On the Tractability of Digraph-Based Task Models

Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi
Uppsala University, Sweden
Email: {martin.stigge | pontus.ekberg | nan.guan | yi}@it.uu.se

***Abstract*—In formal analysis of real-time systems, a major concern is the analysis efficiency. As the expressiveness of models grows, so grows the complexity of their analysis. A recently proposed model, the digraph real-time task model (DRT), offers high expressiveness well beyond traditional periodic task models. Still, the associated feasibility problem on preemptive uniprocessors remains tractable. It is an open question to what extent the expressiveness of the model can be further increased before the feasibility problem becomes intractable.**

In this paper, we study that tractability border. We show that system models with the need for global timing constraints make feasibility analysis intractable. However, our second technical result shows that it remains tractable if the number of global constraints is bounded by a constant. Thus, this paper establishes a precise borderline between tractability and intractability.

I. Introduction

Abstract models used in the analysis of real-time systems aim at representing the system under analysis as precisely as possible for the analysis objective at hand. However, more precise and expressive models tend to increase the computational complexity of the employed methods. Thus, the model choice imposes a critical efficiency trade-off.

A key property one would like to check efficiently is the system's *feasibility*. It states whether one can find a preemptive uniprocessor schedule for all jobs created by the whole system, such that all jobs can meet their associated deadline. For its analysis, a common approach is the decomposition of the system into processes or tasks which are modeled separately and more or less independent. The classical task model is the well-known *Liu and Layland task model* [1], which expresses every task independently as a periodic sequence of recurring and equal jobs. The benefit of this basic model is that many system properties such as feasibility can be checked very efficiently. However, since its expressiveness is rather limited, researchers have proposed increasingly expressive task models. The most general model to date with a tractable feasibility problem is the *Digraph Real-Time task model (DRT)* [2]. For each task, it uses an arbitrary directed graph for modeling the release structure of different types of jobs.

An interesting challenge is to determine how close to intractability the feasibility problem for DRT already is. How far can the research community expect to explore further generalizations before exact feasibility analysis can not be solved efficiently anymore?

In this paper, we study the tractability borderline. We add an extension to DRT which allows to express *global inter-release separation constraints* between non-adjacent job releases. These additional constraints allow modeling of general periods for sub-structures like periodic modes, limited burstiness, and related paradigms. Since the model extension only affects certain release patterns of the modeled jobs, one could expect feasibility analysis to grow only moderately in complexity. However, we show that without further restrictions, this extension leads to intractability. Still, for a bounded version of the extension, we present an efficient analysis.

In particular, based on a precise notion of global inter-release separation constraints, this paper provides the following contributions:

- We show that with a bounded number of constraints, the feasibility problem in the DRT task model is *tractable*, i.e., can be decided in pseudo-polynomial time for systems with bounded utilization.
- We prove that without a bound on the constraint number, feasibility becomes strongly coNP-hard, i.e., *intractable*, even in the restricted setting of constrained deadlines.

For the tractable case, we further suggest several optimizations that are critical to an efficient implementation. Our prototype based on these optimizations is able to analyze randomly generated high-utilization task sets of 50 tasks, each with 20 job types and 2 global constraints, within a few minutes.

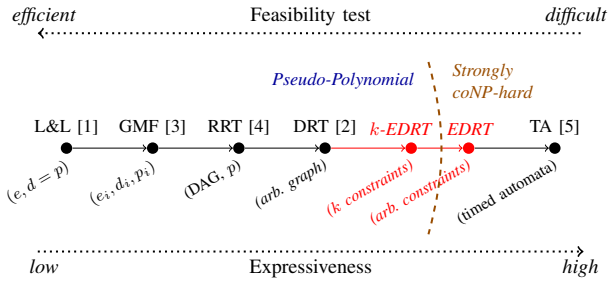


Fig. 1. A hierarchy of task models. Arrows indicate the generalization relationship. The higher the expressiveness, the more expensive the feasibility test. The tractability borderline is identified by our two model variants.

