to DRT Tasks: In this article, we aim to integrate the QPA framework into the DBF analysis for the DRT task model. However, the traditional method for DRT tasks (proposed in [10]) does not fit the QPA framework well. The reason is that QPA uses a backward checking procedure and aims to prune a lot of interval sizes for which overflow will not occur. However, the traditional method in [10] applies a forward checking procedure, i.e., they use a DP-based algorithm to compute the DBFs, which relies on sequentially checking all interval sizes less than L and does not allow interval-size pruning. This is clearly contradictory with the idea of QPA. In order to use QPA for accelerating the analysis of DRT tasks, we will propose some new techniques to solve the DBF (instead of the DP technique), which could fit the QPA framework better (see in Section IV).

IV. Acceleration Techniques

In this section, we propose efficient techniques to accelerate the DBF analysis of the DRT task model. The basic analysis framework is borrowed from the QPA method, which applies a backward checking procedure, and only checks the interval sizes for which the overflow may occur, see in Algorithm 3.

In Algorithm 3, we first compute the interval-size bound L according to Theorem 2 and initialize t to be L (lines 1–3). We check interval sizes t in a backward checking way: for any interval size t of interest, we check whether the DBF $\text{DBF}(t)$ is no more than t. If this is the case, then we set $t := \text{DBF}(t) - 1$ (line 9). Otherwise, an overflow occurs, and we return "unfeasible" (line 11). If no overflow occurs during checking progress, we return "feasible" (line 12).

The main benefit of using this method is the following. We do not need to check interval sizes in a point-to-point way. Instead, once we complete the check of interval size t, and find that $\Delta = t - \text{DBF}(t) \geq 0$, we then can directly "jump" to the smaller interval size $t' = t - \Delta - 1$ which may be far from t, and start the check procedure for the new interval size t'.

Algorithm 3: Acceleration Framework for DRT Tasks

```
1  for each $k = 1, \ldots, n$ do
2      compute the utilization $u_k$ of task $\tau_k$
3  compute $L$ by (6), and let $t := L$
4  while $t > 0$ do
5      for each $k = 1, \cdots, n$ do
6          compute $\text{DBF}_k(t)$
7      compute $\text{DBF}(t)$ by (2)
8      if $t \geq \text{DBF}(t)$ then
9          $t := \text{DBF}(t) - 1$
10     else
11         return "infeasible"
12 return "feasible"
```

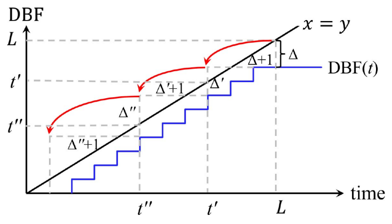

Fig. 3. Basic idea of our acceleration technique.

The correctness of this method is based on the observation described in the following lemma.

Lemma 1: For any interval size t, if $\text{DBF}(t) < t$, then $\text{DBF}(t') \leq t'$ $\forall t' \in [\text{DBF}(t), t]$.

Proof: Since the DBF is nondecreasing when interval size increases, for any interval size $t' \in [\text{DBF}(t), t]$, $\text{DBF}(t') \leq \text{DBF}(t)$. Moreover, since $t' \geq \text{DBF}(t)$, we have $\text{DBF}(t') \leq t'$. ■

According to Lemma 1, for any t such that $\text{DBF}(t) < t$, there is no interval size within $[\text{DBF}(t), t]$ violating the condition of Lemma 1. As illustrated in Fig. 3, the DBF line is always below the threshold line $\text{DBF}(t) = t$ during $[\text{DBF}(t), t]$. Thus, it does not need to check the interval sizes in $[\text{DBF}(t), t]$. In Section V, experimental results show that during the schedulability test, only a few interval sizes should be checked.

So far, we only introduce the framework of our method, but leave two main problems (e.g., DBF computation and utilization computation) that should be solved. Certainly, one choice is to solve the above problems by directly applying Stigge's method in [10]. As stated in Section III-B, Stigge's DP approach does not fit the acceleration framework (Algorithm 3). In the following, we revisit the two computation problems and develop more efficient techniques to solve these problems.

1) *Utilization Computation:* Stigge *et al.* [10] proposed a pseudo-polynomial time algorithm to solve the DRT task's utilization. We find that this utilization computation problem actually has a polynomial-time complexity. We propose an LP for computing the utilization as shown in Section IV-A.
2) *DBF Computation:* We propose an ILP to formulate the DBF computation (see in Section IV-B). By using the ILP technique, we can directly solve $\text{DBF}_k(t)$, which does not restrict to a point-to-point forward checking procedure and, thus, it fits the acceleration framework of Algorithm 3 better. The empirical result shows that our ILP-based method is much faster than Stigge's DP algorithm.

A. Utilization Computation

According to (5), we know that computing τ_k's utilization u_k means to find the *densest cycle* of G_k, where the density $u(\pi)$ of a cycle π is defined in (4). We use LP techniques to solve the densest cycle problem. We first give some useful notations and variables as follows.

For any vertex $v_i \in G_k$, we use $\text{PRED}(v_i)$ to denote the set of v_i's predecessors, i.e., $\text{PRED}(v_i) = \{v_j | (v_j, v_i) \in G_k\}$, and we use $\text{SUCC}(v_i)$ to denote the set of v_i's successors, i.e., $\text{SUCC}(v_i) = \{v_j | (v_i, v_j) \in G_k\}$. For any edge (v_i, v_j) of G_k, we denote a variable $x_{ij} (\leq 1)$ such that $x_{ij} > 0$ if the densest cycle travels (v_i, v_j). Otherwise, $x_{ij} = 0$. The LP model is given as follows:

$$\text{MODEL I:}\quad \max \quad u_k = \sum_{(v_i,v_j)\in G_k} C_i x_{ij} \tag{9}$$

$$\text{s.t.} \quad \sum_{v_j \in \text{PRED}(v_i)} x_{ji} - \sum_{v_j \in \text{SUCC}(v_i)} x_{ij} = 0,\ v_i \in G_k \tag{10}$$

$$\sum_{(v_i,v_j)\in G_k} p_{ij} x_{ij} = 1. \tag{11}$$

Constraint (10) ensures that the flows coming into and going out of the vertex v_i are balanced, which formulates the cycle π of G_k. Constraint (11) normalizes the cycle π's length (to unit time). After this normalization, the (normalized) cycle π's workload $\sum_{(v_i,v_j)\in\pi} C_i x_{ij}$ equals the density $u(\pi)$ of cycle π. Objective function (9) maximizes cycle π's density, and thus obtains the utilization u_k of G_k.

Correctness: In the following, we discuss whether our LP model is capable of solving the densest cycle of G_k. We start with the condition that a densest cycle needs to satisfy (see Lemma 2).

