Real-Time Syst
DOI 10.1007/s11241-014-9209-5

Computing periodic request functions to speed-up the analysis of non-cyclic task models

Haibo Zeng · Marco Di Natale

Abstract Tasks are units of sequential code implementing the system actions and executed concurrently by an operating system. Techniques have been developed to determine, at design time, whether a set of tasks can safely complete before their deadlines. Several models have been proposed to represent conditional executions and dependencies among concurrent tasks for the purpose of schedulability analysis. Among them, task graphs with cyclic recurrent behavior (i.e., those modeled with a single source vertex and a period parameter specifying the minimum amount of time that must elapse between successive activations of the source job) allow for efficient schedulability analysis based on the periodicity of the request and demand bound functions (*rbf* and *dbf*). In this paper, we leverage results from max-plus algebra to identify a recurrent term in *rbf* and *dbf* of general task graph models, even when the execution is neither recurrent nor controlled by a period parameter. As such, the asymptotic complexity of calculating *rbf* and *dbf* is independent from the length of the time interval. Experimental results demonstrate significant improvements on the runtime for system schedulability analysis.

Keywords Real-time schedulability · Task graph model · Max-plus algebra · Request/Demand bound functions

H. Zeng (✉)
Virginia Tech, 302 Whittemore (0111), Blacksburg, VA 24061, USA
e-mail: hbzeng@vt.edu

M. Di Natale
Scuola Superiore Sant'Anna, Via Moruzzi, 1, 56124 Pisa, Italy
e-mail: marco@sssup.it

Published online: 17 September 2014

1 Introduction

Tasks are units of sequential code implementing the system actions and executed concurrently by an operating system. Techniques have been developed to determine, at design time, whether a set of tasks can safely complete before their deadlines. Several abstract models have been proposed to represent conditional executions and dependencies among real-time concurrent tasks for the purpose of schedulability analysis. The available models can be classified based on the concept of *task graph*, as in Fig. 1. In the graph, vertices represent different kinds of jobs, and edges represent the possible flows of control. Each vertex (job) is characterized by its worst-case execution time requirement and relative deadline. Each graph edge is labeled with the minimum separation time between the release of the two vertices (jobs) it connects.

The schedulability analysis of the task models requires the calculation of the request and demand bound functions. In this paper, we leverage results from max-plus algebra to identify a recurrent term in *rbf* and *dbf* of general task graph models, even when the execution is neither recurrent nor controlled by a period parameter. This greatly improves the efficiency of *rbf* and *dbf* calculation, and consequently schedulability analysis. In the following, we first introduce the task graph models and the concepts of *rbf* and *dbf* functions, before explain the contribution of the paper.

1.1 Task graph models

The list of proposed task graph models are summarized in Fig. 1. A *single vertex* task graph (Fig. 1a) corresponds to the simplest model of independent tasks Liu and Layland (1973) activated by periodic or sporadic events.

The multiframe Mok and Chen (1996) and generalized multiframe (GMF) task models Baruah et al. (1999) (Fig. 1b) assume that worst-case execution times are not constant, but are defined according to a cyclic pattern. The corresponding task graph is therefore a *chain of vertices*. A number of papers has analyzed the schedulability of the GMF task model, including Zuhily and Burns (2009).

The recurring branching task model Baruah (1998) (Fig. 1c) allows selection points to determine the behavior for a given task instance, in statements such as "if-then-else" and "switch-case". Hence, it models conditional branches and optional (OR-type) executions. The corresponding task graph is a *directed tree*.

The recurring real-time task model Baruah (2003), as illustrated in Fig. 1(d), allows the task graph to be any *directed acyclic graph* (DAG).

All the above four models (as in the top row of Fig. 1) satisfy the property of *cyclic recurrent behavior*:

- **recurrent**: the graph has a **unique source vertex**. The completion of a sink vertex automatically releases the source job. This execution pattern may be implicit, or it can be modeled by explicitly adding back edges from the sink vertices to the unique source (as in Fig. 1).
- **cyclic**: a **period** parameter defines the minimum time interval that must elapse between two consecutive releases of the source vertex job.

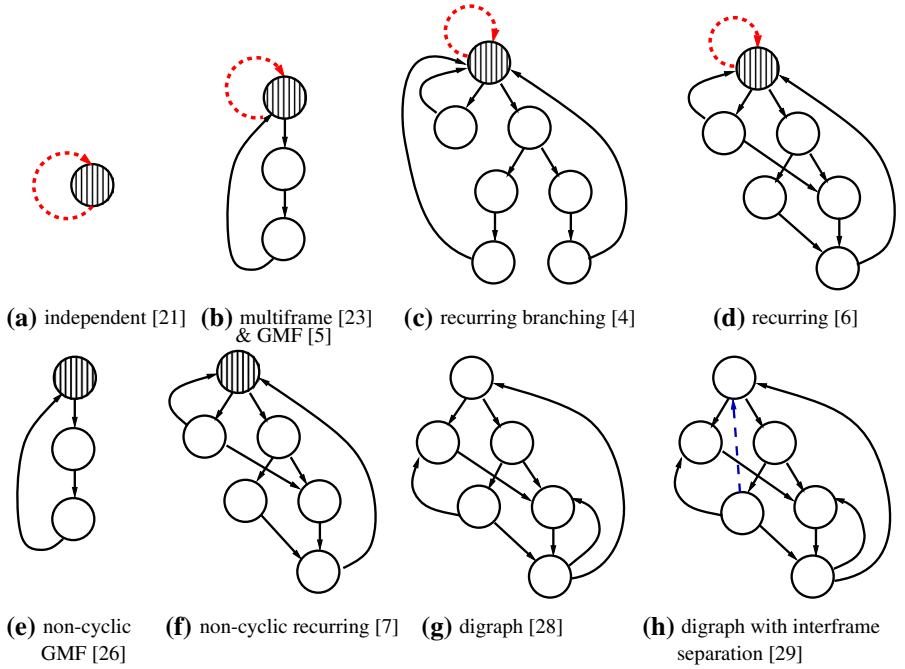

Fig. 1 A summary of the existing task models

In these cyclic recurrent models, the unique source vertex is represented in the figure by a *shadowed node*, with the periodicity constraint represented by a *dotted line* as a self loop on it. A *solid line* indicates a precedence constraint associated with minimum inter-release time.

The non-cyclic generalized multiframe model Moyo (2010) (Fig. 1e) removes the periodicity in the activation pattern of the jobs. More specifically, it is possible to activate any job as long as the minimum separation time with respect to its predecessor has passed. Without considering the (implicit) back edges from the sink vertices to the source vertex, the task graph is still a *directed tree*.

The next task graph model is the one proposed by Anand et al. (2008) (called Recurring Task Model with Branching and Control variables), or in a similar form by Baruah (2010) (as the non-cyclic recurring real-time task model). As in Fig. 1f, it is a generalization of both the recurring real-time Baruah (2003) and the non-cyclic GMF Moyo et al. (2010) models. Such a graph model allows for branches of different length (*anisochronicity*), thus there is no single period parameter characterizes the cyclic behavior of the task graph. The analysis is done by approximating the non-cyclic task with a cyclic (called *isochronous*) one.

These three models (Moyo et al. (2010) Anand et al. (2008) Baruah (2010)) relax the constraint of cyclic execution of the graph, but still assume a recurrent activation model of a source job. In addition, they typically satisfy the **frame separation property**, where the deadline of a job is constrained to be no larger than the inter-release times of its outgoing edges.

The digraph model Stigge et al. (2011) removes the restriction of recurrence by allowing arbitrary cycles and therefore *arbitrary directed graphs* (Fig. 1g). This allows to represent arbitrary release structure of jobs, which significantly increases the expressiveness. Another generalization is to relax the frame separation property to allow arbitrary job deadlines. The digraph model is used for the analysis of implementations of synchronous finite state machines Zeng and Di Natale (2012). The schedulability analysis of digraph tasks under static priority schedulers is shown to be strongly coNP-hard Stigge and Yi (2012). Also, the technique of combinatorial abstraction refinement Stigge and Yi (2013) is introduced to cope with the combinatorial explosion in the schedulability analysis under static priorities.

Stigge et al. (2011) further extend the digraph model by allowing *global inter-release separation constraints* between non-adjacent job release (denoted by *dashed lines* in Fig. 1h). The analysis on the resulting task model is **tractable**, i.e., no worse than pseudo-polynomial time for bounded utilization systems. The model of timed automata with tasks Norström et al. (1999) is a generalization of all the above models, which allows complex dependencies between job release times and task synchronization. However, schedulability analysis is shown to be very expensive and even undecidable in certain variants of the model Fersman et al. (2007). A discussion on the expressiveness and complexity of schedulability analysis for task graph models can be found in Stigge et al. (2011).

A special subset of task digraphs consists of **strongly connected graphs**, where every node is reachable from any other node.[1] We expect many applications of practical interests to be represented by such graphs, because of the need to bring back the system to a (possibly initial) controlled state, which could be a safe state in case of safety-critical systems. For all graphs where there is a unique initial action, a strongly connected graph simply requires that all the actions are reachable (which is intuitively satisfied) and that the initial action is reachable again from any other action. This is the case in most systems, either because of the structure of the code in normal execution (the initial action is, for example, reading a set of sensors that need always be considered), or because of exception handling code. In the latter case, typically a recovery action or state restore action is performed and then execution restarts from the initial default action. All examples in Fig. 1 are strongly connected graphs.

There is also a large body of research that deals with the task graph on multi-core architectures, to explore the parallelism provided by such hardware platforms. It includes the fork-join task graph Saifullah et al. (2011) and its schedulability analysis (e.g., Axer et al. (2013)), the generalized parallel task model Baruah et al. (2012) and its schedulability analysis under multi-task environment Bonifaci et al. (2013). The later represents each task a directed acyclic graph (thus more general than the fork-join task graph). However, these papers have a few differences compared to our work:

- They allow the explicit modeling of possible parallel execution of jobs in a task, while our work only considers sequential execution for jobs of the same task.
- They assume deadline is a task-level parameter, while in the digraph model (our focus) it is job specific.

[1] Tarjan's algorithm Tarjan (1972) can decide in linear time whether a digraph is strongly connected.

– They only have a task-level period parameter to separate the releases of the task instances, while our paper has more complicated mechanisms including inter-release time denoted with the edge between two jobs.

1.2 Request/Demand bound functions

The concepts of **request bound function** (or *rbf*) and **demand bound function** (or *dbf*) have been introduced Baruah (2003) for the analysis of task graphs.

Definition 1 For a task τ, the maximum cumulative execution times by its jobs that have their release times within any time interval of length t is defined as its **request bound function** $\tau.rbf(t)$.

Definition 2 For a task τ, the maximum cumulative execution times by its jobs that have their release times **and deadlines** within any time interval of length t is defined as its **demand bound function** $\tau.dbf(t)$.

Intuitively, *rbf* can be used to compute the maximum amount of execution time that can interfere with a task (by adding the *rbf*s of higher priority tasks). *dbf* quantifies the amount of execution time from jobs that are released and must be completed within a given time interval. Schedulability analysis based on these two functions has been proposed for systems with static and dynamic priority scheduling. The analysis is **tractable**, i.e., no worse than pseudo-polynomial time for bounded utilization systems.

With the exception of timed automata, in all the task models, edges are labeled with a minimum inter-release time. This can be restrictive if job releases are determined by run-time events. For example, it is not straightforward to capture the periodicity of trigger events for multi-rate synchronous finite state machines Zeng and Di Natale (2012).

The periodicity of the request and demand bound functions for the task graph models is of special interest in this paper. For task models with a *cyclic recurrent behavior*, a repeating pattern of the functions *rbf(t)* and *dbf(t)* allows to compute them for large t based on the values of small t. This makes the complexity of computing $rbf(t)$ and $dbf(t)$ independent from t. However, such a periodicity is only studied for task models with a cyclic recurrent behavior.

1.3 Our contributions

In this paper, we study the periodicity of the request and demand bound functions for non-cyclic task models, including the non-cyclic generalized multiframe model Moyo et al. (2010), the non-cyclic recurring task model Baruah (2010), the digraph real-time task model Stigge et al. (2011) and its extension Stigge et al. (2011). We present our results using the digraph real-time task model Stigge et al. (2011), since it is a strict generalization of the models in Moyo et al. (2010) Baruah (2010), and an extended digraph task can be transformed into a plain digraph task with the same *rbf* and *dbf* functions Stigge et al. (2011).

- We make a key connection of *rbf* and *dbf* to the matrix power in max-plus algebra Baccelli et al. (1992), and leverage the related research results (e.g. Molnárová (2005)) to prove their **linear periodicity** Baccelli et al. (1992), i.e., they can be represented by a finite aperiodic part and a periodic part repeated infinitely often. The required proof technique is significantly different from the previous work Zeng and Di Natale (2012) (which provides initial results on the periodicity of execution matrix for synchronous state machines), including the consideration of the sequence of maximum elements of matrix power and the required task graph transformation.
- We develop efficient algorithms to calculate the periodicity parameters for strongly connected task graphs.
- The task graph transformation is then used to derive a tight upper bound on *rbf* and *dbf* functions, and consequently the time interval on which to check feasibility conditions, which improves upon Stigge et al. (2011).
- We prove that the linear periodicity also holds for task graphs with arbitrary deadlines, by considering the periodicity of multi-dimensional arrays (instead of matrices).

Many schedulability techniques rely on such a linear periodicity to provide an efficient representation and computation of the system timing behavior. For example, the Modular Performance Analysis (MPA) toolbox Wandeler and Thiele (2006) based on Real-Time Calculus Thiele et al. (2000) (and in turn, on the min-plus/max-plus algebra) only supports infinite curves with linear periodicity property Künzli et al. (2007). Also, as described in the paper Künzli et al. (2007), linear periodic curves can be safely approximated by the *standard event model*, the one used to describe input/output event streams in SymTA/S Henia et al. (2005). Our paper, for the first time, demonstrates the linear periodicity of non-cyclic task models. **Thus, it opens the possibility of using these task models to represent application structures within these schedulability analysis frameworks**.