A. Prior Work

Figure 1 gives an overview of the expressiveness of existing task models in relation to the complexity of their feasibility problem. The classical *periodic task model* by Liu and Layland [1] models each task as a periodically released job with an implicit deadline equal to the period. An important step in its generalization is the *generalized multiframe* (GMF) model [3]. It allows deadlines to differ from periods, *sporadic* job releases, and, most importantly, a task may contain different job types ("frames"). The assumed behavior is that the task cycles through the different job types. Another significant generalization is the *Recurring Real-Time*

task model (RRT) [4]. It allows modeling of *branches* in the job release structure by representing each task as a directed acyclic graph (DAG). However, after traversing the DAG, each task needs to start again from its initial vertex while complying with a given task period, making all tasks *recurrent*. In a recent result [2], the job release structure has been generalized to arbitrary directed graphs in the *Digraph Real-Time task model (DRT)*. Here, the structure of job releases is not predetermined in any way and can be modeled without any restrictions to the graph topology. Further, deadlines can be arbitrary, i.e., larger than the delay until the next job release. However, despite the strong expressiveness, it has been shown that feasibility can still be decided in pseudo-polynomial time for all the above models.

Another step in expressiveness is the *task automata* model [5]. It allows to express complex dependencies between job releases and task synchronization, at the cost of a very expensive schedulability test.

An example for related hardness results for scheduling problems is a recent result [7] showing that feasibility is already *weakly* coNP-hard for synchronous periodic task models. We however focus on pseudo-polynomial complexity as the consensus for the notion of *tractability* in the real-time community.

II. Task Model

An extended digraph real-time (EDRT) task system $\tau = \{T_1, \ldots, T_N\}$ consists of N independent tasks. A task T is represented by a graph $G(T)$ with both vertex and edge labels, and a constraint set $C(T)$. The vertices $\{v_1, \ldots, v_n\}$ of $G(T)$ represent the types of all the jobs that T can release. Each vertex v_i is labeled with an ordered pair $\langle e(v_i), d(v_i)\rangle$ denoting worst-case execution-time demand $e(v_i)$ and relative deadline $d(v_i)$ of the corresponding job. Both values are assumed to be non-negative integers. The edges of $G(T)$ represent the order in which jobs generated by T are released. Each edge (u, v) is labeled with a non-negative integer $p(u, v)$ denoting the minimum job inter-release separation time. We do not assume a relation between job deadlines $d(u)$ and inter-release separation times $p(u, v)$, i.e., the jobs may have *arbitrary deadlines*. The constraint set $C(T) = \{(from_1, to_1, \gamma_1), \ldots, (from_k, to_k, \gamma_k)\}$ contains k additional global minimum inter-release separation constraints. Each constraint $(from_i, to_i, \gamma_i) \in C(T)$ expresses that between the visits of vertices $from_i$ and to_i, at least γ_i time units must pass. We assume all γ_i to be non-negative integers. For a fixed constant k, we call a task T with $k = \|C(T)\|$ a k-EDRT task.

Note that constraints in $C(T)$ can involve any pair of vertices, they can be self-loops, they can be connected or unconnected vertices, the constraints could even be 0. (Some of these combinations would only make very limited sense.) Further, note that for $k = 0$, i.e., no additional constraints with $C(T)$ being empty, this is the plain DRT model from [2].

Example II.1. *Figure 2 shows an example of a 2-EDRT task with constrained deadlines. We will use it as a running example throughout the rest of the paper.*

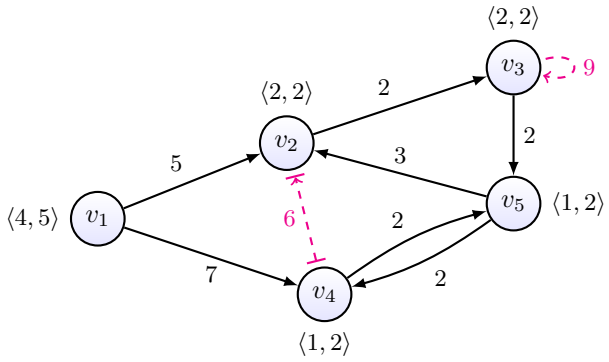


Fig. 2. An example EDRT task containing five different types of jobs and two additional constraints $(v_4, v_2, 6)$ and $(v_3, v_3, 9)$, denoted with dashed arrows. Note that these dashed arrows do *not* represent edges that can be taken. They only denote additional timing constraints.

Semantics: An execution of task T corresponds to a potentially infinite path in $G(T)$. Each visit to a vertex along that path triggers the release of a job with parameters specified by the vertex labels. The job releases are constrained by inter-release separation times specified by both the edge labels and the constraints from $C(T)$.