Lemma 2: The densest cycle is a simple cycle, or a set of simple cycles with the same density.

Proof: Suppose that the densest cycle π is not a simple cycle and, thus, π is constructed by several simple cycles. Without loss of generality, we let π consist of two simple cycles π_1 and π_2. The lemma follows if π_1 and π_2 have the same density. Hence, in the following, we consider the case that π_1 and π_2 have different densities, and we assume $u(\pi_1) > u(\pi_2)$.

By (4), the densities of cycles π_1 and π_2 are calculated as $u(\pi_1) = [e(\pi_1)/l(\pi_1)]$ and $u(\pi_2) = [e(\pi_2)/l(\pi_2)]$, where $e(\pi_i)$ and $l(\pi_i)$ are the workload and the length of the cycle π_i, respectively (for $i = 1, 2$). From $u(\pi_1) > u(\pi_2)$, we have

$$\frac{e(\pi_1)}{l(\pi_1)} > \frac{e(\pi_2)}{l(\pi_2)}$$
$$\Leftrightarrow e(\pi_1)l(\pi_2) + e(\pi_1)l(\pi_1) > e(\pi_2)l(\pi_1) + e(\pi_1)l(\pi_1)$$
$$\Leftrightarrow \frac{e(\pi_1)}{l(\pi_1)} > \frac{e(\pi_1) + e(\pi_2)}{l(\pi_1) + l(\pi_2)}$$

and since $e(\pi) = e(\pi_1) + e(\pi_2)$ and $l(\pi) = l(\pi_1) + l(\pi_2)$, we have $[e(\pi_1)/l(\pi_1)] > [e(\pi)/l(\pi)]$. By (4), we eventually derive $u(\pi_1) > u(\pi)$, i.e., the simple cycle π_1 is denser than π, which contradicts to the assumption that π is the densest cycle. ■

We use the following lemma to show that the solution space of our LP model is restricted to simple cycles of G_k or a set of simple cycles with the same density of G_k.

Lemma 3: MODEL I's solution is a simple cycle or a set of simple cycles with the same density.

Proof: Constraint (10) ensures that the solution of MODEL I is a set of simple cycles. Without loss of generality, we assume that the solution $X = [x_{ij}|(v_i, v_j) \in G_k]$ of MODEL I consists of two simple cycles π_1 and π_2. If π_1 and π_2 have the same density, this lemma certainly holds. Hence, in the following, we focus on the case that π_1 and π_2 have different densities, and we assume $u(\pi_1) > u(\pi_2)$. We rewrite constraint (11) as $\sum_{(v_i,v_j)\in\pi_1} p_{ij}x_{ij} + \sum_{(v_i,v_j)\in\pi_2} p_{ij}x_{ij} = 1$, where $x_{ij} \in X$ for each edge $(v_i, v_j) \in G_k$. We modify the solution X as follows.

1) For each edge $(v_i, v_j) \in \pi_2$, we let $x'_{ij} = 0$.
2) For each $(v_i, v_j) \in \pi_1$, we let $x'_{ij} = x_{ij} + \Delta$, where Δ is a constant defined as follows:

$$\Delta = \frac{1 - \sum_{(v_i,v_j)\in\pi_1} p_{ij}x_{ij}}{\sum_{(v_i,v_j)\in\pi_1} p_{ij}}. \tag{12}$$

3) For each edge $(v_i, v_j) \in G_k - (\pi_1 \cup \pi_2)$, we let $x'_{ij} = 0$.

We now obtain the new solution $X' = [x'_{ij}|(v_i, v_j) \in G_k]$ that only contains simple cycle π_1. In the following, we prove that the solution X' satisfies the constraints of MODEL I.

1) *Satisfaction of Constraint (10):* On the one hand, since X satisfies constraint (10), we know that for $v_i \in \pi_1$

$$\sum_{v_j\in\text{PRED}(v_i)} x_{ji} = \sum_{v_j\in\text{SUCC}(v_i)} x_{ij}$$
$$\Leftrightarrow \sum_{v_j\in\text{PRED}(v_i)} (x_{ji} + \Delta) = \sum_{v_j\in\text{SUCC}(v_i)} (x_{ij} + \Delta)$$
$$\Leftrightarrow \sum_{v_j\in\text{PRED}(v_i)} x'_{ji} = \sum_{v_j\in\text{SUCC}(v_i)} x'_{ij}. \tag{13}$$

On the other hand, for any vertex v_i that is not in cycle π_1, the edges (v_j, v_i) coming into v_i correspond to zero variables $x_{ji} = 0$, and the edges (v_i, v_j) going out of v_i also correspond to zero variables $x_{ij} = 0$, i.e., for $v_i \in G_k - \pi_1$

$$\sum_{v_j\in\text{PRED}(v_i)} x'_{ji} = \sum_{v_j\in\text{SUCC}(v_i)} x'_{ij} = 0. \tag{14}$$

By (13) and (14), the solution X' satisfies constraint (10).

2) *Satisfaction of Constraint (11):* Since only the edges (v_i, v_j) visited along π_1 have positive variable, we have

$$\sum_{(v_i,v_j)\in G_k} p_{ij}x'_{ij} = \sum_{(v_i,v_j)\in\pi_1} p_{ij}x'_{ij}$$

and since $x'_{ij} = x_{ij} + \Delta$, for each $(v_i, v_j) \in \pi_1$, we have

$$\sum_{(v_i,v_j)\in G_k} p_{ij}x'_{ij} = \sum_{(v_i,v_j)\in\pi_1} p_{ij}(x_{ij} + \Delta)$$

and by (12), we know that

$$\sum_{(v_i,v_j)\in G_k} p_{ij}x'_{ij} = \sum_{(v_i,v_j)\in\pi_1} p_{ij}x_{ij} + 1 - \sum_{(v_i,v_j)\in\pi_1} p_{ij}x_{ij} = 1.$$

This indicates that the solution X' satisfies constraint (11).

In sum, the newly obtained solution X' satisfies the constraints of MODEL I and consists of a simple cycle. ■

According to Lemma 3, any solution of MODEL I corresponds to a simple cycle of G_k or a set of simple cycles with the same density. Moreover, by the objective function (9), we know that MODEL I must solve the density of the densest cycles of G_k. Moreover, according to Lemma 2, we conclude the correctness of MODEL I in Corollary 1.

Corollary 1: MODEL I solves the density of the densest cycle.

It should be noted that MODEL I aims to solve the density of the densest cycle. It does not matter whether the densest cycle is unique or not. If there is more than one densest cycle, the solution of MODEL I may correspond to a set of (several) densest cycles, and the objective value of MODEL I equals to the density of (any one of) the densest cycles.

Complexity: The following proposition indicates that our LP model is solved within polynomial time.