The rest of the paper is organized as follows. We first introduce in Sect. 2 the context of our research, including the digraph task model and its schedulability analysis. In Sect. 3 we briefly introduce the background on max-plus algebra and the matrix power sequence under it. In Sect. 4 we prove the main theorem of the paper on the periodicity of the *rbf* and *dbf* functions for digraph real-time tasks. In Sect. 5 we present methods for computing the attributes of periodic *rbf* and *dbf*. In Sect. 6 we provide an improved upper bound on the time interval on which to check feasibility conditions. In Sect. 7 we discuss the case of arbitrary deadlines. Section 8 shows the experimental results demonstrating the significant improvements of our analysis over existing methods. Finally, we conclude the paper in Sect. 9.

2 Digraph task model and schedulability analysis

2.1 Digraph task model and its extension

A **digraph real-time task** (DRT) τ is characterized by a directed graph $\mathcal{D}(\tau) = (\mathbb{V}, \mathbb{E})$ where the set of n vertices $\mathbb{V} = \{v_1, v_2, \ldots, v_n\}$ represents the types of jobs that can

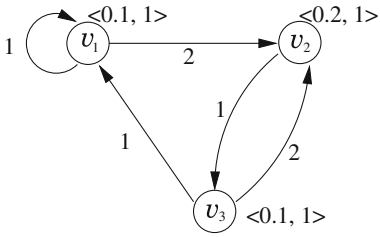

Fig. 2 An example of digraph real-time task

be released for task τ. Each vertex $v_i \in \mathbb{V}$ (or type of job) is characterized by an ordered pair $\langle e(v_i), d(v_i) \rangle$, where $e(v_i)$ and $d(v_i)$ denote its worst-case execution time (WCET) and relative deadline, respectively. Edges represent possible flows of control, i.e., the release order of the jobs of τ. An edge $(v_i, v_j) \in \mathbb{E}$ is labeled with a parameter $p(v_i, v_j)$ that denotes the minimum separation time between the releases of v_i and v_j. We assume that relative deadlines and minimum inter-release times are in $\mathbb{N}^+$ (i.e., they are *positive integers*).

An **event** δ of τ is a pair (t, v) which denotes the release of a job $v \in \mathbb{V}$ at time t. An **event sequence** Δ is a (possibly unbounded) sequence of job release events. A **legal** event sequence $\Delta = [(t_1, v_1), (t_2, v_2), \ldots]$ corresponds to a (potentially infinite) path $(v_1, v_2, \ldots)$ in $\mathcal{D}(\tau)$, in which $\forall i \geq 1$, $(v_i, v_{i+1}) \in \mathbb{E}$, and the release times satisfy $t_{i+1} \geq t_i + p(v_i, v_{i+1})$. A legal event sequence $\Delta = [(t_i, v_i)]$ is called **urgent** if each t_i is the minimum for Δ to be legal, i.e., for an arbitrarily small $\epsilon > 0$ and any i, $\Delta' = [(t_1, v_1), \ldots, (t_i - \epsilon, v_i), \ldots]$ is illegal.

We assume that the tasks satisfy the **l-MAD** property Baruah et al. (1999):

$$\forall (v_i, v_j) \in \mathbb{E}, d(v_i) \leq d(v_j) + p(v_i, v_j) \tag{1}$$

That is, for each edge, the deadline of the sink job is no smaller than the source job. This constraint is less restrictive than the frame separation property, but is sufficient to guarantee that the absolute deadline of the last job is the largest among all jobs in any legal event sequence. In Sect. 7, we relax this constraint and consider the case of arbitrary deadlines.

Example 1 Figure 2 shows an example of a digraph real-time task with 3 vertices (types of jobs), which satisfies the l-MAD property. The node v_1 is associated with a pair $\langle 0.1, 1 \rangle$, which indicates that v_1 has a WCET of 0.1 and a deadline of 1. The edge (v_1, v_2) has a weight of 2, meaning the release times of v_1 and v_2 is separated by at least 2 time units. For all the vertices, Eq. (1) is satisfied, thus the task satisfies the l-MAD property. The event sequence $\Delta = [(5, v_1), (7, v_2), (9, v_3)]$ is legal but not urgent, as the third job (associated with v_3) could be released at time 8.

The **extended digraph real-time task model** (EDRT) generalizes the digraph task model by allowing a set of additional constraints $H(\tau)$ to express a minimum separation time between any two jobs. Each constraint $h = (v_f, v_t, \gamma(v_f, v_t)) \in H(\tau)$ specifies that a minimum time interval $\gamma(v_f, v_t)$ must occur between the releases of v_f and v_t. Each $\gamma(v_f, v_t)$ is assumed to be a positive integer. An EDRT can be

transformed into an equivalent plain digraph task (with the same *rbf* and *dbf*) Stigge et al. (2011). The transformation procedure iteratively replaces global inter-release constraints from the EDRT with additional vertices and adjusts the edge weights.

A **task system** Γ consists of a set of independent real-time tasks $\tau_1, \tau_2, \ldots, \tau_m$. We assume that tasks are scheduled on **a uni-processor with preemptive scheduling**.

2.2 Schedulability analysis

We review the schedulability analysis techniques developed for systems scheduled with dynamic or static priority. For **dynamic priority** scheduling, Earliest Deadline First (EDF) is optimal for independent tasks on a preemptive uni-processor Liu and Layland (1973).

Theorem 1 (Stigge et al. (2011)) *A task system* Γ *is schedulable by dynamic priority (EDF) if and only if the sum of the demand bound functions for all tasks over any time interval does not exceed the length of the interval, that is,*

$$\forall t \geq 0, \sum_{\tau \in \Gamma} \tau.dbf(t) \leq t \tag{2}$$

For a task system scheduled with **static priority**, schedulability is guaranteed if, for each task, the available cpu time is no smaller than the total execution time required by the task itself and all the higher priority tasks.

Theorem 2 (Baruah (2003)) *For a task system* Γ *with static priorities, task* $\tau_i \in \Gamma$ *is schedulable if and only if*

$$\forall t \geq 0, \exists t' \leq t \text{ such that } \tau_i.dbf(t) + \sum_{\tau_j \in hp(i)} \tau_j.rbf(t') \leq t' \tag{3}$$

where $hp(i)$ *is the set of tasks with priority higher than* τ_i.

The above theorem can be derived from the one in Baruah (2003) (originally for the analysis on recurring real-time tasks), as the proof also applies to the extended digraph task model.

In practice, the schedulability of dynamic priority systems is analyzed by checking whether there exists a counterexample to Theorem 1:

$$\exists t \geq 0 \text{ such that } \sum_{\tau \in \Gamma} \tau.dbf(t) > t$$

A similar approach can be derived for static priority systems. Thus, schedulability analysis requires the efficient computation of the *rbf* and *dbf* functions of a task τ over a time interval of given length t. Also, since it is practically impossible to check the schedulability condition for all (integer) $t \geq 0$, an upper bound t_f on such a counterexample is defined in Sect. 6, which improves upon Stigge et al. (2011).

For the periodic task model Liu and Layland (1973), the *rbf* and *dbf* functions of task τ with period p and WCET e are

$$\tau.rbf(t) = \left\lceil \frac{t}{p} \right\rceil e, \quad \tau.dbf(t) = \left\lfloor \frac{t}{p} \right\rfloor e$$

For other cyclic recurrent task graphs, such as the most generic one (the recurring task model Baruah (2003)), every cycle in the graph contains the unique source node for which two consecutive releases are separated by the period parameter. Such properties lead to a regular repeating pattern of *rbf* and *dbf*. For a recurring task with period p, the maximum execution request for any path from the unique source node to a sink node is denoted as e. Its *rbf* and *dbf* satisfy the following equations for sufficiently large r (where $q = \frac{e}{p}$) Baruah (2003)

$$\forall j \in \mathbb{N}^+, \begin{cases} \tau.rbf(r + j \cdot p) = \tau.rbf(r) + j \cdot p \cdot q \\ \tau.dbf(r + j \cdot p) = \tau.dbf(r) + j \cdot p \cdot q \end{cases} \tag{4}$$

This allows the calculation of *rbf* and *dbf* functions with a complexity that is asymptotically independent from the time interval. However, such a property has not been demonstrated for tasks without cyclic recurrent properties.

3 Max-plus algebra

In this section, we introduce the necessary background on max-plus algebra and the periodicity of matrix power. The max-plus algebra Baccelli et al. (1992) is defined over $\mathbb{R}^* = \mathbb{R} \bigcup \{-\infty\}$ where the **addition** (denoted by $\oplus$) and **multiplication** (denoted by $\otimes$) operations are defined as

$$a \oplus b = \max(a, b), \quad a \otimes b = a + b$$

Note that the element $-\infty$ is neutral with respect to $\oplus$, i.e.,

$$\forall a \in \mathbb{R}^*, \ a \oplus (-\infty) = a$$

Likewise, 0 is neutral with respect to $\otimes$, i.e.,

$$\forall a \in \mathbb{R}^*, \ a \otimes 0 = a$$

Example 2 As an example of the max-pus operators,

$$5 \oplus 8 = \max(5, 8) = 8, \quad 5 \otimes 8 = 5 + 8 = 13$$

3.1 Matrix and its power sequence

The matrix operations over $\mathbb{R}^*$ are defined in the same way as the matrix operation over any field. For example, for matrix $\mathbf{A}$ and $\mathbf{B}$ with the same dimension, $\mathbf{C} = \mathbf{A} \oplus \mathbf{B}$ is defined by taking the maximum between the corresponding elements of $\mathbf{A}$ and $\mathbf{B}$, i.e., $c_{i,j} = a_{i,j} \oplus b_{i,j}, \forall i, j$.

For two matrices $\mathbf{A} \in \mathbb{R}^*(m, k)$ and $\mathbf{B} \in \mathbb{R}^*(k, n)$, the result of the multiplication is a matrix $\mathbf{C} \in \mathbb{R}^*(m, n)$, where

$$c_{i,j} = \bigoplus_{l=1}^{k} \left(a_{i,l} \otimes b_{l,j} \right) = \max_{l=1}^{k} \left(a_{i,l} + b_{l,j} \right)$$

The k-th power of a square matrix $\mathbf{A} \in \mathbb{R}^*(n, n)$, denoted as $\mathbf{A}^{(k)}$, is recursively defined as the multiplication of $\mathbf{A}^{(k-1)}$ and $\mathbf{A}^{(1)} = \mathbf{A}$.

$$a_{i,j}^{(k)} = \max_{l=1}^{n} \left(a_{i,l}^{(k-1)} + a_{l,j} \right)$$

The properties of $\mathbf{A}$ are assessed through its graph $\mathcal{G}(\mathbf{A})$.

Definition 3 The **graph $\mathcal{G}(\mathbf{A})$ of a square matrix $\mathbf{A} \in \mathbb{R}^*(n, n)$** is a weighted digraph $(\mathbb{V}, \mathbb{E}, w)$ with nodes $\mathbb{V} = \{1, \ldots, n\}$. Every finite $a_{i,j}$ defines an **edge** $(i, j) \in \mathbb{E}$ weighted by its value $a_{i,j}$. If $a_{i,j} = -\infty$, there is no edge from i to j. A **path** Π in $\mathcal{G}$ is a sequence of nodes $(i_1, i_2, \ldots, i_{t+1})$ where each (i_k, i_{k+1}) is an edge in $\mathbb{E}$. The **length** $|\Pi|$ of Π is t, the number of **edges** in the path. If $i_1 = i_{t+1}$, Π is called a **cycle**. The **weight** of a path Π, $w(\Pi)$, is the sum of the weights of its edges. For a cycle c with length $|c| > 0$, its **cycle mean** $\bar{w}(c)$ is the ratio between its weight and length, i.e., $\bar{w}(c) = w(c)/|c|$. The maximum mean of any cycle in $\mathcal{G}(\mathbf{A})$ is denoted as $\lambda(\mathbf{A})$.

$\mathcal{G}(\mathbf{A})$ is **strongly connected** if all its nodes are contained in a common cycle. In this case, $\mathbf{A}$ is defined as **irreducible**.

Definition 4 Given a subset of the vertexes $\mathbb{K} \subseteq \mathbb{V}$ defining a strongly connected component $\mathcal{K} = (\mathbb{K}, \mathbb{E} \bigcap (\mathbb{K} \times \mathbb{K}))$ of $\mathcal{G}(\mathbf{A})$, its maximum cycle mean $\lambda(\mathcal{K})$ is defined as the maximum of $\bar{w}(c)$ where $c \subseteq \mathcal{K}$. $\mathcal{K}$ is called a **highly connected component** of $\mathcal{G}(\mathbf{A})$ if $\lambda(\mathcal{K}) = \lambda(\mathbf{A})$. $HCC^*(\mathcal{G}(\mathbf{A}))$ denotes the set of highly connected components with a cycle. The **high period** of $\mathcal{K} \in HCC^*(\mathcal{G}(\mathbf{A}))$ is defined as

$$\text{hper}(\mathcal{K}) = gcd\left\{ |c| : c \text{ is a cycle in } \mathcal{K}, \bar{w}(c) = \lambda(\mathbf{A}) \right\}$$

Definition 5 An **elementary path** is a path with no cycle. The operation of **cycle deletion** replaces a cycle $(i_1, i_2, \ldots, i_1)$ with a single node i_1. Given two paths Π and Π', Π' is a **cycle extension** of Π, denoted as $\Pi \subseteq_c \Pi'$, if Π can be created from Π' by a finite number of cycle-deletions.