To explicitly denote necessary waiting times at the vertices, we extend paths in $G(T)$ with delay information:

Definition II.2 (Timed Path)**.** *For a path $\pi = (\pi_0, \ldots, \pi_l)$ in $G(T)$, a* timed path *$\tilde{\pi} = (\pi_0, \delta_0, \pi_1, \delta_1, \ldots, \delta_{l-1}, \pi_l)$ is derived by adding waiting times $\delta_i \in \mathbb{R}_{\geqslant 0}$ between the vertices. We call $\tilde{\pi}$*

- legal*, if the waiting times are consistent with the given inter-release separation constraints, i.e., if for $i < j \leqslant l$:*
  1) *$\delta_i \geqslant p(\pi_i, \pi_{i+1})$, and*
  2) *$\delta_i + \delta_{i+1} + \ldots + \delta_{j-1} \geqslant \gamma$ for all $(\pi_i, \pi_j, \gamma) \in C(T)$,*
- urgent*, if all δ_i are minimal for $\tilde{\pi}$ being legal, i.e., for all prefixes $(\pi_0, \delta_0, \ldots, \delta_{j-1}, \pi_j)$ of $\tilde{\pi}$ and all $\varepsilon > 0$, the timed path $(\pi_0, \delta_0, \ldots, \delta_{j-1} - \varepsilon, \pi_j)$ is* not *legal.*

This extends to infinite paths in a natural way. Note that legal timed paths may contain non-integral delays δ_i, but for urgent timed paths, all δ_i are necessarily integers[1].

Using timed paths, we can define the semantics of an EDRT task set τ as the set of *job sequences* that can be generated by τ. A *job* is denoted by a 3-tuple (r, e, d) with release at (absolute) time r, worst-case execution time e and deadline at (absolute) time d. A job sequence $\sigma = [(r_0, e_0, d_0), (r_1, e_1, d_1), \ldots]$ is *generated by* T, if and only if there is a (potentially infinite) timed path $\tilde{\pi} = (\pi_0, \delta_0, \pi_1, \delta_1, \ldots)$ that is legal for T and satisfies for all i:

1) $e_i = e(\pi_i)$,
2) $d_i = r_i + d(\pi_i)$,
3) $r_{i+1} - r_i = \delta_i$.

[1]Otherwise, the fractional part of the first non-integral delay could be moved to the next delay. This preserves legality of the path, but shortens the accumulated duration of the prefix, showing that the original path was not urgent.

For a task set τ, a job sequence σ is *generated by* τ, if it is a composition of sequences $\{\sigma_T\}_{T\in\tau}$, which are individually generated by the tasks T of τ.

Example II.3. *For the example task T in Figure 2, consider the job sequence $\sigma = [(2.3, 1, 4.3), (5.1, 1, 7.1), (8.9, 2, 10.9)]$. It is generated by T, since it corresponds to timed path $\tilde{\pi} = (v_4, 2.8, v_5, 3.8, v_2)$ which is legal. If we reduce the second delay in $\tilde{\pi}$ to get $\tilde{\pi}' = (v_4, 2.8, v_5, 3, v_2)$, the result would not be legal: v_2 is visited 5.8 time units after v_4 which makes $\tilde{\pi}'$ violate the constraint $(v_4, v_2, 6)$.*

Note that $\tilde{\pi}$ is legal, but not urgent. An urgent timed path involving the same sequence of vertices is $\tilde{\pi}'' = (v_4, 2, v_5, 4, v_2)$. In fact, $\tilde{\pi}''$ is the unique urgent timed path along these vertices.

III. Feasibility

The main focus of this work on the EDRT task model is to solve the associated feasibility problem:

Definition III.1 (Feasibility)**.** *A task set τ is* preemptive uniprocessor feasible*, if and only if all job sequences generated by τ can be executed on a preemptive uniprocessor platform such that all jobs meet their deadlines.*

In particular, for a job (r, e, d) to be scheduled successfully, there must be an accumulated duration of e time units where the job executes exclusively on the processor within the time interval $[r, d]$. It is known that Earliest Deadline First (EDF) is an optimal scheduling algorithm for scheduling independent jobs on a preemptive uniprocessor. Thus, the *feasibility* problem is equivalent to *EDF schedulability*.

The two main results in this paper are that the feasibility problem for k-EDRT is tractable (Section IV), but for the general EDRT model, i.e., without a bound on the number of constraints, it is strongly *coNP*-hard (Section VI). In the remainder of this section, we introduce some general concepts related to feasibility analysis which will be used for establishing the two main results.