Proposition 1: There are m variables and $n + 1$ constraints in MODEL I, where n is the number of vertices in G_k, and m is the number of edges in G_k.

Proof: Since each edge (v_i, v_j) of G_k corresponds to a variable x_{ij}, MODEL I contains m variables. For each vertex $v_i \in G_k$, there is a constraint of (10). Moreover, there is a single constraint of (11). MODEL I contains $n+1$ constraints. ■

Recall that Stigge's method for computing utilization u_k has the pseudo-polynomial time complexity, e.g., $O(P^2|V_k|)$, where $|V_k|$ is the number of vertices in G_k, and $P = \sum_{(v_i,v_j)\in G_k} p_{ij}$. Clearly, our LP model has lower computational complexity than the Stigge's method.

B. DBF Computation

For a given t, computing $\text{DBF}_k(t)$ of τ_k is to find the optimal path π of G_k such that π has the maximum workload and π's length is bounded by t according to (1). In this section, we solve such an optimal path by using ILP techniques. Before going into details, we first introduce an auxiliary digraph G'_k of G_k as follows.

To construct the auxiliary digraph G'_k, we add a *source* vertex v_{src} and a *sink* vertex v_{snk} into the original graph G_k. Both the newly added vertices have zero execution time (i.e., $C_{src} = 0$, $C_{snk} = 0$) and zero deadline (i.e., $D_{src} = 0$, $D_{snk} = 0$). For each vertex v_i of G_k, we add an edge (v_{src}, v_i) from the source vertex v_{src} to v_i, and add an edge (v_i, v_{snk}) from v_i to the sink vertex v_{snk}. The edge (v_{src}, v_i) has a zero period, i.e., $p_{src,i} = 0$. The period of the edge (v_i, v_{snk}) equals to the deadline of v_i, i.e., $p_{i,snk} = D_i$. The auxiliary digraph G'_k contains all vertices and edges of G_k as well as the newly added source vertex, sink vertex, and their associated edges. In the following, we give a very simple example to show how to construct such an auxiliary digraph.

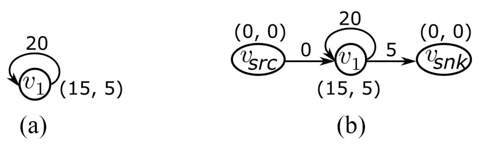

Fig. 4. Illustration for constructing the auxiliary digraph. (a) Original digraph. (b) Auxiliary digraph.

Example 2: We consider the original digraph G_k that contains a single vertex v_1 and a self-loop edge (v_1, v_1) as shown in Fig. 4(a). The WCET and deadline of v_1 are $C_1 = 15$ and $D_1 = 5$, respectively. The period of (v_1, v_1) is $p_{11} = 20$. The auxiliary digraph G'_k of G_k is shown in Fig. 4(b), where a source vertex v_{src} and a sink vertex with zero WCET and zero deadline are added into the original digraph G_k. Moreover, the edge (v_{src}, v_1) has a zero period and the edge (v_1, v_{snk}) has a period $p_{1,snk} = 5$.

In the auxiliary digraph, the source and sink vertices and their associated edges are essential for solving the optimal path of the original digraph G_k. We let Ω be the set of paths in the auxiliary digraph G'_k that start with the source vertex v_{src} and end at the sink vertex v_{snk}. The following lemma indicates a one-to-one correspondence between the paths in the original digraph G_k and the paths in Ω of the auxiliary digraph G'_k.

Lemma 4: For any path π of G_k, there is a path π' in Ω such that π and π' have the same workload and the same length.

Proof: Without loss of generality, we assume that the path π of G_k starts with v_i and ends at v_j, i.e., $\pi = (v_i, \ldots, v_j)$. We add two edges (v_{src}, v_i) and (v_j, v_{snk}) to π, and obtain the path $\pi' = (v_{src}, v_i, \ldots, v_j, v_{snk})$ of G'_k. Clearly, π' starts with the source vertex v_{src} and ends at the sink vertex v_{snk}, and thus, $\pi' \in \Omega$. As we know that π' has two vertices (e.g., v_{src} and v_{snk}) more than π, the workload of π' is calculated as $e(\pi') = C_{src} + C_{snk} + e(\pi)$. Since $C_{src} = 0$ and $C_{snk} = 0$, we have $e(\pi') = e(\pi)$. Since π' has two edges (v_{src}, v_i) and (v_j, v_{snk}) more than π, and the last edge of π' is (v_j, v_{snk}), the length of π' is calculated as $l(\pi') = p_{src,i} + \sum_{(v_x,v_y)\in\pi} p_{xy} + p_{j,snk} + D_{snk}$. Since $p_{j,snk} = D_j$ and $D_{snk} = 0$, we have $l(\pi') = p_{src,i} + l(\pi)$. Moreover, since $p_{src,i} = 0$, we have $l(\pi') = l(\pi)$. This completes the proof. ■

According to Lemma 4, we can use the path set Ω of the auxiliary digraph G'_k to derive $\text{DBF}_k(t)$ as follows.

Lemma 5: For any task τ_k, $\text{DBF}_k(t)$ is calculated as follows:

$$\text{DBF}_k(t) = \max\{e(\pi) | \pi \in \Omega \wedge l(\pi) \le t\}. \tag{15}$$

Proof: From Lemma 4, each path π of the original digraph G_k corresponds to a path π' of Ω such that π and π' have the same workload and the same length. By (1), we derive (15). ■

According to Lemma 5, we can calculate $\text{DBF}_k(t)$ by finding an optimal path π of Ω with the maximum workload and a bounded length $l(\pi) \le t$. Recall that Ω contains the paths of the auxiliary digraph G'_k that start with v_{src} and end at v_{snk}. We model such a path π of Ω into an ILP formulation. For the sake of convenience, we propose our ILP model in two steps. First, we introduce a basic model that only contains a part of the constraints, called MODEL II. This model is easy to understand, but has some flaws. Then, we introduce the remaining constraints of our ILP, which are more complicated, but address the shortcomings of MODEL II.

Basic Model: For any edge (v_i, v_j) of the auxiliary digraph G'_k, we denote a non-negative integer variable z_{ij} to represent the times we travel the edge (v_i, v_j). Based on these notations, we give an ILP model as MODEL II as follows:

$$\text{MODEL II:} \quad \max \quad \text{DBF}_k(t) = \sum_{(v_i,v_j)\in G'_k} C_i z_{ij} \tag{16}$$

$$\text{s.t.} \quad \sum_{v_i \in \text{SUCC}(v_{src})} z_{src,i} = 1 \tag{17}$$

$$\sum_{v_i \in \text{PRED}(v_{snk})} z_{i,snk} = 1 \tag{18}$$

$$\sum_{v_j \in \text{PRED}(v_i)} z_{ji} - \sum_{v_j \in \text{SUCC}(v_i)} z_{ij} = 0, v_i \in G_k \tag{19}$$

$$\sum_{(v_i,v_j)\in G'_k} p_{ij} z_{ij} \le t. \tag{20}$$

Objective function (16) maximizes the workload of path π of G'_k. Constraints (17) and (18), respectively, enforce that the path π starts with the source vertex v_{src} and ends at the sink vertex v_{snk}, i.e., $\pi \in \Omega$. Constraint (19) ensures that for each vertex v_i of G_k, the flows coming into v_i and the flows going out of v_i are the same, which is necessary to formulate a path. Here, $\text{PRED}(v_i)$ contains all predecessors of v_i (including v_{src}), and $\text{SUCC}(v_i)$ contains all successors of v_i (including v_{snk}). Constraint (20) ensures that π's length is bounded by t.