The **set of paths** of $\mathcal{G}(\mathbf{A})$ from node i to j is denoted as $\mathbb{P}_{\mathcal{G}(\mathbf{A})}(i, j)$. The **set of elementary paths** is denoted as $\mathbb{P}^*_{\mathcal{G}(\mathbf{A})}(i, j)$. The subset of the paths of length t is $\mathbb{P}^t_{\mathcal{G}(\mathbf{A})}(i, j)$. The **power sequence for an elementary path** $\Pi \in \mathbb{P}^*_{\mathcal{G}(\mathbf{A})}(i, j)$ is the

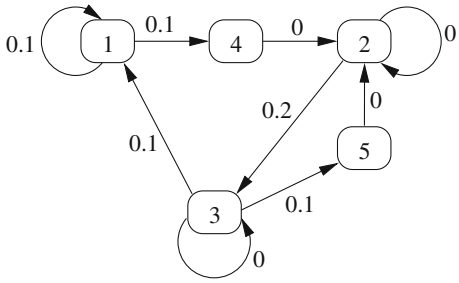

Fig. 3 The digraph corresponding to the matrix of Eq. (6)

sequence of the maximum weights among all the cycle extensions of Π. Each term in the sequence is

$$a_{\Pi}^{(t)} = \max\left\{w(\Pi') : \Pi' \in \mathbb{P}^{t}_{\mathcal{G}(\mathbf{A})}(i, j), \Pi \subseteq_c \Pi'\right\}$$

Intuitively, each element $a_{i,j}^{(t)}$ of $\mathbf{A}^{(t)}$ defines the path of length t with the maximum weight in $\mathcal{G}(\mathbf{A})$ from node i to j. The following fundamental theorem in max-plus algebra defines the relationship between the power of a matrix $\mathbf{A}$ and the maximum weights of the paths in $\mathcal{G}(\mathbf{A})$, and consequently, the power of elementary paths.

Theorem 3 (Baccelli et al. (1992)) *The power sequence of* $\mathbf{A} \in \mathbb{R}^{*}(n, n)$*, for all* $t \in \mathbb{N}^{+}$ *and all* $i, j \in \{1, \ldots, n\}$ *can be computed as*

$$\begin{aligned} a_{i,j}^{(t)} &= \max\left\{w(\Pi') : \Pi' \in \mathbb{P}^{t}_{\mathcal{G}(\mathbf{A})}(i, j)\right\} \\ &= \max\left\{a_{\Pi}^{(t)} : \Pi \in \mathbb{P}^{*}_{\mathcal{G}(\mathbf{A})}(i, j)\right\} \end{aligned} \tag{5}$$

Example 3 Consider the square matrix $\mathbf{A}$ in (6) and its digraph $\mathcal{G}(\mathbf{A})$ in Fig. 3.

$$\mathbf{A} = \begin{bmatrix} 0.1 & -\infty & -\infty & 0.1 & -\infty \\ -\infty & 0 & 0.2 & -\infty & -\infty \\ 0.1 & -\infty & 0 & -\infty & 0.1 \\ -\infty & 0 & -\infty & -\infty & -\infty \\ -\infty & 0 & -\infty & -\infty & -\infty \end{bmatrix} \tag{6}$$

The set of elementary paths from node 3 to node 2 is

$$\mathbb{P}^{*}_{\mathcal{G}(\mathbf{A})}(3, 2) = \left\{(3, 5, 2), (3, 1, 4, 2)\right\},$$

and every other path in $\mathbb{P}_{\mathcal{G}(\mathbf{A})}(3, 2)$ is a cycle extension to either $(3, 5, 2)$ or $(3, 1, 4, 2)$.

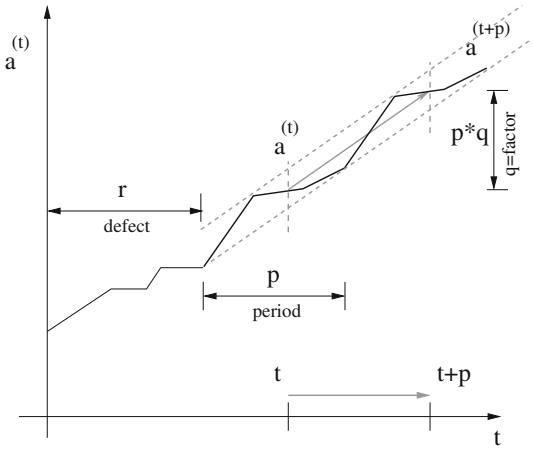

Fig. 4 An example of almost linear periodic sequence

3.2 Linear periodicity and general periodicity

We now review the results on linear periodicity and general periodicity in max-plus algebra. We first introduce the notion of linear periodicity of a sequence or of a matrix, respectively.

Definition 6 A sequence $a^* = \{a^{(t)}\}, t \in \mathbb{N}^+$ is **almost linear periodic**, if there exist a real number $q \in \mathbb{R}$ and a pair of positive integers r and p such that

$$\forall t > r, \quad a^{(t+p)} = a^{(t)} + p \cdot q \tag{7}$$

The smallest p with the above property is the **linear period** of a^*, denoted as $p = \mathrm{lper}(a^*)$. q is the **linear factor** of a^*, or $q = \mathrm{lfac}(a^*)$. Finally, the smallest r with the above property is the **linear defect**, or $r = \mathrm{ldef}(a^*)$.

Definition 7 The matrix $\mathbf{A} = (a_{i,j})$ is defined as **almost linear periodic** if the power sequence $a^*_{i,j}$ of each element $a_{i,j}$ in $\mathbf{A}^* = \{\mathbf{A}^{(t)}\}, t \in \mathbb{N}^+$ is almost linear periodic. The matrix $\mathrm{lfac}(\mathbf{A}^*) = (\mathrm{lfac}(a^*_{i,j}))$ is the **linear factor matrix** of $\mathbf{A}$, the number $\mathrm{ldef}(\mathbf{A}) = \max\{\mathrm{ldef}(a^*_{i,j})\}$ is the **linear defect** of $\mathbf{A}$, and $\mathrm{lper}(\mathbf{A}) = lcm\{\mathrm{lper}(a^*_{i,j})\}$ is the **linear period** of $\mathbf{A}$.

Example 4 Figure 4 provides an example of almost linear periodic sequence. In reality, the power values of the sequence are discrete (the graph shows them as continuous to highlight the periodic recurrence). Intuitively speaking, the linear defect of an almost linear periodic sequence is the length of its transient (non-periodic) part, while its linear factor is the rate at which the sequence increases in the periodic part.

A matrix is almost linear periodic if it is irreducible (as demonstrated in Baccelli et al. (1992)). Gavalec (2000) proposed an $O(n^3)$ algorithm for computing the linear period and factor of an irreducible matrix based on the following theorem:

Theorem 4 (Gavalec (2000)) *An irreducible matrix* $\mathbf{A} \in \mathbb{R}^*(n,n)$ *is almost linear periodic, its linear factor is* $lfac(\mathbf{A}) = \mathbf{Q}$*, with* $q_{i,j} = \lambda(\mathbf{A})$ *for all* i, j*; its linear period is*

$$lper(\mathbf{A}) = lcm\Big\{hper(\mathcal{K}) : \mathcal{K} \in HCC^*(\mathcal{G}(\mathbf{A}))\Big\}. \tag{8}$$

Example 5 For the square matrix $\mathbf{A}$ in (6) and its digraph $\mathcal{G}(\mathbf{A})$ in Fig. 3, we can calculate the power sequence of $a_{1,1}$ as 0.2, 0.2, 0.2, 0.5, 0.6, 0.7, 0.8, ..., which exhibits a steady increase of 0.1 per step after the initial transient phase.

In fact, since $\mathcal{G}(\mathbf{A})$ is strongly connected, $\mathbf{A}$ is irreducible. There are three cycles (1, 1), (2, 3, 5, 2), and (1, 4, 2, 3, 1) with a length of 1, 2, and 3 respectively, and each with a cycle mean of 0.1. Since there is no other cycle with a larger cycle mean, $\lambda(\mathbf{A}) = 0.1$, and the high period of $\mathcal{G}(\mathbf{A})$ (which is a highly connected component itself) is $gcd\{1,2,3\} = 1$. By Theorem 4, $\mathbf{A}$ is almost linear periodic with a linear factor 0.1 and linear period 1. Equation (9) shows how to apply the linear periodicity of $\mathbf{A}$ to compute its power sequence with its linear defect equal to 6.

$$\forall t \geq 6, \mathbf{A}^{(t)} = \begin{bmatrix} 0 & -0.1 & 0 & 0 & 0 \\ 0.1 & 0 & 0.1 & 0.1 & 0.1 \\ 0 & -0.1 & 0 & 0 & 0 \\ 0 & -0.1 & 0 & 0 & 0 \\ 0 & -0.1 & 0 & 0 & 0 \end{bmatrix} + 0.1 \times t \tag{9}$$

When the matrix is reducible, it can still be almost linear periodic, but deciding whether this is the case has been demonstrated to be an NP-complete problem Gavalec (2000). However, even when the matrix is not linear periodic, it is still possible to avoid computing the *rbf* function over a long time interval by leveraging the concept of general periodicity (Molnárová (2005)), a generalization of linear periodicity.

Definition 8 A sequence $a^* = (a^{(t)}), t \in \mathbb{N}^+$ is defined as **almost generally periodic**, if there exist a pair of integers r and p and a vector $Q(i) \in \mathbb{R}^*(i = 1, \ldots, p)$ such that

$$\forall i = 1, \ldots, p, \forall t > r, t \equiv i \ (\text{mod } p), \ a^{(t+p)} = a^{(t)} + p \cdot Q(i) \tag{10}$$

The smallest p with the above property is the **general period** of a^*, or $p = \text{gper}(a^*)$. Q is the **general factor** of a^*, or $Q = \text{gfac}(a^*)$. Finally, the smallest r with the above property is the **general defect**, or $r = \text{gdef}(a^*)$.

Definition 9 Matrix $\mathbf{A}$ is **almost generally periodic** if the power sequence $a^*_{i,j}$ of each element in $\mathbf{A}^* = \{\mathbf{A}^{(t)}\}, t \in \mathbb{N}^+$ is almost generally periodic. $\text{gfac}(\mathbf{A}^*) = (\text{gfac}(a^*_{i,j}))$ is the **general factor matrix** of $\mathbf{A}$, $\text{gdef}(\mathbf{A}) = \max_{i,j}\{\text{gdef}(a^*_{i,j})\}$ is its **general defect**, and $\text{gper}(\mathbf{A}) = lcm\{\text{gper}(a^*_{i,j})\}$ is its **general period**.

Example 6 Figure 5 provides a representation of the power values for a generic $a_{i,j}$ of an almost generally periodic matrix with period p and general factor Q. After the defect, the values of the subset of points $a^{(t)}$ with t modulo $p = i$ are placed on a line

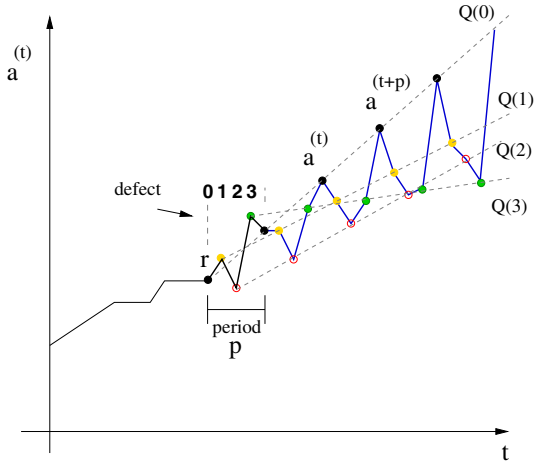

Fig. 5 An example of almost generally periodic sequence

with slope $Q(i)$ (the graph shows the values as joined by a continuous line for clarity). The figure only illustrates the concept of general periodicity. To emphasize the subsets of points belonging to a linear progression, it is not monotonic and therefore is not meant to represent an *rbf* or *dbf* function.

The following theorem states the applicability of the periodicity property to all matrices.

Theorem 5 (Molnárová (2005)) *Every matrix is almost generally periodic over the max-plus algebra.*

Theorem 5 is based on lemmas that show how the power sequence of every element in the matrix is almost generally periodic, and the maximum of two almost generally periodic sequences is almost generally periodic.

The problem of computing the general period gper(**A**) or general factor matrix gfac(**A**) for a given square matrix **A** is shown to be NP-hard Molnárová (2005), where the complexity is expressed in terms of the size of the matrix (or the corresponding graph). However, it is asymptotically independent from the power of the matrix (or the time interval for the *rbf* and *dbf* functions), which is not part of the input to the algorithm.

In the following, we prove that **the** *dbf* **and** *rbf* **functions are long-term periodic for task graphs without cyclic recurrent properties**, including the non-cyclic GMF model Moyo et al. (2010), the non-cyclic recurring task model Baruah (2010), the digraph real-time task (DRT) model Stigge et al. (2011), and its extension (EDRT) Stigge et al. (2011). We present our results using the *digraph task model*, since it is a strict generalization of the non-cyclic GMF and non-cyclic recurring task models, and an EDRT can be transformed into an equivalent plain digraph task with the same *rbf* and *dbf* Stigge et al. (2011).

4 Demonstrating the periodicity of the *rbf* and *dbf*

In this section, we prove the linear periodicity of the *rbf* and *dbf* for a generic digraph task. The proof is built on the following two ideas:

- As the graph edge weight is a representation of the computing load in a given time interval (and the elements of the matrix power represent the processor loads requested in the time interval), demonstrating the periodicity of the *rbf* and *dbf* requires that the sequence of the maximum elements of the matrix powers is also periodic (the periodicity of each element is not enough).
- We provide a task graph transformation to a unit digraph task (UDRT). In UDRT, the total execution time request in a given time interval is encoded as the elements of a max-plus matrix power sequence. This allows to leverage results from max-plus algebra and formally demonstrate the periodicity of *rbf* and *dbf* for non-recurrent graphs.