A. The Demand Bound Function

A common framework in schedulability theory for deciding feasibility is the *demand bound function (dbf)*. Intuitively, a *dbf* expresses the accumulated execution time that a task set can *demand* from the processor within any time interval of given length. In particular, it considers each execution requirement that is both *released* within the interval and needs to be *finished* before the end of the interval. Formally:

Definition III.2 (Demand Bound Function)**.** *For a task T and an interval length t, $dbf_T(t)$ denotes the* maximum *cumulative execution time requirement of jobs with both release time and deadline within an interval of length t, over all job sequences generated by T. Further, for a task set τ,*

$$dbf(t) := \sum_{T\in\tau} dbf_T(t).$$

By definition, *dbf* is *tight* in the sense that for each t, there is a job sequence generated by task set τ in which some jobs actually have an execution demand of $dbf(t)$ within an interval of t time units. Note that the definition of $dbf(t)$ as a sum of $dbf_T(t)$ of all tasks T relies on their independence of each other. Note further that we assume time to be dense, so *dbf* is defined for all $t \in \mathbb{R}_{\geqslant 0}$.

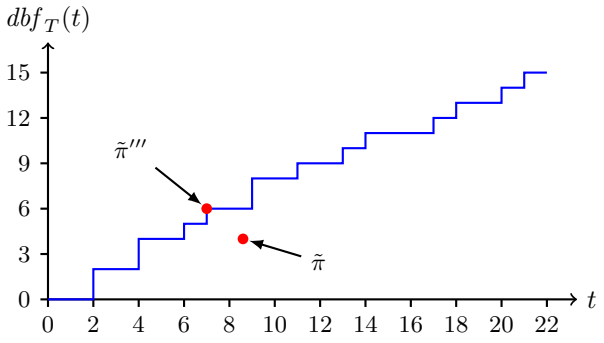


Fig. 3. Demand bound function for the EDRT task in Figure 2. Two depicted dots in this diagram illustrate the connection between the *dbf* and timed paths through $G(T)$ using $\tilde{\pi}$ and $\tilde{\pi}'''$ from Example III.3.

Example III.3. *Consider again job sequence $\sigma = [(2.3, 1, 4.3), (5.1, 1, 7.1), (8.9, 2, 10.9)]$ from Example II.3, generated by task T from Figure 2. This sequence shows that in a time interval $t = 10.9 - 2.3 = 8.6$, task T may generate a demand of $1+1+2 = 4$ on the processor. Thus, $dbf_T(8.6) \geqslant 4$. In fact, we can derive this information from the associated timed path $\tilde{\pi} = (v_4, 2.8, v_5, 3.8, v_2)$ by summation over the execution demand labels and by summation over the delays and addition of relative deadline $d(v_2)$.*

Considering the urgent timed path $\tilde{\pi}'' = (v_4, 2, v_5, 4, v_2)$, also from Example II.3, we see that already within 8 time units, a demand of 4 can be created along that path. However, a third timed path $\tilde{\pi}''' = (v_1, 5, v_2)$ with an accumulated execution demand of 6 shows that within already 7 time units, the demand can be even higher. In fact, this urgent timed path $\tilde{\pi}'''$ creates the maximum demand within an interval of 7 (and also 8) time units. Thus, $dbf_T(7) = dbf_T(8) = 6$, as we can also read from Figure 3.

We are especially interested in *urgent* timed paths, because they play an important role when calculating the *dbf*. Knowing only the urgent timed paths is sufficient for determining the *dbf*, as follows. A timed path's duration, i.e., the sum of all its delays, does not increase by making it urgent (via adjustment of its delays). Thus, for every timed path, there is an urgent timed path which contributes just as much information to the *dbf*. Consequently, even though the *dbf* is concerned with *all* timed paths, only the *urgent* timed paths need to be considered. Further, any prefix of an urgent timed path is by definition urgent as well. This is critical for an enumeration approach based on extending paths further and further, which is why our definition of urgency is that strict. Note that this renders the second of the following two timed paths through $G(T)$ in Figure 2 non-urgent, even though they have the same duration:

$$(v_4, 2, v_5, 4, v_2) \quad \text{vs.} \quad (v_4, 3, v_5, 3, v_2)$$

A tight demand bound function can be used in a precise feasibility test, thanks to the following proposition:

Proposition III.4. *An EDRT task system τ is preemptive uniprocessor feasible if and only if:*

$$\forall t \geqslant 0 : dbf(t) \leqslant t$$

A proof of this can be established in a very similar way as in previous work, e.g. [3], and we omit it here. Intuitively, there must be a feasible schedule for all job sequences generated by τ if and only if, for all time interval lengths t, the execution time demand of τ fits into that interval.