To illustrate how MODEL II solves $\text{DBF}_k(t)$, we take the DRT task in Fig. 4 as an example. MODEL II is applied on the auxiliary digraph G'_k as shown in Fig. 4(b). For a given interval length $t = 25$, the solution of MODEL II is $z_{src,1} = 1, z_{11} = 1$, and $z_{1,snk} = 1$. This corresponds to the path $\pi' = (v_{src}, v_1, v_1, v_{snk})$. We know that π' has the workload $e(\pi') = 30$ and the length $e(\pi') = 25$, and accordingly, the objective value solved by MODEL II equals 30, i.e., the DBF calculated by MODEL II is $\text{DBF}_k(25) = 30$. As shown in Fig. 4(a), the optimal path π of the original digraph G_k with the length 25 is $\pi = (v_1, v_1)$. The workload of π is $e(\pi) = 30$, and thus, $\text{DBF}_k(25) = 30$. Therefore, MODEL II computes the DBF correctly (at least for the instance in Fig. 4).

Lemma 6: For a given interval length t, the solution of MODEL II upper-bounds $\text{DBF}_k(t)$.

Proof: For any path $\pi \in \Omega$ with length bounded by t, we can construct a vector $Z = (z_{ij} | (v_i, v_j) \in G'_k)$ as follows. For any edge $(v_i, v_j) \in G'_k$, if the path π contains (v_i, v_j), we let z_{ij} equal the times for which the edge (v_i, v_j) is traveled in π. Otherwise, (v_i, v_j) is not contained in π, and we let $z_{ij} = 0$. We now show that Z satisfies all constraints of MODEL II.

1) Since there are only out-going edges of the source vertex v_{src}, and the path π starts with v_{src}, we know that there is exactly one vertex v_i such that $z_{src,i} = 1$ and, thus, Z satisfies constraint (17).
2) Since the sink vertex v_{snk} only has in-coming edges, and π ends at v_{snk}, there is exactly one vertex v_j such that $z_{j,snk} = 1$, and thus, Z satisfies constraint (18).
3) For any vertex v_i of G_k, if v_i is traveled in π, then the times we enter into v_i must equal to the times we leave

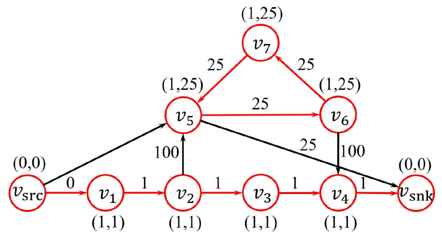

Fig. 5. Counter-example for MODEL II.

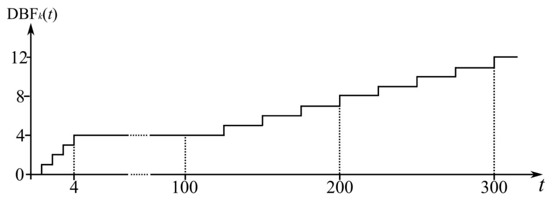

Fig. 6. DBF of the DRT task τ_k in Fig. 5.

v_i since v_i is a medium vertex of π and, therefore, Z satisfies constraint (19).

4) Since $D_{snk} = 0$, the length of π equals $\sum_{(v_i,v_j)\in\pi} p_{ij} = \sum_{(v_i,v_j)\in G'_k} p_{ij}z_{ij}$. Moreover, since the length of π is bounded by t, we have $\sum_{(v_i,v_j)\in G'_k} p_{ij}z_{ij} \le t$, i.e., constraint (20) holds.

In sum, the solution space of MODEL II contains all paths of Ω with length bounded by t. Moreover, for any path π, since the workload of π equals $\sum_{(v_i,v_j)\in\pi} C_i = \sum_{(v_i,v_j)\in G'_k} C_i z_{ij}$, the objective function (16) aims to obtain the maximum workload among the solutions of MODEL II (including the paths $\pi \in \Omega$ with length bounded by t). According to Lemma 5, $\text{DBF}_k(t)$ equals the maximum workload among all paths of Ω with length bounded by t. Therefore, the objective value solved by MODEL II is no less than $\text{DBF}_k(t)$. ■

Flaws of MODEL II*:* MODEL II may not solve $\text{DBF}_k(t)$ exactly, since MODEL II may have some feasible solutions that are not paths in G_k. We give a counter-example as follows.

Example 3: We consider the auxiliary digraph G'_k with additional vertices v_{src} and v_{snk} as shown in Fig. 5. For each vertex $v_i \in G_k$, there are two edges (v_{src}, v_i) and (v_i, v_{snk}). For a given $t = 100$, we use MODEL II to solve the maximum workload of the path in G'_k which is from v_{src} to v_{snk} and has the length bounded by 100. We obtain the solution Z described as follows: $z_{src,1} = 1$, $z_{12} = 1$, $z_{23} = 1$, $z_{34} = 1$, $z_{4,snk} = 1$, $z_{56} = 1$, $z_{67} = 1$, and $z_{75} = 1$.The solution Z corresponds to a path π_1 from v_{src} to v_{snk} and a cycle π_2 that contains vertices v_5, v_6, and v_7, as marked red in Fig. 5. The total length of π_1 and π_2 is 79($\le$ 100), and the total workload solved by MODEL II is $4+3 = 7$. Therefore, the DBF solved by MODEL II is $\text{DBF}_k(100) = 7$. Actually, there are two paths in G'_k from v_{src} to v_{snk} and with length no more than 100. The first path is $\pi_3 = (v_{src}, v_1, v_2, v_3, v_4, v_{snk})$ with length 4 and workload 4. The second path is $\pi_4 = (v_{src}, v_5, v_6, v_7, v_5, v_{snk})$ with length 100 and workload 4. The maximum workload of both paths equals 4. According to Lemma 5, the actual $\text{DBF}_k(100)$ equals 4 (see in Fig. 6).