We first recapture several definitions and lemmas that are useful for our extension to max-plus algebra (Sect. 4.1).

Definition 10 Given an elementary path Π, the set of strongly connected components that includes at least one node of Π is $SCC^*_\Pi(\mathcal{G}(\mathbf{A})) = \{\mathcal{K} : \mathcal{K} \bigcap \Pi \neq \emptyset\}$. The **maximum mean** $\lambda(\Pi)$ of Π is the maximum cycle mean of any element $\mathcal{K}$ in $SCC^*_\Pi(\mathcal{G}(\mathbf{A}))$, and its **period** $\mathrm{lper}(\Pi)$ is defined as the least common multiple of the high periods of highly connected components of $\mathcal{K}$.

$$\lambda(\Pi) = \max\left\{\lambda(\mathcal{K}) : \mathcal{K} \in SCC^*_\Pi(\mathcal{G}(\mathbf{A}))\right\}$$
$$\mathrm{lper}(\Pi) = lcm\left\{\mathrm{hper}(\mathcal{K}') : \mathcal{K}' \in HCC^*(\mathcal{K}), \lambda(\mathcal{K}) = \lambda(\Pi)\right\}$$

If Π does not share any node with any strongly connected component, then $\lambda(\Pi) = -\infty$.

Lemma 6 (Gavalec (2000) Molnárová (2005)) *For an elementary path Π, if its maximum cycle mean $\lambda(\Pi) > -\infty$, then a^*_Π is almost linear periodic with period $lper(\Pi)$ (or its integer divisor) and factor $\lambda(\Pi)$; otherwise, it is almost generally periodic with a general factor of $-\infty$.*

Two other lemmas in Molnárová (2005) provide sufficient conditions for the maximum of two almost linear periodic sequences to be linear periodic.

Lemma 7 (Molnárová (2005)) *Consider two almost linear periodic sequences a^* and b^*, where a^* has period p_a and factor q, and b^* has period p_b and the same factor q. Then the sequence $\max(a^*, b^*)$ is almost linear periodic with period p as an integer divisor of $lcm(p_a, p_b)$, and factor q.*

Lemma 8 (Molnárová (2005)) *Consider two almost linear periodic sequences a^* and b^*. a^* has period p_a and factor q_a. The elements of b^* are* **all finite**, *with period p_b and factor $q_b > q_a$. Then the maximum sequence $\max(a^*, b^*)$ is almost linear periodic with period p_b and factor q_b.*

As a special case of Lemma 7, if the two almost linear periodic sequences a^* and b^* have the same period and factor, the maximum sequence $\max(a^*, b^*)$ is almost linear periodic with the same period and factor, and its defect is $\max\{\text{ldef}(a^*), \text{ldef}(b^*)\}$.

The proof of the linear periodicity of an irreducible matrix (Theorem 4) derives from these lemmas. By Lemma 6 and the fact that the matrix is irreducible (its graph itself is a strongly connected component), for any elementary path Π, a^*_Π is almost linear periodic with period $lcm\{\text{hper}(\mathcal{K}) : \mathcal{K} \in HCC^*(\mathcal{G}(\mathbf{A}))\}$ and factor $\lambda(\mathbf{A})$. By Theorem 3 and Lemma 7, $\forall i, j$, $a^*_{i,j}$ is almost linear periodic with the same period and factor.

4.1 Periodicity of largest element sequence of matrix power

We now prove that the maximum element of a matrix power sequence is linear periodic. *We only consider non-trivial matrices (with at least one cycle in the associated graph).* The demonstration requires additional definitions and lemmas as intermediate steps.

Definition 11 The **largest element sequence** of $\mathbf{A} \in \mathbb{R}^*(n, n)$, denoted as $a^*_{\max} = \{a^{(t)}_{\max}\}, \forall t \in \mathbb{N}^+$, is defined as the sequence of the largest element in the power sequence of $\mathbf{A}$, i.e., $a^{(t)}_{\max} = \max_{i,j} a^{(t)}_{i,j}, \forall t \in \mathbb{N}^+$.

Combining the definition and Theorem 3, the largest element sequence is the maximum among the power of all the elementary paths in the graph.

$$\begin{aligned} a^{(t)}_{\max} &= \max_{i,j} \left\{ a^{(t)}_{\Pi'} : \Pi' \in \mathbb{P}^t_{\mathcal{G}(\mathbf{A})}(i, j) \right\} \\ &= \max \left\{ a^{(t)}_{\Pi} : \Pi \text{ is an elementary path in } \mathcal{G}(\mathbf{A}) \right\} \end{aligned} \tag{11}$$

Definition 12 A generally periodic sequence a^* with factor Q_a is **dominated** by a linear periodic sequence b^* with factor q_b, if b^* is **eventually finite** ($\exists r$ such that $\forall t > r$, $b^{(t)}$ is finite), and each element in Q_a is less than or equal to q_b.

The following lemma generalizes Lemmas 7 and 8, demonstrating the linear periodicity of the maximum sequence of a generally periodic sequence and a dominating linear periodic sequence.

Lemma 9 *Consider a general periodic sequence a^* (period p_a and factor Q_a) and a dominating linear periodic sequence b^* (period p_b and factor q_b). Then the sequence* $\max(a^*, b^*)$ *is almost linear periodic with period as an integer divisor of $p = lcm(p_a, p_b)$ and factor q_b.*

Proof By the definitions of periodicity, there exists an integer r such that for any given $i \in \{1, \ldots, p\}$

$$\forall j \in \mathbb{N}^+, \begin{cases} a^{(r+i+j\cdot p)} = a^{(r+i)} + j \cdot p \cdot Q_a(i) \\ b^{(r+i+j\cdot p)} = b^{(r+i)} + j \cdot p \cdot q_b > -\infty \end{cases}$$

We consider the maximum sequence of $a^{(r+i+j\cdot p)}$ and $b^{(r+i+j\cdot p)}$. There are two cases.

Case 1: $Q_a(i) < q_b$. In this case, even if the sequence b^* has a smaller initial value, because of its dominating linear factor, there must exist an index r_i' such that the elements of b^* with index $\geq r_i'$ will eventually be larger than the corresponding elements of a^*.

We define $s_i = \dfrac{a^{(r+i)} - b^{(r+i)}}{p(q_b - Q_a(i))}$ if $a^{(r+i)} > b^{(r+i)}$; otherwise $s_i = 0$. By simple arithmetics, we can prove that $\forall j \in \mathbb{N}^+$, $\max(a^{(r_i'+j\cdot p)}, b^{(r_i'+j\cdot p)}) = b^{(r_i'+j\cdot p)} = b^{(r_i')} + j \cdot p \cdot q_b$, where $r_i' = r + i + \left\lceil \dfrac{s_i}{p} \right\rceil p$.

Case 2: $Q_a(i) = q_b$. The power sequence of a has the same factor as b. Thus, $\forall j \in \mathbb{N}^+$, $\max(a^{(r_i'+j\cdot p)}, b^{(r_i'+j\cdot p)}) = \max(a^{(r_i')}, b^{(r_i')}) + j \cdot p \cdot q_b$, where $r_i' = r + i$.

Thus, as a generalization of the two cases, $\forall i = 1, \ldots, p$, $\forall j \in \mathbb{N}^+$, there exists an integer $r' = \max_i(r_i')$ such that

$$\max(a^{(r'+j\cdot p)}, b^{(r'+j\cdot p)}) = \max(a^{(r')}, b^{(r')}) + j \cdot p \cdot q_b$$

This means that the maximum sequence of a^* and b^* is linear periodic with period $lcm(p_a, p_b)$ (or an integer divisor of it), and factor q_b. □

We now prove the linear periodicity for the sequence of the largest matrix element. It is done by constructing a subset of the elementary paths whose largest element sequence dominates those of the other paths.

Theorem 10 *The sequence of the largest element of any non-trivial square matrix is almost linear periodic.*

Proof For a non-trivial square matrix $\mathbf{A}$, assume its maximum cycle mean $\lambda(\mathbf{A}) > -\infty$. There must exist a cycle c such that $\bar{w}(c) = \lambda(\mathbf{A})$. Without loss of generality, we denote $c = (1, 2, \ldots, |c|, 1)$. We consider a sequence that is the maximum of the powers of the elementary paths in c: $b^{(t)} = \max\{a_\Pi^{(t)} : \Pi \subset c\}$.

The power sequence of a generic elementary path $\Pi \subset c$ is almost linear periodic with factor $\lambda(\mathbf{A})$. By Lemma 7, b^* is also almost linear periodic with the same factor. Moreover, $\forall t \in \mathbb{N}^+$, we can find a path Π' which is the concatenation of j repetitions of $(1, 2, \ldots, |c|)$ and $\Pi = (1, 2, \ldots, k+1)$, where $j = \lfloor \frac{t}{|c|} \rfloor$ and $k = t - j \cdot |c|$. Π' is a cycle extension of the elementary path Π in c. As $b^{(t)} \geq w(\Pi')$, we have $\forall t \in \mathbb{N}^+$, $b^{(t)}$ is finite.

By Lemma 6, the power sequences of all other (finitely many) elementary paths are generally periodic with a factor no larger than $\lambda(\mathbf{A})$, thus dominated by b^*. By Lemma 9, the sequence of the largest element (the maximum among b^* and those of the other elementary paths) is almost linear periodic.

Example 7 The sequence of the largest element of the irreducible matrix in Eq. (6) can be derived as

$$\forall t \geq 6, \quad a_{\max}^{(t)} = 0.1 + 0.1 \times t \tag{12}$$

4.2 Task transformation

To apply the above extended results on max-plus algebra, **a task transformation and a refinement of the *rbf* and *dbf* functions are needed**. We first present the transformation of a task digraph into an equivalent digraph where all inter-release times are equal to one.[2] Such a digraph task with unit-weighted edges is defined as a **unit digraph task (UDRT)**. For $\mathcal{D}(\tau) = (\mathbb{V}, \mathbb{E})$, we generate an equivalent UDRT $\mathcal{D}'(\tau') = (\mathbb{V}', \mathbb{E}')$ by the following rules:

- Each vertex $v_i \in \mathbb{V}$ corresponds to k_i vertices $v_{i,1}, v_{i,2}, \ldots, v_{i,k_i}$ in $\mathbb{V}'$, where

$$k_i = \max\left\{1, \max_{v_j:(v_i,v_j)\in\mathbb{E}} p(v_i, v_j)\right\}. \tag{13}$$

- $v_{i,1}$ is labeled with an execution time $e(v_{i,1}) = e(v_i)$, and all the other new vertices have zero execution time. Deadlines can be assigned arbitrarily, as we are only interested in the *rbf* function of the transformed graph. The calculation of the *dbf* function does not use UDRT directly, but is done by leveraging its relationship with *rbf* (Eq. (20)).
- For every $k = 1, 2, \ldots, k_i - 1$, an edge $(v_{i,k}, v_{i,k+1}) \in \mathbb{E}'$ is added to connect these nodes in a chain topology.
- Each edge $(v_i, v_j) \in \mathbb{E}$ corresponds to an edge $(v_{i,k}, v_{j,1})$ in $\mathbb{E}'$ where $k = p(v_i, v_j)$.
- All edges in $\mathbb{E}'$ are labeled as 1.

We now prove the following property of the transformation, which allows us to use the transformed UDRT to study the *rbf* of a digraph task.

Lemma 11 *The request bound functions of a digraph task $\mathcal{D}(\tau)$ and the transformed UDRT $\mathcal{D}'(\tau')$ are the same.*

Proof Let $l = |\Delta|$ be the number of events in an arbitrary legal event sequence $\Delta = [(t_i, v_i)]$ in $\mathcal{D}(\tau)$. We construct an event sequence Δ' in $\mathcal{D}'(\tau')$ by replacing each event (t_i, v_i) except the last one in Δ with a sequence of events $(t_i + k - 1, v_{i,k})$ for $k = 1, \ldots, p(v_i, v_{i+1})$. The last event (t_l, v_l) in Δ is replaced with $(t_l, v_{l,1})$. Δ' is legal as the inter-release time between any two events is no smaller than 1. Thus $\tau.rbf(t) \leq \tau'.rbf(t)$. Similarly, we can prove $\tau'.rbf(t) \leq \tau.rbf(t)$. □

In unit digraph task models, in a legal event sequence (and the corresponding path) containing n events (n nodes), the minimum separation time between the release times of the last and first events is $n - 1$. Hence, **the length of the path in τ' represents the inter-release time of the first and last vertices for the corresponding urgent event sequence**.

Example 8 For the digraph task τ in Fig. 2, the transformed unit digraph task τ' is shown in Fig. 6. v_1 in τ is transformed to two vertices $v_{1,1}$ and $v_{1,2}$ in τ', v_2 is transformed to $v_{2,1}$, and v_3 to $v_{3,1}$ and $v_{3,2}$.

[2] It is sufficient to transform the task into a digraph with inter-release times equal to $gcd(p(e) : e \in \mathbb{E})$.

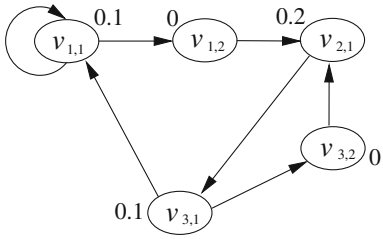

Fig. 6 The unit digraph task transformed from Fig. 2. The minimum inter-release times are all equal to 1. Deadlines are omitted

We extend the definition of *rbf* by adding the indication of the initial and final jobs of the path on which it is computed. Also, the definition of the refined *rbf* is slightly modified by removing the execution time (with its release time) of the last job. Both are necessary to simplify the formulation of *rbf* in the max-plus algebra, allowing a simple composition operation (as explained below).