For the plain DRT model, we showed in [2] how this proposition can be used for an efficient feasibility test, by efficient computation of dbf for a bounded number of relevant values for t. We will use that approach as a basis in Section IV.

B. Utilization

Another useful metric is the *utilization* of a task set. Intuitively, the utilization describes the maximum execution demand rate that the task set may create *asymptotically*. It is important for applying the efficient feasibility test sketched above for Proposition III.4. As shown in [2], the bound up to which condition $dbf(t) \leqslant t$ needs to be checked is derived from the utilization of the given task set.

For a formal definition, we can use the demand bound function introduced above.

Definition III.5 (Utilization)**.** *For a task T and a task set τ, we define their* utilizations*:*

$$U(T) := \lim_{t \to \infty} \frac{dbf_T(t)}{t}$$
$$U(\tau) := \sum_{T \in \tau} U(T)$$

This definition is more general than the one in [2] which is based on the "most dense" cycle through the DRT task graph. However, it is easy to see that for plain DRT, i.e., without constraints $C(T)$, both definitions are equivalent.

Clearly, a task set with a utilization above 1 can not be schedulable, since for a sufficiently large interval t, its demand bound function must exceed t. Thus, we can restrict our attention to task systems with a utilization bounded by a constant $c < 1$, similar to [2].

IV. $k-$Bounded Case: Efficient Analysis

As a first step in presenting our method for efficiently deciding feasibility for a k-EDRT task system, we give a short overview of the method described in [2] for plain DRT. We focus on the calculation of the demand bound function for a given DRT task T. It is calculated up to a certain bound which depends on the task set's utilization.

- The first observation (cf. Figure 3) is that the demand bound function changes only at certain points ("steps"), and thus it is sufficient to determine these points.
- Each point corresponds to a path through $G(T)$. More precisely, it corresponds to an *urgent timed path*; such paths in the plain DRT model have the property that all delays correspond exactly to the edge labels. Thus, it is sufficient to just consider ordinary paths through $G(T)$, since all delays can be directly read from the edges. From each path π, one can derive its accumulated execution demand $e(\pi)$ and deadline $d(\pi)$, both via summation over parameters of its vertices and edges. In particular, for $\pi = (\pi_0, \ldots, \pi_l)$, one defines:

$$\textit{Execution demand:} \quad e(\pi) := \sum_{i=0}^{l} e(\pi_i)$$
$$\textit{Deadline:} \quad d(\pi) := \sum_{i=0}^{l-1} p(\pi_i, \pi_{i+1}) + d(\pi_l)$$

  The pair $\langle e(\pi), d(\pi) \rangle$ is called a *demand pair*. Determining all demand pairs is sufficient because of the following correspondence:

$$dbf_T(t) = \max \{ e \mid \langle e, d \rangle \text{ demand pair with } d \leqslant t \}.$$

- In order to determine the set of all demand pairs up to t, all paths through $G(T)$ up to a certain length need to be considered. Since the number of such paths is exponential, a path abstraction called *demand triples* is introduced. A demand triple $\langle e, d, v \rangle$ consists of a demand pair $\langle e, d \rangle$ and a vertex v from $G(T)$, and is an abstraction of all paths $\pi = (\pi_0, \ldots, \pi_l)$ with $e = e(\pi)$, $d = d(\pi)$ and $v = \pi_l$. The total number of possible demand triples up to a given t is pseudo-polynomially bounded, preventing exponential explosion of the method.
- Finally, calculation of $dbf_T(t)$ is done[2] by an iterative procedure that generates all demand triples, starting from 0-paths, i.e., with the set $\{\langle e(v_i), d(v_i), v_i \rangle \mid v_i \in G(T)\}$. In each step, a previously generated demand triple $\langle e, d, v \rangle$ is extended by considering all outgoing edges from v, one by one, to obtain potentially new demand triples. Thus, longer and longer paths up to t are explored and eventually all relevant demand pairs (as part of the demand triple abstraction) are computed.