Example 3 shows that the DBF solved by MODEL II may be much larger than the actual value, e.g., the ratio of the gap between them is about 75%. Therefore, MODEL II cannot solve the DBF exactly. The main reason is that the solution of MODEL II may contain *isolated cycles* (we say a cycle π is *isolated* if the vertices of π are unreachable from the source vertex v_{src}). To preclude isolated cycles, we will add some additional variables and constraints into MODEL II as follows.

Additional Constraints: For each vertex $v_i \in G_k$, we define a Boolean variable y_i such that $y_i = 1$ if v_i is traveled along the path π. Otherwise, $y_i = 0$. The following constraints show the relation between the variables z_{ij} and y_i. For each vertex $v_i \in G_k$, using the big number $B = \infty$

$$\sum_{v_j\in\text{SUCC}(v_i)} z_{ij} \ge y_i \tag{21}$$

$$By_i \ge \sum_{v_j\in\text{SUCC}(v_i)} z_{ij}. \tag{22}$$

Constraint (21) ensures that if the vertex v_i is traveled (e.g., $y_i = 1$), then at least one edge goes out of v_i, i.e., $\sum_{v_j\in\text{SUCC}(v_i)} z_{ij} \ge 1$. Constraint (22) ensures that if some edges going out of v_i are traveled (e.g., $\sum_{v_j\in\text{SUCC}(v_i)} z_{ij} \ge 1$), then the vertex v_i must be traveled (e.g., $y_i = 1$).

In the following, we will propose the constraint to preclude isolated cycles. Before going into details, we first introduce some useful notations.

Definition 5 (Cut Set): For vertex $v_i \in G_k$, the cut set $\text{CUT}(v_i)$ is the set of edges in the auxiliary digraph G'_k such that:

1) the vertex v_i is not reachable from the source vertex v_{src} if the edges of $\text{CUT}(v_i)$ are all removed;
2) for any subset $E \subset \text{CUT}(v_i)$, the vertex v_i is reachable from the source vertex v_{src} if the edges of E are removed.

Moreover, we denote the minimum cut set $\text{CUT}_{\min}(v_i)$ as the cut set of v_i that has the minimum edge number, i.e.,

$$\text{CUT}_{\min}(v_i) = \min_{\text{CUT}(v_i)\in G'_k} |\text{CUT}(v_i)|.$$

For example, in Fig. 5, the cut set $\text{CUT}(v_6)$ of v_6 may be $\{(v_5, v_6)\}$ or $\{(v_{src}, v_5), (v_2, v_5)\}$. The minimum cut set $\text{CUT}_{\min}(v_6) = \{(v_5, v_6)\}$. From Definition 5, we know that if a path starting with v_{src} travels v_i, then the path π must travel the edge in $\text{CUT}(v_i)$. This is formulated by the following constraint. For each $v_i \in G_k$, using big number $B = \infty$

$$B(1 - y_i) + \sum_{(v_j,v_l)\in\text{CUT}(v_i)} z_{jl} \ge 1 \quad \forall\text{CUT}(v_i) \in G'_k. \tag{23}$$

The correctness of constraint (23) is illustrated as follows. If the vertex v_i is traveled (e.g., $y_i = 1$), then constraint (23) becomes $\sum_{(v_j,v_l)\in\text{CUT}(v_i)} z_{jl} \ge 1$, indicating that the edge of $\text{CUT}(v_i)$ must be traveled. If the vertex v_i is not traveled (e.g., $y_i = 0$), then constraint (23) becomes $B \ge 1$, indicating that it is not necessary to travel the edges of $\text{CUT}(v_i)$.

We combine the newly added variables y_i (for all vertices $v_i \in G_k$) and Constraints (21)–(23) into MODEL II, and obtain the following ILP model, denoted as MODEL III, i.e.,

$$\begin{aligned} &\text{MODEL III} \quad \text{Objective function (16)} \\ &\text{s.t.} \quad \text{Constraints (17)–(23).} \end{aligned}$$

Algorithm 4: Row Generation Algorithm

```
1 solve MODEL II, and obtain the solution $Z^*$
2 **while** *the solution $Z^*$ contains isolated cycles* **do**
3     **for** *each isolated cycle $\pi$ of $G_k$* **do**
4         CUT := $\arg\min_{v_i \in \pi} |\text{CUT}_{\min}(v_i)|$
5         add $B(1-y_i) + \sum_{(v_i,v_j)\in \text{CUT}} z_{ij} \geq 1$ into MODEL II
6     solve MODEL II (with newly added constraints), and
      obtain $Z^*$
7 **return** $\text{DBF}_k(t) = \sum_{(v_i,v_j)\in G'_k} e_i z^*_{ij}$
```

Clearly, MODEL III solves the optimal path with the length no more than t and with the maximum workload, and according to Lemma 5, MODEL III exactly derives $\text{DBF}_k(t)$. However, MODEL III may have an exponential number of constraints in (23), since the number of cut sets $\text{CUT}(v_i)$ for a vertex v_i may be exponential in m, the number of edges in G_k. It is very hard to enumerate all constraints of (23) and, thus, MODEL III cannot be directly solved. In the following, we propose a row generation algorithm to solve MODEL II efficiently.

Row Generation Algorithm: We solve MODEL III in an iterative way. The main idea is that we do not enumerate all constraints of MODEL III, and instead, we solve MODEL III with part of its constraints, and iteratively add the constraints only when necessary. As we know that in the operational research (OR) community, the variables in the ILP are usually called as *columns*, and the constraints are called as the *rows* of the ILP. Accordingly, our approach that iteratively adds new constraints is called the *row generation algorithm* as given in Algorithm 4.

In Algorithm 4, we solve MODEL II, the initial ILP model with (16)–(20) and obtain the solution Z^*. If Z^* indicates a feasible path, then the objective function of Z^* is returned as $\text{DBF}_k(t)$. Otherwise, there are some isolated cycles in the solution Z^*. For each isolated cycle π, we find the minimum cut set CUT among all vertices of π (lines 3 and 4). Based on the minimum cut set CUT, we add a constraint of (23) into MODEL II as shown in line 5. Then, we solve the updated model and obtain the solution Z^*. If Z^* contains the isolated cycles, then we start the computation process (lines 2–6) in the next iteration. This process repeats until the solution contains no isolated cycle.

It should be noted that Algorithm 4 iteratively adds the constraints in (23) into MODEL II, and it adds one constraint at a time. There are exponential number of constraints implied by (23), and thus, one may wonder whether Algorithm 4 will do exponentially many iterations. The following lemma answers this question by showing that only a polynomial number of constrains in (23) are added when Algorithm 4 terminates.

Lemma 7: Algorithm 4 adds at most $|V_k|$ constraints in (23) into MODEL II when it terminates, where $|V_k|$ is the number of vertices in G_k.