Definition 13 Given a pair of vertices v_i and v_j, the **request bound function** of a digraph task τ in the time interval t, denoted as $\tau.rbf(v_i, v_j, t)$, is defined as the maximum sum of execution times of any legal event sequence $[(t_k, v_k), k = 1, \ldots, l]$ of τ such that

- the vertex corresponding to the first event is $v_1 = v_i$;
- the vertex corresponding to the last event is $v_l = v_j$;
- $t_{l-1} - t_1 \leq t$;
- $\tau.rbf(v_i, v_j, t) = \max \sum_{k=1}^{l-1} e(v_k)$.

If v_j is not reachable from v_i within any time interval of length t, then we define $\tau.rbf(v_i, v_j, t) = -\infty$. With this definition, the domain for the possible *rbf* values is $\mathbb{R}^* = \mathbb{R} \bigcup \{-\infty\}$.

According to the definition, the *rbf* function of the task is the maximum among all the pairs v_i and v_j

$$\tau.rbf(t) = \max_{i,j} \left\{ \tau.rbf(v_i, v_j, t) \right\} \tag{14}$$

For a UDRT τ, $rbf(v_i, v_j, t)$ is additive, i.e., $\forall i, j, \forall t_1, t_2$,

$$\tau.rbf(v_i, v_j, t_1 + t_2 + 1) = \max_{m} \left\{ \tau.rbf(v_i, v_m, t_1) + \tau.rbf(v_m, v_j, t_2) \right\} \tag{15}$$

Thus, $rbf(v_i, v_j, t_1 + t_2 + 1)$ of a UDRT τ for a long interval of length $t_1 + t_2 + 1$ can be computed from its values for shorter intervals of length t_1 and t_2. The need for this composition explains the omission of the execution time of the last job, and the sum of the time intervals is incremented by one when concatenating two paths, as the inter-release time of (v_{m-1}, v_m) is not included in the interval t_1 or t_2. Dynamic programming techniques can be used for an efficient calculation of $rbf(v_i, v_j, t)$. In addition, the computation can be represented as a matrix power sequence in max-plus algebra. Thus, we can leverage the related results to find its periodicity.

For a generic digraph task, Eq. (15) does not apply since the inter-release times are not uniformly one. Also, the general *rbf* function (without constraints on the source and sink vertices) is not additive:

$$\begin{aligned}\tau.rbf(t_1) + \tau.rbf(t_2) &= \max_{i,k}\left\{\tau.rbf(v_i, v_k, t_1)\right\} + \max_{l,j}\{\tau.rbf(v_l, v_j, t_2)\Big\} \\ &= \max_{i,j,k,l}\left\{\tau.rbf(v_i, v_k, t_1) + \tau.rbf(v_l, v_j, t_2)\right\} \\ &\geq \max_{i,j,k=l}\left\{\tau.rbf(v_i, v_k, t_1) + \tau.rbf(v_l, v_j, t_2)\right\} \\ &= \tau.rbf(t_1 + t_2 + 1)\end{aligned}$$

Thus, dynamic programming techniques and max-plus algebra cannot directly help speedup the computations of the *rbf* functions. **The transformation to a unit digraph task and the refinement of its *rbf* function are necessary**.

In the next subsection, we demonstrate the periodicity of the *rbf* function for a UDRT. With this result, by Lemma 11, the original digraph also has an almost linear periodic *rbf*. The periodicity of its *dbf* function is then proved by making a connection to its *rbf* function.

4.3 Periodicity of *rbf*

For a UDRT τ with n vertices, we define the **execution request matrix** as $\mathbf{A} \in \mathbb{R}^*(n, n)$, with elements $a_{i,j} = \tau.rbf(v_i, v_j, 0)$. Generalizing to any time interval t, we denote $a_{i,j}^{(t+1)} = \tau.rbf(v_i, v_j, t)$. Now Eq. (15) can be rewritten as (denoting $k = t_1 + 1$ and $l = t_2 + 1$, then $k + l = t_1 + t_2 + 2$)

$$\forall i, j, \ \forall k, l \quad a_{i,j}^{(k+l)} = \max_m \left(a_{i,m}^{(k)} + a_{m,j}^{(l)}\right) \tag{16}$$

This is exactly the definition of matrix power under the max-plus algebra. In other words, **the *rbf* function of a UDRT τ over a time interval of length t can be expressed as the $(t+1)$-th power of its execution request matrix $\mathbf{A}$**.

Informally, we can derive $\mathcal{G}(\mathbf{A})$ from $\mathcal{D}(\tau')$ by:

- duplicating the topology of $\mathcal{D}(\tau')$;
- adding a self-loop with an edge of weight 0 for each vertex $v_{i,1}$. This is to allow the possibility of untight event sequences. Thus, $\mathbf{A}$ is *non-trivial*.
- assigning the edge (u, v) in $\mathcal{G}(\mathbf{A})$ with the same weight as the source node u in $\mathcal{D}(\tau')$.

Example 9 Equation (6) computes the execution request matrix $\mathbf{A}$ for the UDRT $\mathcal{D}(\tau')$ in Fig. 6. $\mathcal{D}(\tau')$ and $\mathcal{G}(\mathbf{A})$ (Fig. 3) are similar, with corresponding vertices: $v_{1,1} \Leftrightarrow 1$, $v_{2,1} \Leftrightarrow 2$, $v_{3,1} \Leftrightarrow 3$, $v_{1,2} \Leftrightarrow 4$, and $v_{3,2} \Leftrightarrow 5$.

For the example, the linear periodicity of matrix $\mathbf{A}$ and the *rbf* function is demonstrated in Eqs. (9) and (12) respectively.

We now prove the periodicity of *rbf* for a digraph task.

Theorem 12 *The rbf function of a generic digraph task τ is almost linear periodic, i.e., there exist a real number q and a pair of integers r and p such that*

$$\forall t > r,\ \tau.rbf(t+p) = \tau.rbf(t) + p \cdot q \tag{17}$$

Proof We consider the unit digraph task τ' obtained from τ and its execution request matrix $\mathbf{A}$. By Eq. (14),

$$\tau.rbf(t) = \tau'.rbf(t) = \max_{i,j} a_{i,j}^{(t+1)} = a_{\max}^{(t+1)} \tag{18}$$

i.e., $\tau.rbf(t)$ is the maximum element of the $(t+1)$-th power of its execution request matrix. The linear periodicity of the *rbf* function of τ follows immediately from Theorem 10. □

4.4 Periodicity of *dbf*

We now prove the periodicity of the *dbf* function for a digraph task. The definition of *dbf* is also extended to include a restriction to a given pair of start and end vertices.

Definition 14 Given a pair of vertices v_i and v_j, the **demand bound function** of a digraph task τ during the time interval t, denoted as $\tau.dbf(v_i, v_j, t)$, is defined as the maximum sum of execution times of any legal event sequence $[(t_k, v_k), k = 1, \ldots, l]$ of τ such that

- the vertex corresponding to the first event is $v_1 = v_i$;
- the vertex corresponding to the last event is $v_l = v_j$;
- $\forall k = 1, \ldots, l, d_k + t_k - t_1 \leq t$;
- $\tau.dbf_{i,j}(t) = \max \sum_{k=1}^{l} e(v_k)$.

Different from *rbf*, **we define *dbf* to include the execution time and deadline of the last job**, since it is not computed by composition, but by starting from the corresponding *rbf*.

Theorem 13 *The dbf function of a generic digraph task $\mathcal{D}(\tau) = (\mathbb{V}, \mathbb{E})$ is almost linear periodic, i.e., there exist a real number q and a pair of integers r and p such that*

$$\forall t > r,\ \tau.dbf(t+p) = \tau.dbf(t) + p \cdot q \tag{19}$$

Proof By the **l-MAD** property, the last job in an event sequence always has the latest deadline, thus the execution times of all nodes in the sequence should be included in the *dbf*. For the consideration of $\tau.dbf(v_i, v_j, t)$, the second to last node $v_k : (v_k, v_j) \in \mathbb{E}$ has a release time of $t - d(v_j) - p(v_k, v_j)$. For sufficiently large t, it is

$$\begin{aligned}\tau.dbf(v_i, v_j, t) &= \max_{v_k:(v_k,v_j)\in\mathbb{E}} \left\{\tau.rbf(v_i, v_j, t - d(v_j) - p(v_k, v_j))\right\} + e(v_j) \\ &= \tau.rbf(v_i, v_j, t - d(v_j) - \min_{v_k:(v_k,v_j)\in\mathbb{E}} p(v_k, v_j)) + e(v_j) \quad (20)\end{aligned}$$

Thus, $\tau.dbf(v_i, v_j, t)$ can be computed by simply shifting $\tau.rbf(v_i, v_j, t)$: to the right by $d(v_j) + \min_{v_k:(v_k,v_j)\in\mathbb{E}} p(v_k, v_j)$, and up by $e(v_j)$. This of course does not affect its periodicity. Thus $\tau.dbf(t)$ is almost linear periodic with the same linear factor and period as the corresponding *rbf*. □

Example 10 For the digraph task τ in Fig. 2, we have

$$\begin{aligned}\tau.dbf(v_1, v_1, t) &= \tau.rbf(v_1, v_1, t - d(v_1) - \min\left\{p(v_3, v_1), p(v_1, v_1)\right\}) + e(v_1) \\ &= \tau.rbf(v_1, v_1, t - 2) + 0.1\end{aligned}$$

From Eq. (9), $\forall t \geq 6, \tau.rbf(v_1, v_1, t - 1) = a^{(t)}_{1,1} = 0.1 \times t$. Thus for any $t \geq 7$, $\tau.dbf(v_1, v_1, t) = 0.1 \times t$. Similarly, we can derive the *dbf* functions for other pairs of source and sink vertices,

$$\forall t \geq 8,\ dbf(t) = \begin{bmatrix} 0 & -0.1 & 0 \\ 0.1 & 0 & 0.1 \\ 0 & -0.1 & 0 \end{bmatrix} + 0.1 \times t \quad (21)$$

and the *dbf* function is

$$\forall t \geq 8,\ \ \tau.dbf(t) = 0.1 + 0.1 \times t.$$

Although we proved the linear periodicity of the *rbf* and *dbf* functions for any digraph task, the result is of limited practical use when the execution request matrix is reducible (the digraph task is not strongly connected). In this case, to the best of our knowledge, there is no general procedure for computing the exact value (or an upper bound) of the general defect.

5 Computing the periodicity parameters for strongly connected digraphs

In this section, we provide details on efficient algorithms to compute the exact value or upper bounds for the periodicity parameters for *strongly connected task digraphs*. We expect most applications of practical interests to be represented by such graphs, especially for safety-critical systems, because of the need to bring back the system to a safe state. In such a digraph task $\mathcal{D}(\tau)$, the execution request matrix $\mathbf{A}$ of the

transformed UDRT $\mathcal{D}'(\tau')$ is irreducible (the digraph $\mathcal{G}(\mathbf{A})$ is strongly connected). The sequence of each element in $\mathbf{A}$ is almost linear periodic with period $p = lcm\{\text{hper}(\mathcal{K}) : \mathcal{K} \in HCC^*(\mathcal{G}(\mathbf{A}))\}$ and factor $q = \lambda(\mathbf{A})$ (Theorem 4). By Lemma 7, the sequence of the largest element is linear periodic with the same period and factor.

We are interested in efficient algorithms for computing the parameters p, q, and r. The unit graph UDRT $\mathcal{D}'(\tau')$, obtained by transformation from the original graph $\mathcal{D}(\tau)$, is used to demonstrate the existence of these parameters (for a strongly connected graph) and their relationship with the *rbf* and *dbf*. The computation of p, q, and r can be performed on the unit graph or the original graph, as discussed in the following sections. Since the transformed UDRT $\mathcal{D}'(\tau')$ can have a much larger size than the original graph $\mathcal{D}(\tau)$, in most cases, it is much simpler and more efficient when operating on $\mathcal{D}(\tau)$ (or a more compact transformation than $\mathcal{D}'(\tau')$).

5.1 Linear factor

The linear factor $q = \lambda(\mathbf{A})$ is the maximum cycle mean in $\mathcal{G}(\mathbf{A})$. The value of $q = \lambda(\mathbf{A})$ is the worst-case utilization (the asymptotic maximum value of the ratio execution request/demand rate) of the digraph task. This is the maximum cycle ratio (or cost to time ratio) in the original digraph Dasdan (2004). Dasdan (2004) conducted a comparative study on the six algorithms to compute the maximum cycle ratio, and reported that the one in Young et al. (1991) is practically the fastest. It is a parametric shortest path algorithm with complexity $O(|\mathbb{V}||\mathbb{E}|\log(|\mathbb{V}|))$, where $|\mathbb{V}|$ and $|\mathbb{E}|$ are the number of vertices and edges in the original digraph task, respectively. It operates on a finite sequence of shortest paths trees, and tries to create a new one by replacing the predecessor of a node v in the tree with another predecessor. The algorithm stops when a cycle is detected, and the ratio of the cycle is the maximum cost to time ratio. The pseudo-code of the algorithm is shown in Fig. 8 of Dasdan (2004).

Alternatively, because of the correspondence between the paths in the digraph task $\mathcal{D}(\tau)$ and its UDRT $\mathcal{D}'(\tau')$, q can be computed on $\mathcal{D}'(\tau')$ using the algorithm in Karp (1978), with complexity $O(|\mathbb{V}'||\mathbb{E}'|)$, where $|\mathbb{V}'|$ and $|\mathbb{E}'|$ are the number of vertices and edges in $\mathcal{D}'(\tau')$, respectively. The algorithm works on the digraph $\mathcal{G}(\mathbf{A})$ corresponding to the execution request matrix of $\mathcal{D}'(\tau')$. It is based on the Karp's theorem Karp (1978), which states that

$$q = \max_{v} \min_{0 \le k \le n-1} \frac{S_n(v) - S_k(v)}{n - k}$$

where $S_k(v)$ is the weight of the shortest path of length k from any source u to node v, and $n = |\mathbb{V}'|$ is the number of nodes in the graph. Starting from $k = 1$, it iteratively computes the values of $S_k(v)$ for $k = 1, \ldots, n-1$ as follows

$$S_k(v) = \max_{(u,v) \in \mathbb{E}'} \left\{ S_{k-1}(u) + e(u) \right\}$$

5.2 Linear period

The linear period p is computed using the algorithm in Theorem 3.6 of Gavalec (2000). The algorithm operates through five steps:

- **Step 1:** compute the maximum cycle mean $\lambda(\mathbf{A})$;
- **Step 2:** compute the metric matrix $\mathbf{W}$ (the matrix of all-pairs longest paths) of $\mathbf{A} - \lambda(\mathbf{A})$;
- **Step 3:** find the highly connected components of the digraph $\mathcal{G}(\mathbf{W})$;
- **Step 4:** delete from $\mathcal{G}(\mathbf{W})$ the edges that are not contained in any zero-cycle. Note that since $\mathbf{W}$ is obtained by removing $\lambda(\mathbf{A})$ from all edges of $\mathbf{A}$, the zero cycles are the cycles with maximum mean;
- **Step 5:** compute the non-trivial highly connected components in the digraph $\mathcal{G}(\mathbf{W})$ using Balcer-Veinott's condensation algorithm Balcer and Veinott (1973).

When operating on the UDRT $\mathcal{D}'(\tau')$, steps 3-5 are all of complexity $O(|\mathbb{V}'|^2)$. The matrix $\mathbf{A} - \lambda(\mathbf{A})$ has no positive cycles, hence Step 2 can be computed with the Floyd-Warshall algorithm Floyd (1962). The complexity of the algorithm is cubic $O(|\mathbb{V}'|^3)$ with respect to the number of nodes. Therefore, Step 2 has the highest complexity and determines the overall complexity of the procedure.

Step 2 can be improved by leveraging a special property of $\mathcal{D}'(\tau')$. In $\mathcal{D}'(\tau')$, node $v_{i,k}$ with $k \geq 2$ is derived from node v_i in $\mathcal{D}(\tau)$ with inter-arrival time greater than 1. Hence, every path to $v_{i,k}$ goes through $v_{i,1}$ and its successor nodes $v_{i,j}$ ($j = 2, \ldots, k$), the latter all have an associated zero WCET. Correspondingly, in the digraph of $\mathbf{A} - \lambda(\mathbf{A})$, the outgoing edge from the vertex $v_{i,j}$ ($j = 2, \ldots, k$) has weight $-\lambda(\mathbf{A})$. Thus, the weight $w(v_{i,2}, v_{m,1})$ of the longest path between two nodes $v_{i,2}$ and $v_{m,1}$ can be computed iteratively by adding two contributions where $v_{i,j}$ is any intermediate node with index $j = 2, \ldots, k$

- the weight of the path $w(v_{i,j}, v_{m,1})$,
- $(j-2)$ edges of weight $-\lambda(\mathbf{A})$ from $v_{i,2}$ to $v_{i,j}$.

Hence,

$$w(v_{i,2}, v_{m,1}) = w(v_{i,j}, v_{m,1}) - (j-2)\lambda(\mathbf{A}) \quad \forall j = 2, \ldots, k$$

or equivalently,

$$w(v_{i,j}, v_{m,1}) = w(v_{i,2}, v_{m,1}) + (j-2)\lambda(\mathbf{A}) \quad \forall j = 2, \ldots, k$$

Generalizing, the largest weight of any path between a pair of nodes $v_{i,k}$ and $v_{j,m}$ of the graph from $\mathcal{D}'(\tau')$ can be computed as:

$$w(v_{i,j}, v_{k,m}) = \begin{cases} w(v_{i,1}, v_{k,1}), & \text{if } j = 1 \text{ and } m = 1 \\ w(v_{i,1}, v_{k,2}) - (m-2)\lambda(\mathbf{A}), & \text{if } j = 1 \text{ and } m \geq 2 \\ w(v_{i,2}, v_{k,1}) + (j-2)\lambda(\mathbf{A}), & \text{if } j \geq 2 \text{ and } m = 1 \\ w(v_{i,2}, v_{k,2}) + (j-m)\lambda(\mathbf{A}), & \text{if } j \geq 2 \text{ and } m \geq 2 \end{cases} \tag{22}$$

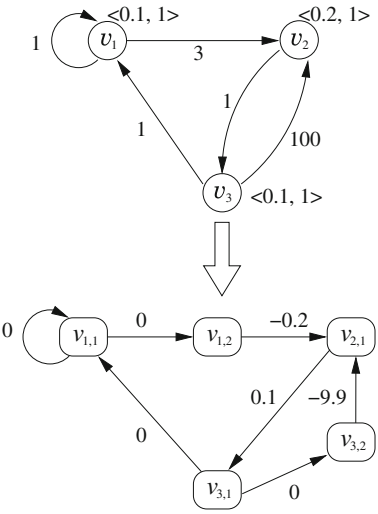

Fig. 7 The original task (*top*) and the transformed graph $\mathcal{D}^\circ$ (*bottom*)

Then, the problem reduces to computing the longest paths between the subset of nodes $v_{i,j}$ and $v_{k,m}$ in the graph $\mathcal{G}(\mathbf{A} - \lambda(\mathbf{A}))$, having values of j and m equal to either 1 or 2 and deriving all the other path weights by adding a proportional term as in (22).

Hence, instead of operating on the unit digraph $\mathcal{D}'$, it is more convenient to operate on a simplified digraph $\mathcal{D}^\circ$, obtained from $\mathcal{D}'$ by iteratively coalescing each node $v_{i,j}$ with a single outgoing edge with its successor $v_{i,j+1}$.

However, $\mathcal{D}^\circ$ can also be constructed directly from the original task $\mathcal{D}(\tau) = (\mathbb{V}, \mathbb{E})$ as follows:

- For each vertex $v_i \in \mathbb{V}$, if the maximum inter-arrival time among all of its outgoing edges $k_i \geq 2$ (as in (13)), then two vertices $v^\circ_{i,1}$ and $v^\circ_{i,2}$ are generated in $\mathcal{D}^\circ$. Otherwise, only one corresponding vertex $v^\circ_{i,1}$ is created.
- An edge $(v^\circ_{i,1}, v^\circ_{i,2})$ connects $v^\circ_{i,1}$ and $v^\circ_{i,2}$ with a weight equal to $e(v_i) - \lambda(\mathbf{A})$.
- For each edge $(v_i, v_j) \in \mathbb{E}$, if $p(v_i, v_j) = 1$, we add an edge $(v^\circ_{i,1}, v^\circ_{j,1})$ with a weight equal to $e(v_i) - \lambda(\mathbf{A})$; otherwise, an edge $(v^\circ_{i,2}, v^\circ_{j,1})$ is added with weight $(1 - p(v_i, v_j))\lambda(\mathbf{A})$.

Example 11 Figure 7 shows a task graph (top of the figure, with linear factor $\lambda(\mathbf{A}) = 0.1$) and its transformed digraph $\mathcal{D}^\circ$ for a compact representation of $\mathbf{A} - \lambda(\mathbf{A})$ (bottom of the figure). $\mathcal{D}^\circ$ only has 5 nodes and 7 edges, as opposed to 104 nodes and 106 edges in the corresponding UDRT.

As there is no positive cycle in $\mathcal{D}^\circ$, the Floyd–Warshall algorithm Floyd (1962) can be applied to calculate the longest paths between any nodes $v_{i,k}$ and $v_{j,m}$, where k and m are equal to either 1 or 2. This is done in $O(|\mathbb{V}|^3)$ time as the digraph $\mathcal{D}^\circ$ has $2 \times |\mathbb{V}|$ nodes. The other elements required in Step 2 can be derived using Eq. (22), with linear complexity.

5.3 Linear defect

Using the definition of the max-plus algebra, it is easy to verify that the maximum element of each matrix in the power series of $\mathbf{A}$ can be computed as $a^{(t)}_{\max} = \mathbf{0}^T \otimes \mathbf{A}^{(t)} \otimes \mathbf{0}$, where $\mathbf{0} = (0, 0, \ldots, 0)^T$ is the zero vector. The defect of such maximum element is clearly lower than or equal to the defect for the vector of elements obtained for the linear dynamical system $\mathbf{A}^* \otimes \mathbf{0}$ (Hartmann and Arguelles (1999)). Hence, the linear defect of $\mathbf{A}^* \otimes \mathbf{0}$ is an upper bound to the defect of $a^*_{\max}$.

The linear defect of a linear dynamical system can be computed with complexity $O(|\mathbb{V}'|^3(\log r + \log p))$ Hartmann and Arguelles (1999) (where r and p are its linear defect and period). This is done by iteratively computing the matrix power and finding the first r that satisfies $\mathbf{A}^{(r+p)} \otimes \mathbf{0} = \mathbf{A}^{(r)} \otimes \mathbf{0} + p \times q$. In addition, upper bounds on r can be calculated more efficiently (Hartmann and Arguelles (1999) Charron-Bost et al. (2011)) with complexity $O(|\mathbb{V}'|^2)$ on top of the algorithms to compute p and q.

The alternative is to directly work on the original digraph task. The linear dynamical system $\mathbf{A}^{(t)} \otimes \mathbf{0} = (a^{(t)}_j = \max_i\{rbf(v_i, v_j, t-1)\})$ is the vector of the *rbf* functions ending in v_j. Such *rbf* functions can be computed using a similar algorithm as proposed in Stigge et al. (2011) (Fig. 5 in the paper), with complexity $O((|\mathbb{V}| + |\mathbb{E}|) \cdot t)$. Starting from $t = 1$, the algorithm iteratively update the *rbf* function $a^{(t)}_j$ with dynamic programming techniques.

5.4 Computing p, q, and r on the UDRT versus the original graph

The algorithms presented in the previous section avoid the need to construct and operate on the possibly large UDRT graph, and allow the linear periodicity parameters on the original graph to be computed (typically more efficiently). In our experiments, we use them for the comparison with the state-of-the-art analysis Stigge et al. (2011).

However, in selected cases, the algorithms that operate on the UDRT may be faster. In particular, the complexity of computing the linear factor on the transformed UDRT is $O(|\mathbb{V}'||\mathbb{E}'|)$, where $|\mathbb{V}'|$ and $|\mathbb{E}'|$ are the number of vertices and edges respectively. This is not necessarily higher than the complexity $O(|\mathbb{V}||\mathbb{E}|\log(|\mathbb{V}|))$ of the algorithm that operates on the original graph (as discussed in Sect. 5.1). We leave for future work the study on the comparison and optimal selection of these algorithms.

6 An improved upper bound on t_f

In this section, we compute tight linear upper bounds on the *rbf* and *dbf* functions, which are used to derive an upper bound for t_f, the time limit for checking the feasibility conditions in (2) and (3). To do so, we use the previously developed task transformation and the power sequence on its execution request matrix.

For the digraph task τ, the graph of its execution request matrix is $\mathcal{G}(\mathbf{A}) = (\mathbb{V}, \mathbb{E}, w)$. We consider $\mathcal{G}'(\mathbf{A}) = \mathcal{G}(\mathbf{A}) - \lambda(\mathbf{A})$, i.e., $\mathcal{G}'(\mathbf{A}) = (\mathbb{V}, \mathbb{E}, w')$ where $w'(e) = w(e) - \lambda(\mathbf{A})$ for all $e \in \mathbb{E}$. $\mathcal{G}'(\mathbf{A})$ shares the same topology as $\mathcal{G}(\mathbf{A})$, thus for any path Π in $\mathcal{G}(\mathbf{A})$, there exists the same path in $\mathcal{G}'(\mathbf{A})$, with weight

$$w'(\Pi) = w(\Pi) - |\Pi| \cdot \lambda(\mathbf{A}) \tag{23}$$

Hence, the maximum cycle mean of $\mathcal{G}'(\mathbf{A})$ is zero, and the longest path in $\mathcal{G}'(\mathbf{A})$ is well defined.

We now derive a linear upper bound on the *rbf* function.

Theorem 14 *The rbf and dbf functions of a task digraph with matrix* $\mathbf{A}$ *are upper bounded by linear functions of time with proportionality coefficient* $\lambda(\mathbf{A})$ *and constant terms* C^{rbf} *and* C^{dbf}.

$$\forall t \geq 0, \begin{cases} \tau.rbf(t) \leq C^{rbf} + t \cdot \lambda(\mathbf{A}) \\ \tau.dbf(t) \leq C^{dbf} + t \cdot \lambda(\mathbf{A}) \end{cases} \tag{24}$$

where the constant terms are

$$C^{rbf} = \max\left\{w'(\Pi) | \Pi \textit{ is a path in } \mathcal{G}'(\mathbf{A})\right\} + \lambda(\mathbf{A})$$
$$C^{dbf} = C^{rbf} + \min\{\max_{(v_k, v_j) \in \mathcal{D}(\tau)} \left\{e(v_j) - (d(v_j) + p(v_k, v_j)) \cdot \lambda(\mathbf{A})\right\}, -\lambda(\mathbf{A}) \cdot d_{\min}\}$$

and $d_{\min}$ *is the smallest deadline of any node in the graph,* $d_{\min} = \min_{v_j \in \mathcal{D}(\tau)} d(v_j)$.