Proof: In each iteration, we add one constraint $\mathcal{C}$ in (23) for any isolated cycle π, according to lines 4 and 5 of Algorithm 4. Without loss of generality, we assume that the cycle π contains a vertex v_i. The constraint $\mathcal{C}$ added for precluding π is $B(1-y_i)+\sum_{(v_j,v_l)\in\text{CUT}(v_i)} z_{jl} \geq 1$, which is associated with the vertex v_i, and moreover, the cut set used in the constraint $\mathcal{C}$ is denoted as $\text{CUT}(v_i)$. After adding the constraint $\mathcal{C}$, we ensure that any cycle π that contains v_i should not be isolated. Otherwise, we suppose that a cycle π containing v_i is isolated, i.e., the vertices in π (including v_i) are traveled, but none of them is reachable from the source vertex v_{src} of G'_k. According to Definition 5, no edge in $\text{CUT}(v_i)$ is traveled, indicating that $\sum_{(v_j,v_l)\in\text{CUT}(v_i)} z_{jl} = 0$. Moreover, we know that $y_i = 1$ since v_i is traveled. Therefore, the constraint $\mathcal{C}$ (that have been added into MODEL II) is violated. This leads to a contradiction.

From above, once Algorithm 4 adds a constraint that is associated with a vertex v_i of G_k, then other constraints that are associated with the same vertex v_i will not be added at the later iterations. It indicates that the constraints added by Algorithm 4 are associated with different vertices of G_k. At each iteration, there is at least one isolated cycle, which must contain at least one vertex. Therefore, after at most $|V_k|$ iterations, the constraints added by Algorithm 4 are associated with all vertices of G_k, and Algorithm 4 must terminate. ■

Lemma 7 shows that instead of doing exponentially many iterations, Algorithm 4 only iterates at most $|V_k|$ times. In fact, Algorithm 4 seldomly does such many iterations. In our experimental work, we observe that isolated cycles can be estimated after only one or two iterations. This makes our method perform very efficiently in practice.

V. Evaluations

This section reports the comparison between the Stigge's method A_{STG} in [10] and our method A_{SUN}. We randomly generate the DRT task sets consisting of n DRT tasks and with the total utilization u. Each DRT task has 5–9 vertices. The execution time of each vertex ranges from 1 to 4, and the period of each edge ranges from 100 to 200. We implement our models by using Python API of Z3 solver. The code runs on a PC with Intel Core i5-6300U CPU at 2.4GHz with 8G RAM. Figs. 7–12 conduct experiments with different combinations of parameters. We exhibit our experiments by using the box plot. In a box plot, the top and the bottom of the box, respectively, represent the first and third quartiles of data. The middle line of the box represents the median of data. The whiskers extended from the box show the range of data. For each data point, 1000 random experiments have been run. The longitudinal axes are log 10 transformed to adjust for the wide range of data.

A. Evaluation of Schedulability Analysis

In this section, we evaluate the overall schedulability analysis methods. We record the number of interval sizes (written as #num in the figures) checked by A_{STG} and A_{SUN}, denoted as Γ_{STG} and Γ_{SUN}, respectively. We also measure the computation time of each schedulability analysis method. We use t_{SUN} and t_{STG} to denote the computation time of our method and Stigge's method, respectively. Since Stigge's method needs to check all interval lengths ranging from 1 to L when a task set is schedulable (but can stop once an overflow is found for an

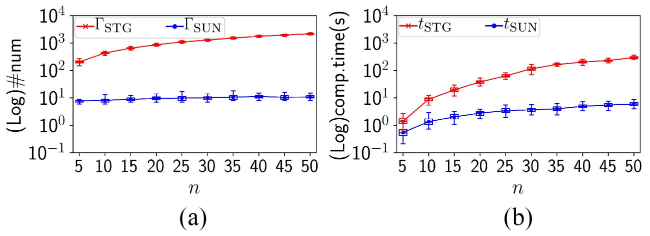

Fig. 7. Impact of the number of tasks in schedulable task sets. (a) Evaluate #num. (b) Evaluate comp.time.

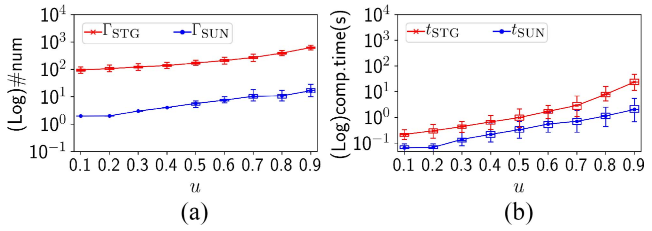

Fig. 8. Impact of the utilization of schedulable task sets. (a) Evaluate #num. (b) Evaluate comp.time.

unschedulable task system), we do the experiments separately for schedulable and unschedulable task sets.

1) Experiments on Schedulable Task Sets: We experiment on the task sets which are schedulable. Fig. 7 shows how the task number n impacts the schedulability method's performance. We let each task set's utilization be 0.6, and the task set with such an utilization has a 50% chance to be schedulable. If a generated task set is unschedulable, the program discards it and does not count it into the experimental results. As shown in Fig. 7(a), the checked interval-size number Γ_{SUN} is 10 on average, which does not significantly change when the task number n increases. Γ_{STG} is 1193 on average, which is 117 times larger than Γ_{SUN}. As shown in Fig. 7(b), t_{STG} (as well as t_{SUN}) increases when the task number n increases. When the task number is large (e.g., $n = 50$), t_{STG} is at least 118.68 s, and t_{SUN} is no more than 3.45 s, i.e., our method is 34 times faster than Stigge's method.

Fig. 8 shows the impact on the task set's utilization u. We set the task number of each instance as $n = 5$ to ensure that Stigge's method can analyze the generated task sets with both low and high utilization within a few seconds. As shown in Fig. 8(a), the checked interval-size numbers Γ_{SUN} and Γ_{STG} tend to increase when the utilization u increases. Γ_{SUN} is 7 on average, which is 32 times smaller than Γ_{STG}. As shown in Fig. 8(b), the computation time t_{STG} as well as t_{SUN} both grow in an exponential pattern when the utilization u increases. Our method is 5.7 times faster than the Stigge's method on average.

2) Experiments on Unschedulable Task Sets: We do experiments on the unschedulable task sets. Figs. 9 and 10 show the impact on task number n and the utilization u, respectively. In Fig. 9, we set the utilization $u = 0.6$. In Fig. 10, we let the task number $n = 5$, and let the utilization range from 0.5 to 0.9, since it is hard to generate an unschedulable task set when the utilization is less than 0.5. We observe that the checked interval-size number Γ_{SUN} is not always smaller than Γ_{STG}. This is because that for an unschedulable task set, the first deadline missing point is often closer to 1. Stigge's

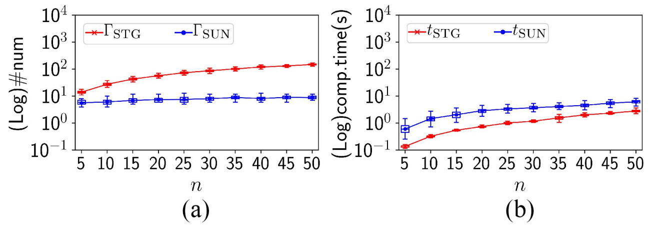

Fig. 9. Impact of the number of tasks in unschedulable task sets. (a) Evaluate #num. (b) Evaluate comp.time.