Proof Combining Eqs. (18) and (11), and assuming the unit graph is used for the purpose of this demonstration, we have

$$\begin{aligned} & \tau.rbf(t) \\ &= \max\{w(\Pi) | \Pi \text{ is a path in } \mathcal{G}(\mathbf{A}), |\Pi| = t + 1\} \\ &= \max\{w'(\Pi) + (t+1)\lambda(\mathbf{A}) | \Pi \text{ is a path in } \mathcal{G}'(\mathbf{A}), |\Pi| = t + 1\} \qquad \text{– by (23)} \\ &\leq \max\{w'(\Pi) + (t+1)\lambda(\mathbf{A}) | \Pi \text{ is a path in } \mathcal{G}'(\mathbf{A})\} \\ &= C^{rbf} + t \cdot \lambda(\mathbf{A}) \end{aligned}$$

The bound on *dbf* is derived by combining the bound on *rbf* with (20) and the previously demonstrated inequality $dbf(t) \leq rbf(t - d_{\min})$. □

C^{rbf} can be computed in $O(|\mathbb{V}|^3)$ time ($\mathbb{V}$ is the number of nodes in $\mathcal{G}'(\mathbf{A})$) using the Floyd–Warshall algorithm Floyd (1962).

We now prove that these linear bounds improve on the existing bounds available from the literature.

Theorem 15 *The linear bound (24) is* **tighter than the corresponding bound provided in** *Stigge et al. (2011)* **for the** *dbf:* $\tau.dbf(t) \leq S + t \cdot \lambda(\mathbf{A})$ *where* $S = \sum_{v_i \in \mathcal{D}(\tau)} e(v_i)$ *is the sum of the WCETs of the vertexes in* $\mathcal{D}(\tau)$.

Proof The two linear bounds have the same proportionality coefficient $\lambda(\mathbf{A})$. Hence, it is sufficient to show that $C^{dbf} \leq S$.

Since there is no positive cycle in $\mathcal{G}'(\mathbf{A})$, there must exist a longest path Π in $\mathcal{G}'(\mathbf{A})$ which is also an elementary path. Thus, W is no greater than the maximum sum of

the positive edge weights in the elementary path Π (where each node v_i appears at most once). Also, the task transformation guarantees that the weight $w(g)$ of an edge $g = (v_i, v_j)$ in $\mathcal{G}(\mathbf{A})$ equals either zero or the WCET $e(v_i)$ of the source node v_i in $\mathcal{D}(\tau)$.

$$\begin{aligned} C^{rbf} &\leq \sum_{g=(i,j)\in\Pi} \max\{w'(g), 0\} \\ &\leq \sum_{v_k\in\mathcal{D}(\tau)} \max\{e(v_k) - \lambda(\mathbf{A}), 0\} \\ &\leq \sum_{v_k\in\mathcal{D}(\tau)} e(v_k) - \lambda(\mathbf{A}) \\ &= S - \lambda(\mathbf{A}) \end{aligned}$$

This implies that $C^{dbf} \leq C^{rbf} + \lambda(\mathbf{A}) = S$. □

Example 12 For the example digraph task in Fig. 2, the linear bound on the *rbf* is $0.2+0.1\times t$. For *dbf*, it is $0.1+0.1\times t$. As a comparison, the bound on *dbf* from Stigge et al. (2011) is $0.4 + 0.1 \times t$.

The linear bounds on *rbf* and *dbf* in (24) can be used to derive the maximum length t_f of the time interval to be checked, where the utilization of task τ_i is the maximum cycle mean $\lambda_i(\mathbf{A})$ of its digraph.

Under the assumption that the total utilization is less than 1, i.e., $\sum_{\tau_j\in\Gamma} \lambda_j(\mathbf{A}) < 1$, a task $\tau_i \in \Gamma$, scheduled with static priority, is not schedulable if (countering the hypothesis of Theorem 2)

$$\begin{aligned} &\exists t_0, \forall t' \leq t_0,\ \tau_i.dbf(t_0) + \sum_{\tau_j\in hp(i)} \tau_j.rbf(t') > t' \\ \Longrightarrow\ &\exists t_0,\ \tau_i.dbf(t_0) + \sum_{\tau_j\in hp(i)} \tau_j.rbf(t_0) > t_0 \\ \Longrightarrow\ &C_i^{dbf} + t_0\cdot\lambda_i(\mathbf{A}) + \sum_{\tau_j\in hp(i)} \left(C_j^{rbf} + \lambda_j(\mathbf{A}\right) + t_0\cdot\lambda_j(\mathbf{A})) > t_0 \\ \Longrightarrow\ &t_0 < \frac{C_i^{dbf} + \sum_{\tau_j\in hp(i)}(C_j^{rbf} + \lambda_j(\mathbf{A}))}{1 - \lambda_i(\mathbf{A}) - \sum_{\tau_j\in hp(i)} \lambda_j(\mathbf{A})} \end{aligned} \tag{25}$$

Thus, $t_f = \dfrac{C_i^{dbf} + \sum_{\tau_j\in hp(i)}(C_j^{rbf} + \lambda_j(\mathbf{A}))}{1 - \lambda_i(\mathbf{A}) - \sum_{\tau_j\in hp(i)} \lambda_j(\mathbf{A})}$ is a safe upper bound on the set of time instants to check the schedulability condition in Eq. (3).

For systems with dynamic priority (EDF), it is

$$\begin{aligned} &\exists t_0,\ \sum_{\tau_i \in \Gamma} \tau_i.dbf(t_0) > t_0 \\ \Longrightarrow\ &\exists t_0,\ \sum_{\tau_i \in \Gamma} \left(C_i^{dbf} + t_0 \cdot \lambda_i(\mathbf{A}) \right) > t_0 \\ \Longrightarrow\ &\exists t_0,\ t_0 < \frac{\sum_{\tau_i \in \Gamma} C_i^{dbf}}{1 - \sum_{\tau_i \in \Gamma} \lambda_i(\mathbf{A})} \end{aligned} \tag{26}$$

Hence, if (2) holds for all the time points $t < \frac{\sum_{\tau_i \in \Gamma} C_i^{dbf}}{1 - \sum_{\tau_i \in \Gamma} \lambda_i(\mathbf{A})}$, then the system is EDF schedulable.

7 Arbitrary deadlines

In this section, we prove the periodicity of the *rbf* and *dbf* functions even when the **l-MAD** assumption is not valid, i.e., deadlines are arbitrary but bounded. Therefore, it is possible that in an event sequence, some job other than the last one has the latest deadline, and in general, Eq. (20) does not hold. This change does not affect the periodicity of the *rbf* function, as it does not depend on the deadline of the jobs. However, the linear relation between *rbf* and *dbf*, defined in Eq. (20) (and exploited to demonstrate the periodicity of the *dbf*), needs to be redefined considering a larger set of possible ending nodes.

First, we compute an upper bound on the distance between the starting and ending nodes in a path where the starting node may have a deadline later than that of the ending node.

$$l_{\max} = \max \left\{ l : (v_1, \ldots, v_l) \in \Pi_{\mathcal{D}(\tau)}, d(v_1) > \sum_{i=1}^{l-1} p(v_i, v_{i+1}) + d(v_l) \right\}$$

Next, the definition of *rbf* is extended to include a set of possible $\max\{1, l_{\max}\}$ ending nodes in the path.

Definition 15 Given a set of $k = l_{\max} + 1$ vertices $u_{i_1}, u_{i_2}, \ldots, u_{i_k}$, the *request bound function* of a digraph task τ during any time interval of length t, denoted as $\tau.rbf(u_{i_1}, u_{i_2}, \ldots, u_{i_k}, t)$, is defined as the maximum sum of execution times of any legal event sequence $[(t_m, u_m), m = 1, \ldots, l]$ of τ such that

- the vertex corresponding to the first event is $u_1 = u_{i_1}$;
- the vertexes corresponding to the $k-1$ possible last events are $u_l = u_{i_k}$, $u_{l-1} = u_{i_{k-1}}, \ldots, u_{l-k+2} = u_{i_2}$;
- $\forall j = 1, \ldots, l-1, t_j - t_1 \leq t$;
- $\tau.rbf(u_{i_1}, u_{i_2}, \ldots, u_{i_k}, t) = \max \sum_{j=1}^{l-1} e(u_j)$.

As in the case of l-MAD, this definition of *rbf* does not include any reference to the release time or the execution time of the last job.

In the l-MAD case, $\mathbf{A}^{(t)}$ is a two-dimensional array, where the indices i and j of the elements $a_{i,j}^{(t)}$ define the starting and ending nodes in the path of $\mathcal{G}(\mathbf{A})$. For multiple ending nodes, the previous definition needs to be extended to a multi-dimensional array.

Definition 16 For any t and $k \geq 2$, and for any $i_1, i_2, \ldots, i_k \in \{1, \ldots, n\}$, we denote by $\Pi^t_{\mathcal{G}(\mathbf{A})}(i_1, i_2, \ldots, i_k)$ the set of all paths in $\mathcal{G}(\mathbf{A})$, of length t, where the first node in the path is i_1, and the last $k-1$ nodes are $i_2, \ldots, i_k$.

Definition 17 For a square matrix $\mathbf{A} \in \mathbb{R}^*(n, n)$ and its corresponding graph $\mathcal{G}(\mathbf{A})$, the power sequence of the k-dimensional array for any integer $k \geq 2$, denoted as $\mathbf{W}^* = (w^{(t)}_{i_1,i_2,\ldots,i_k})$, $\forall t \in \mathbb{N}^+$, is defined as the maximum weight of all the paths in $\Pi^t_{\mathcal{G}(\mathbf{A})}(i_1, i_2, \ldots, i_k)$. The set of paths and elementary paths can be defined similarly. We denote them as $\mathbb{P}_{\mathcal{G}(\mathbf{A})}(i_1, i_2, \ldots, i_k)$ and $\mathbb{P}^*_{\mathcal{G}(\mathbf{A})}(i_1, i_2, \ldots, i_k)$ respectively.

Lemma 6 can be extended to multi-dimensional arrays, based on the same reasoning in Molnárová (2005) (Lemma 5.1 in the paper, for the case of matrices).

Lemma 16 *For an elementary path $\Pi \in \mathbb{P}^*_{\mathcal{G}(\mathbf{A})}(i_1, i_2, \ldots, i_k)$, if its maximum cycle mean $\lambda(\Pi) > -\infty$, then a^*_Π is almost linear periodic with a linear factor of $\lambda(\Pi)$; otherwise, it is almost generally periodic with a general factor of $-\infty$.*

The following lemma extends Theorem 10 to a multi-dimensional array for the linear periodicity of its maximum element.

Lemma 17 *The sequence of the maximum element of the k-dimensional array for any non-trivial square matrix is almost linear periodic.*

The proof is similar to the one in Theorem 10, obtained by constructing a subset of the elementary paths that dominate other paths, and have the same linear factor.

The definition of *dbf* function is also extended to a starting vertex and a set of ending vertices.

Definition 18 Given a set of k vertices $u_{i_1}, u_{i_2}, \ldots, u_{i_k}$, the *demand bound function* of a digraph task τ during the time interval t, denoted as $\tau.dbf(u_{i_1}, u_{i_2}, \ldots, u_{i_k}, t)$, is defined as the maximum sum of execution times from any legal event sequence $[(t_m, u_m), m = 1, \ldots, l]$ of τ such that

- the vertex corresponding to the first event is $u_1 = u_{i_1}$;
- the vertexes corresponding to the $k-1$ possible last events are $u_l = u_{i_k}$, $u_{l-1} = u_{i_{k-1}}, \ldots, u_{l-k+2} = u_{i_2}$;
- $\forall j = 1, \ldots, l, d_j + t_j - t_1 \leq t$;
- $\tau.dbf(u_{i_1}, u_{i_2}, \ldots, u_{i_k}, t) = \max \sum_{j=1}^{l} e(u_j)$.

Given a set of k vertices $u_{i_1}, u_{i_2}, \ldots, u_{i_k}$, we denote the set of possible absolute deadlines for the ending vertices as (assuming the release time of u_{i_2} is zero)

$$D = \left\{ d(u_{i_j}) + \sum_{m=2}^{j-1} p(u_{i_m}, u_{i_{m+1}}) \mid j = 2, \ldots, k \right\}$$

For each $d \in D$, we denote as $S(d)$ the set of vertices in the path with an absolute deadline larger than d:

$$S(d) = \left\{ i_j \mid d(u_{i_j}) + \sum_{m=2}^{j-1} p(u_{i_m}, u_{i_{m+1}}) > d \right\}$$

With these notations and extensions, we derive the relationship between the *rbf* and *dbf* functions. We note that the inter-release time between u_{i_2} and $u_{i_{k-1}}$ (the last node included in $\tau.rbf(u_{i_1}, u_{i_2}, \ldots, u_{i_k}, t)$) is $\sum_{m=2}^{j-2} p(u_{i_m}, u_{i_{m+1}})$. To compute $\tau.dbf(u_{i_1}, u_{i_2}, \ldots, u_{i_k}, t)$, the latest deadline may be one of those $d \in D$, and the corresponding release time for $u_{i_{k-1}}$ is $t + \sum_{m=2}^{j-2} p(u_{i_m}, u_{i_{m+1}}) - d$. Hence,

$$\begin{aligned} &\tau.dbf(u_{i_1}, u_{i_2}, \ldots, u_{i_k}, t) \\ &= \max_{d \in D} \left\{ \tau.rbf\left(u_{i_1}, u_{i_2}, \ldots, u_{i_k}, t + \sum_{m=2}^{j-2} p(u_{i_m}, u_{i_{m+1}}) - d\right) - \sum_{j \in S(d)} e(u_j) + e(u_{i_k}) \right\} \end{aligned} \tag{27}$$

We now prove the linear periodicity of the *dbf* function for digraph tasks with arbitrary deadlines.

Theorem 18 *The dbf function for a digraph task τ with arbitrary deadlines is almost linear periodic.*

Proof As in Eq. (27), $\tau.dbf(u_{i_1}, u_{i_2}, \ldots, u_{i_k}, t)$ can be computed as the maximum of a finite number of terms, each corresponding to a transformation (by moving along the two axes) of a periodic *dbf* function $\tau.rbf(u_{i_1}, u_{i_2}, \ldots, u_{i_k}, t)$. By Lemma 17, $\tau.dbf(t)$ is almost linear periodic. □

Example 13 For the digraph task in Fig. 2, suppose the deadline for vertex v_2 is now 3, thus $l_{\max} = 2$ and $k = 3$. Therefore, it is necessary to consider two ending vertices and a three-dimensional array. We consider $\tau.dbf(v_1, v_2, v_3, t)$ as an example. In this case, $D = \{3, 1 + 1\} = \{3, 2\}$, and $S(2) = \{v_2\}$, $S(3) = \emptyset$. By Eq. (27),

$$
\begin{aligned}
&\tau.dbf(v_1, v_2, v_3, t) \\
&= \max\left\{\tau.rbf(v_1, v_2, v_3, t-3) + e(v_3), \tau.rbf(v_1, v_2, v_3, t-2) - e(v_2) + e(v_3)\right\} \\
&= \max\left\{\tau.rbf(v_1, v_2, v_3, t-3) + 0.1, \tau.rbf(v_1, v_2, v_3, t-2) - 0.1\right\}
\end{aligned}
$$

8 Experimental results

In this section, we evaluate the improvements on the efficiency of schedulability analysis compared to the state-of-the-art Stigge et al. (2011). We generate applications consisting of random sets with $n = 5$ to 40 tasks. Each task is a random graph with 1–15 nodes. For each n, 1,000 sets are generated and then examined for schedulability. The number of outgoing edges from a node is randomly distributed as: 40 % of the node has one outgoing edge, 4 % with two, 10 % with three, and 10 % with four. These outgoing edges are randomly connected to any other node (including the source itself). The average (in- and out-) degree of the nodes is 3.8, and 77.4 % of the task graphs are strongly connected. The base period of each task is generated by the product of one to three factors, each randomly drawn from the harmonic sets (2, 4), (6, 12), (5, 10). The inter-release time is scaled by a factor randomly extracted from the set {1, 2, 4, 5, 10}. The deadline of a node is the minimum inter-release time among those of its out-going edges, thereby enforcing the frame separation property. The execution times of the task are selected such that its utilization is uniformly distributed. For comparison, we assume the system is scheduled using EDF, the only policy considered in the previous analysis Stigge et al. (2011).

The schedulability analysis is performed in three sequential steps:

1. calculation of the linear periodicity parameters, which is only necessary for our algorithm;
2. calculation of the *dbf* function within t_f;
3. check of schedulability conditions in (2) for $t < t_f$ in increasing order. If at any point t, (2) is violated, then the system is deemed as unschedulable. Otherwise, it is schedulable.

We compare the performance of our algorithms, applied to the original digraph tasks, with the analysis in Stigge et al. (2011). Figure 8 shows the improvement for systems with 20 tasks on t_f, the runtime for calculating the *dbf* function, and the total runtime. It also plots the percentage of strongly connected digraph tasks with $r < t_f$ where t_f is the improved bound computed in Eq. (26), i.e., the tasks for which we explore the linear periodicity to calculate the *dbf* function.

As shown in Fig. 8, the computed t_f is 3 times smaller for all utilizations (50–99 %). The speedup on the total runtime and the portion for the calculation of *dbf* function is a factor of around 1.8 at 60 % utilization and then increases even further, leveraging the improvements in the computed t_f value. For a very high (99 %) utilization, nearly 97 % of the strongly connected digraph tasks have a linear defect smaller than t_f.

Figure 9 plots the improvement on the total runtime versus n, the number of tasks in the system. The speedup is almost independent from n, since t_f is uniformly reduced about 3 times for all tasks, and the linear periodicity of *rbf* and *dbf* only depends on the graph structure.

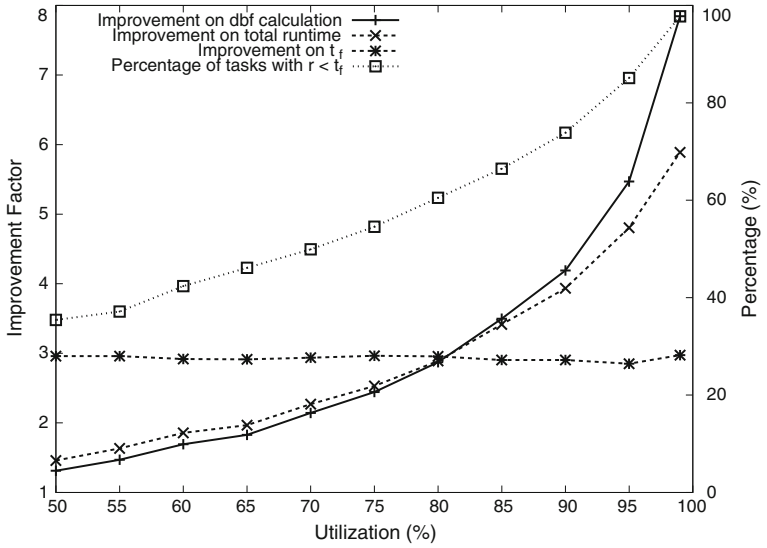

Fig. 8 The improvement compared to the analysis in Stigge et al. (2011) and the percentage of tasks with $r < t_f$ (number of tasks $n = 20$)

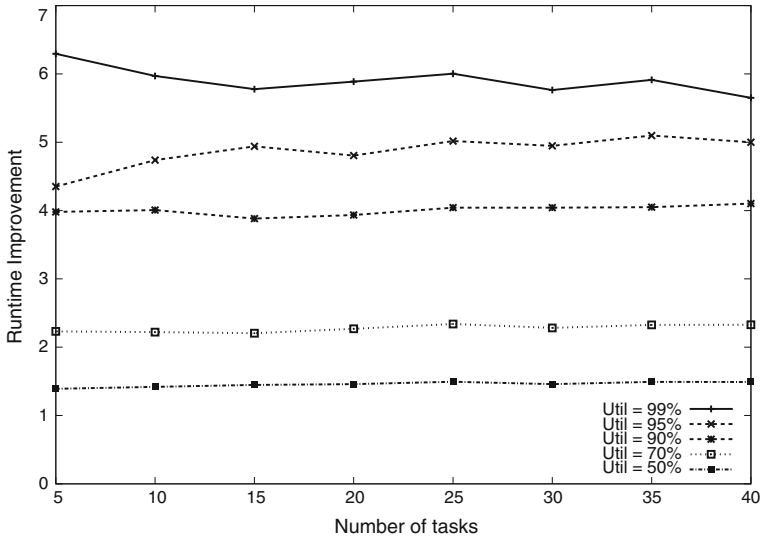

Fig. 9 The runtime improvement versus the number of tasks in the system

9 Conclusion

In this paper, for the first time, we demonstrate the linear periodicity of non-cyclic task models. We leverage results from max-plus algebra, extend the concept of periodicity of execution requests to task models without cyclic recurrent behavior, including the digraph task model and its extension. This essentially makes the asymptotic complexity

of calculating *rbf* and *dbf* functions independent from the time interval t. We provide polynomial-time algorithms for the computation of the linear periodicity parameters for strongly connected graphs. Experimental results demonstrate that such a property can be used to improve the efficiency of schedulability analysis for real-time systems captured by these task models.

The significance of our paper are multifold. It introduces max-plus algebra, a potentially useful analysis tool for real-time systems. Our contribution can possibly lead to more efficient analysis for other task models. It also opens the possibility of using non-recurrent task models to represent application structures within the SymTA/S Henia et al. (2005) and MPA Wandeler and Thiele (2006) schedulability analysis frameworks. Besides providing a comparison of the algorithms to compute the linear periodicity parameters, the future work also includes extending our results to other task models with parallelism.

References

Anand M, Easwaran A, Fischmeister S, Lee I (2008) Compositional feasibility analysis of conditional real-time task models. In: Proceedings of the 11th IEEE international symposium on object oriented real-time distributed computing, pp 391–398, May 2008.

Axer P, Quinton S, Neukirchner M, Ernst R, Döbel B, Härtig H (2013) Response-time analysis of parallel fork-join workloads with real-time constraints. In: Proceedings of the 25th euromicro workshop on real-time systems, pp 215–224, July 2013.

Baccelli F, Cohen G, Olsder G, Quadrat J (1992) Synchronization and linearity: an algebra for discrete event systems. Wiley, New York

Baruah S (1998) Feasibility analysis of recurring branching tasks. In: Proceedings of the 10th Euromicro Workshop on Real-Time Systems, pp 138–145, June 1998.

Baruah S, Chen D, Gorinsky S, Mok A (1999) Generalized multiframe tasks. Real-Time Syst 17(1):5–22

Baruah S (2003) Dynamic- and static-priority scheduling of recurring real-time tasks. Real-Time Syst 24(1):93–128, January 2003.

Baruah S (2010) The non-cyclic recurring real-time task model. In Proceedings of the 31st IEEE Real-Time Systems Symposium, pp 173–182, December 2010.

Baruah S, Bonifaci V, Marchetti-Spaccamela A, Stougie L, Wiese A (2012) A generalized parallel task model for recurrent real-time processes. In Proceedings of the 33rd IEEE real-time systems symposium, pp 63–72, December 2012.

Balcer Y, Veinott AF (1973) Computing a graph's period quadratically by node condensation. Discret Math 4(4):295–303

Bonifaci V, Marchetti-Spaccamela A, Stougie L, Wiese A (2013) Feasibility analysis in the sporadic DAG task model. In Proceedings of the 25th euromicro conference on real-time systems, pp 225–233, July 2013.

Charron-Bost B, Függer M, Nowak T (2011) On the transience of linear max-plus dynamical systems. Comput Res Repos (CoRR). arXiv:1111.4600 2011.

Dasdan A (2004) Experimental analysis of the fastest optimum cycle ratio and mean algorithms. ACM Trans Design Autom Electron Syst 9(4):385–418, October 2004.

Fersman E, Krcal P, Pettersson P, Yi W (August 2007) Task automata: schedulability, decidability and undecidability. Inform Comput 205(8):1149–1172

Floyd R (1962) Algorithm 97: shortest path. Commun ACM 5(6):345, June 1962.

Gavalec M (2000) Linear matrix period in max-plus algebra. Linear Algebra Appl 307(1–3):167–182, March 2000.

Gavalec M (2000b) Polynomial algorithm for linear matrix period in max-plus algebra. Central Eur J Oper Res 8(3):247–258, 2000.

Hartmann M, Arguelles C (May 1999) Transience bounds for long walks. Math Oper Res 24(2):414–439

Henia R, Hamann A, Jersak M, Racu R, Richter K, Ernst R (March 2005) System level performance analysis—the SymTA/S approach. IEE Proc Comput Digital Techn 152(2):148–166

Karp R (1978) A characterization of the minimum cycle mean in a digraph. Discret Math 23(3):309–311, 1978.
Künzli S, Hamann A, Ernst R, Thiele L (2007) Combined approach to system level performance analysis of embedded systems. In: Proceedings of the IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis, pp 63–68, September 2007.
Liu CL, Layland JW (January 1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. J ACM 20(1):46–61
Norström C, Wall A, Yi W (1999) Timed automata as task models for event-driven systems. In: Proceedings of the 6th conference on real-time computing systems and applications, pp 182–189, 1999.
Mok AK, Chen D (1996) A multiframe model for real-time tasks. In Proceedings of the 17th IEEE real-time systems symposium, pp 22–29, December 1996.
Molnárová M (2003) Computational complexity of nachtigall's representation. Optimization 52(1):93–104, 2003.
Molnárová M (2005) Generalized matrix period in max-plus algebra. Linear Algebra Appl 404:345–366, July 2005.
Moyo NT, Nicollet E, Lafaye F, Moy C (2010) On schedulability analysis of non-cyclic generalized multiframe tasks. In: Proceedings of the 22nd euromicro conference on real-time systems, pp 271–278, July 2010.
Saifullah A, Agrawal K, Lu C, Gill C (2011) Multi-core real-time scheduling for generalized parallel task models. In: Proceedings of the 32nd IEEE real-time systems symposium, pp 217–226, December 2011.
Stigge M, Ekberg P, Guan N, Yi W (2011) The digraph real-time task model. In: Proceedings of the 16th IEEE real-time and embedded technology and applications symposium, pp 71–80, April 2011.
Stigge M, Ekberg P, Guan N, Yi W (2011b) On the tractability of digraph-based task models. In: Proceedings of the 23rd euromicro conference on real-time systems, pp 162–171, July 2011.
Stigge M, Yi W (2012) Hardness results for static priority real-time scheduling. In: Proceedings of the 24th euromicro conference on real-time systems, pp 189–198, July 2012.
Stigge M, Yi W (2013) Combinatorial abstraction refinement for feasibility analysis. In Proceedings of the 34th IEEE real-time systems symposium, pp 340–349, December 2013.
Tarjan, R E (1972) Depth-first search and linear graph algorithms. SIAM J Comput 1(2): 146–160, 1972.
Thiele L, Chakraborty S, Naedele M (2000) Real-time calculus for scheduling hard real-time systems. In Proceedings of the IEEE international symposium on circuits and systems, pp 101–104, 2000.
Wandeler E, Thiele L (2006) Real-time calculus (RTC) toolbox. http://www.mpa.ethz.ch/Rtctoolbox. Accessed 17 March 2014
Young N, Tarjant R, Orlin J (March 1991) Faster parametric shortest path and minimumbalance algorithms. Networks 21(2):205–221
Zeng H, Di Natale M (2012) Schedulability analysis of periodic tasks implementing synchronous finite state machines. In Proceedings of the 24th euromicro conference on real-time systems, pp 353–362, July 2012.
Zuhily A, Burns A (October 2009) Exact scheduling analysis of non-accumulatively monotonic multiframe tasks. Real-Time Syst 43(2):119–146