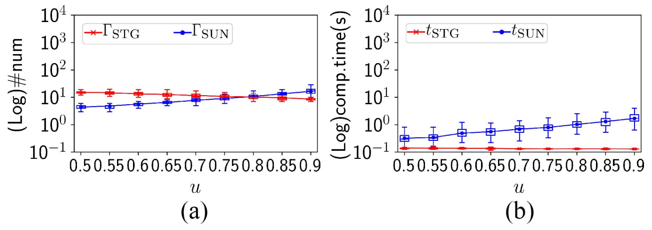

Fig. 10. Impact of the utilization of unschedulable task sets. (a) Evaluate #num. (b) Evaluate comp.time.

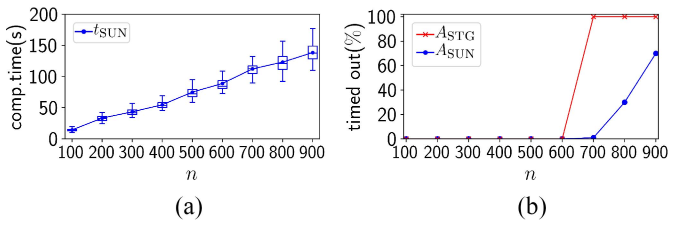

Fig. 11. Evaluate our method for solving large-scale task sets. (a) Comp.time of A_{SUN} ($u = 6$). (b) Time-out instance ratio ($u = 9$).

method uses forward searching strategy and, thus, it discovers the very first deadline missing point and terminates after checking few interval sizes (e.g., no more than 46 on average). On the other hand, our method starts from L and works backward toward 1, which needs to check 8 or more interval sizes before concluding that the task set is unschedulable. Although our method cannot dominate Stigge's method when analyzing unschedulable task sets, our method is very fast, i.e., the computation time of our method is no more than 2.62 s as shown in Figs. 9(b) and 10(b).

3) Scalability of Our Method: Our method can scale to larger task sets. As shown in Fig. 11(a), our method is capable of analyzing large-scale task sets (consisting of 900 DRT tasks) within 76.10 s. As we know that there is an exponential pattern in the growth of our method's computation time when the utilization increases, our method may be slow when dealing with large-scale task sets with high utilization (e.g., $u = 0.9$). If the method (e.g., A_{STG} and A_{SUN}) cannot complete the schedulability analysis within 500 s, it is considered to have "timed-out." We evaluate the timed-out instance ratio of both methods with different numbers of tasks as shown in Fig. 11(b). We observe that when the task set has a high utilization 0.9, and the task number exceeds 800, more than 33% instances are timed-out under our method. We also observe that if our method is timed-out, Stigge's method is slow as well, i.e., timed-out instance ratio of A_{STG} sharply increases to 100% when the task number exceeds 700 (and at the same data point, our method's time-out instance ratio is only 1%).

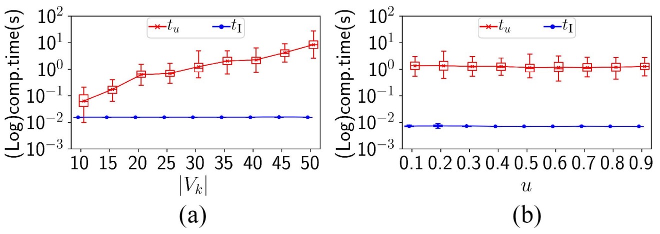

Fig. 12. Evaluate computation time for solving utilization. (a) $u = 0.5$. (b) $|V_k| = 30$.

B. Evaluation for Utilization Computing

We also evaluate the computation time for solving the utilization. We use t_I to denote the computation time for solving MODEL I and use t_u to denote the computation time of Stigge's utilization computation method. As shown in Fig. 12, MODEL I can be solved within 0.01 s, and Stigge's method exceeds 1.94 s on average. The computation time t_I increases when the vertex number $|V_k|$ and the utilization u_k of G_k increase.

VI. Related Work

Graph-based task models include recurring branching [3], recurring [4], noncyclic recurring [5], GMF [6], and so on. One of the most expressive graph-based task models is the DRT task model [10] using arbitrary directed graphs for modeling task activations. This section mainly summarizes the related work for DRT tasks. Stigge *et al.* [10] first proposed the DRT task model and developed a pseudo-polynomial time approach to analyze the schedulability of DRT tasks under the dynamic priority scheduling (e.g., EDF). In [8], they further clarify the tractability of schedulability analysis of DRT tasks (under EDF). For static priority (SP) schedulers, Stigge and Yi [12] showed the intractability of the schedulability analysis of DRT tasks, and in [13], they proposed an iterative approach to efficiently cope with the combinatorial explosion in the analysis process. Using the similar technique, Stigge *et al.* [14] solved the response time analysis (RTA) problem for DRT task under SP scheduling. Guan *et al.* [15] proposed an approximation algorithm to solve the RTA problem for DRT tasks under SP scheduling, and in [16], they used the real-time calculus for the delay analysis of each vertex in DRT tasks. Gu *et al.* [17] used graph transformation methods to improve the schedulability of DRT tasks under SP scheduling. For both dynamic and SP schedulers, Zeng and Natale [18] used the max-plus algebra technique to improve the efficiency of the schedulability analysis for the strongly connected DRT tasks. Please see the survey [19] for details.

DRT task models are extended into different application scenarios. Guan *et al.* [20] and Abdullah *et al.* [21] studied the scheduling strategies for the DRT tasks with resource sharing constraints. Ekberg and Yi [22] used DRT task model to describe the mixed criticality system. Mohaqeqi *et al.* [23], Fradet *et al.* [24], and Xu *et al.* [25] studied the dependent DRT tasks, i.e., there are dependence constraints among DRT tasks. Ben *et al.* [26] studied the DRT tasks with the probabilistic WCET. Sun [27], Zahaf *et al.* [28], and Houssam *et al.* [29] extended the DRT task model to support the inner parallel tasks. Mohaqeqi *et al.* [30] used the DRT task model to formulate the data flow graphs. Abdullah *et al.* [31] and Mohaqeqi *et al.* [32] used DRT tasks to formulate Ada code. Guo and Baruah studied the adaptive varying rate (AVR) tasks and used DRT as a possible representation model to formulate AVR tasks [33].

VII. Conclusion

The traditional feasible analysis for the DRT task model used a DP-based approach to deal with the exponential explosion, but is hard to be accelerated. In this article, we provide efficient techniques to accelerate the analysis of DRT tasks. By using LP techniques, we reduce utilization computation's complexity from pseudo-polynomial time to polynomial time. By using the ILP techniques, we propose an efficient method to compute the DBF, which facilitates the implementation of acceleration techniques. The experimental work shows that our method has a much shorter run time than the traditional method.

Acknowledgment

The students at Northeastern University, particularly Xisheng Li and Zhiyu Hu have done part of the coding work. The authors would like to thank them for their hard work.

References

[1] 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, 1973.

[2] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proc. RTSS*, 1990, pp. 182–190.

[3] S. K. Baruah, "Feasibility analysis of recurring branching tasks," in *Proc. 10th EUROMICRO Workshop Real Time Syst.*, 1998, pp. 138–145.

[4] S. K. Baruah, "Dynamic-and static-priority scheduling of recurring real-time tasks," *Real Time Syst.*, vol. 24, no. 1, pp. 93–128, 2003.

[5] S. K. Baruah, "The non-cyclic recurring real-time task model," in *Proc. RTSS*, 2010, pp. 173–182.

[6] S. K. Baruah, D. Chen, S. Gorinsky, and A. K. Mok, "Generalized multiframe tasks," *Real Time Syst.*, vol. 17, no. 1, pp. 5–22, 1999.

[7] A. K. Mok and D. Chen, "A multiframe model for real-time tasks," *IEEE Trans. Softw. Eng.*, vol. 23, no. 10, pp. 635–645, Oct. 1997.

[8] M. Stigge, P. Ekberg, N. Guan, and W. Yi, "On the tractability of digraph-based task models," in *Proc. ECRTS*, 2011, pp. 162–171.

[9] E. Fersman, P. Krcál, P. Pettersson, and W. Yi, "Task automata: Schedulability, decidability and undecidability," *Inf. Comput.*, vol. 205, no. 8, pp. 1149–1172, 2007.

[10] M. Stigge, P. Ekberg, N. Guan, and W. Yi, "The digraph real-time task model," in *Proc. RTAS*, 2011, pp. 71–80.

[11] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with EDF scheduling," *IEEE Trans. Comput.*, vol. 58, no. 9, pp. 1250–1258, May 2009.

[12] M. Stigge and W. Yi, "Hardness results for static priority real-time scheduling," in *Proc. ECRTS*, 2012, pp. 189–198.

[13] M. Stigge and W. Yi, "Combinatorial abstraction refinement for feasibility analysis," in *Proc. RTSS*, 2013, pp. 340–349.

[14] M. Stigge, N. Guan, and W. Yi, "Refinement-based exact response-time analysis," in *Proc. ECRTS*, 2014, pp. 143–152.

[15] N. Guan, C. Gu, M. Stigge, Q. Deng, and W. Yi, "Approximate response time analysis of real-time task graphs," in *Proc. RTSS*, 2014, pp. 304–313.

[16] N. Guan, Y. Tang, Y. Wang, and W. Yi, "Delay analysis of structural real-time workload," in *Proc. Design Autom. Test Europe (DATE)*, 2015, pp. 223–228.

[17] C. Gu, N. Guan, Z. Feng, Q. Deng, X. S. Hu, and W. Yi, "Transforming real-time task graphs to improve schedulability," in *Proc. RTCSA*, 2016, pp. 29–38.

[18] H. Zeng and M. D. Natale, "Using max-plus algebra to improve the analysis of non-cyclic task models," in *Proc. ECRTS*, 2013, pp. 205–214.
[19] M. Stigge and W. Yi, "Graph-based models for real-time workload: A survey," *Real Time Syst.*, vol. 51, no. 5, pp. 602–636, 2015.
[20] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Resource sharing protocols for real-time task graph systems," in *Proc. ECRTS*, 2011, pp. 272–281.
[21] J. Abdullah, G. Dai, M. Mohaqeqi, and W. Yi, "Schedulability analysis and software synthesis for graph-based task models with resource sharing," in *Proc. RTAS*, 2018, pp. 261–270.
[22] P. Ekberg and W. Yi, "Schedulability analysis of a graph-based task model for mixed-criticality systems," *Real Time Syst.*, vol. 52, no. 1, pp. 1–37, 2016.
[23] M. Mohaqeqi, J. Abdullah, N. Guan, and W. Yi, "Schedulability analysis of synchronous digraph real-time tasks," in *Proc. ECRTS*, 2016, pp. 176–186.
[24] P. Fradet, X. Guo, J.-F. Monin, and S. Quinton, "A generalized digraph model for expressing dependencies," in *Proc. RTNS*, 2018, pp. 72–82.
[25] R. Xu, L. Zhang, N. Ge, and X. Blanc, "Schedulability analysis of real-time tasks with precedence constraints," in *Proc. SEKE*, 2018, pp. 518–527.
[26] S. Ben-Amor, D. Maxim, and L. Cucu-Grosjean, "Schedulability analysis of dependent probabilistic real-time tasks," in *Proc. RTNS*, 2016, pp. 99–107.
[27] J. Sun, "Feasibility of fork-join real-time task graph models: Hardness and algorithms," *ACM Trans. Embedded Comput. Syst.*, vol. 15, no. 1, pp. 1–28, 2016.
[28] H.-E. Zahaf, A. E. H. Benyamina, G. Lipari, R. Olejnik, and P. Boulet, "Modeling parallel real-time tasks with DI-graphs," in *Proc. RTNS*, 2016, pp. 339–348.
[29] H.-E. Zahaf, G. Lipari, M. Bertogna, and P. Boulet, "The parallel multi-mode digraph task model for energy-aware real-time heterogeneous multi-core systems," *IEEE Trans. Comput.*, vol. 68, no. 10, pp. 1511–1524, Oct. 2019.
[30] M. Mohaqeqi, J. Abdullah, and W. Yi, "Modeling and analysis of data flow graphs using the digraph real-time task model," in *Proc. AEiC*, 2016, pp. 15–29.
[31] J. Abdullah, M. Mohaqeqi, and W. Yi, "Synthesis of ADA code from graph-based task models," in *Proc. SAC*, 2017, pp. 1467–1472.
[32] M. Mohaqeqi, J. Abdullah, and W. Yi, "An executable semantics for synchronous task graphs: From sdrt to ada," in *Proc. AEiC*, 2017, pp. 137–152.
[33] Z. Guo and S. K. Baruah, "Uniprocessor EDF scheduling of AVR task systems," in *Proc. ACM/IEEE 6th Int. Conf. Cyber Phys. Syst.*, 2015, pp. 159–168.