This concludes the overview of the method from [2]. Without additional timing constraints $C(T)$, it provides a feasibility test that runs in pseudo-polynomial time for systems with bounded utilization. However, if $k > 0$, it is not directly applicable anymore. The reason is that an urgent timed path $\tilde{\pi} = (\pi_0, \delta_0, \ldots, \delta_{l-1}, \pi_l)$ in the k-EDRT model can not as easily be extended with a new vertex v. In the plain DRT model, in order to derive the new delay δ_l to extend $\tilde{\pi}$ with v for obtaining $(\pi_0, \delta_0, \ldots, \pi_l, \delta_l, v)$, it is sufficient to just consider the previously last vertex π_l, since the delay δ_l will just be $p(\pi_l, v)$. This is why the demand triple abstraction works, since it only needs to record the last vertex of the

[2]The sketched method is directly applicable for tasks with *constrained deadlines*, i.e., where for all vertices u and their outgoing edges (u, v) it is required that $d(u) \leqslant p(u, v)$. A slight modification of the method, also presented in [2], makes it suitable for use in the general setting of *arbitrary deadlines*, which we assume for the EDRT model in this paper.

abstracted paths. However, in the presence of additional constraints from $C(T)$, v may be the to_i vertex of some constraint, imposing additional waiting time. We call such a constraint *active*. Therefore, earlier vertices visited in $\tilde{\pi}$ (and their delays) also need to be considered. In other words, since the demand triple abstraction only records the last vertex, it "forgets" information about the active constraints.

Example IV.1. *To illustrate this problem, consider the urgent timed path $\tilde{\pi}'' = (v_4, 2, v_5, 4, v_2)$ from Example II.3 for the example task in Figure 2. The sketched graph exploration would start with a demand triple $\xi^{(1)} = \langle 1, 2, v_4 \rangle$ for 0-path $\tilde{\pi}^{(1)} = (v_4)$ and extend it to $\xi^{(2)} = \langle 2, 4, v_5 \rangle$ as an abstraction of path $\tilde{\pi}^{(2)} = (v_4, 2, v_5)$. Clearly, this demand triple $\xi^{(2)}$ (in contrast to the timed path $\tilde{\pi}^{(2)}$ that it abstracts) lost the information that constraint $(v_4, v_2, 6)$ is active at v_5. Thus, we can not derive from $\xi^{(2)}$ that a delay of at least 4 time units is necessary before visiting v_2.*

A similar problem arises for determining a task's utilization. This is done in [2] by finding simple cycles in $G(T)$. Consider cycle (v_5, v_2, v_3, v_5) in $G(T)$. Because of the constraint on revisiting v_3 earliest after 9 time units, the duration of this cycle is actually 9 time units, giving it a density of $5/9$. (That is, accumulated execution time for all vertices but the last, divided by the path's duration.) The other simple cycle (v_5, v_4, v_5) has an even lower density of $1/2$. However, the real utilization of T is in fact $7/12$, demonstrated via cycle $(v_5, v_2, v_3, v_5, v_4, v_5)$ with a density of $7/12$. Note that this cycle is not simple.

In summary, we are facing two challenges when adapting the feasibility test to the k-EDRT setting:

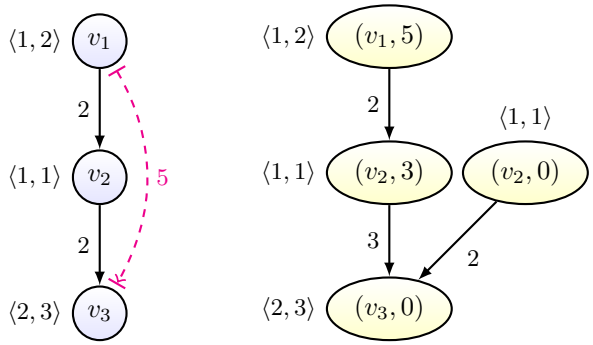
1) While traversing $G(T)$ using the demand triple abstraction, we must keep information about the *state* of all k constraints, i.e., to what extent they are active.
2) The utilization of a task can not be determined easily by just considering simple cycles in $G(T)$ as done in [2]. Also here, the additional constraints must be honored and a "most dense" timed cycle may not be simple.

We solve both by translating each given k-EDRT task T into an equivalent plain DRT task T'. The key idea is to store information about the constraints of the original task in the vertices of the new task while adjusting the edges accordingly. For each constraint $(from_i, to_i, \gamma_i)$, we keep a *countdown* starting at γ_i in all vertices. It records how many time units at least passed since $from_i$ has been visited the last time. Consequently, vertex to_i is only allowed to be visited when the corresponding countdown is 0, potentially imposing an additional waiting penalty. These additional delays are considered when labeling the edges of T', making them potentially larger than the corresponding edge labels of T. After the translation of all tasks in the k-EDRT model τ, we can run the analysis method from [2] on the newly created DRT model τ' in order to solve the feasibility problem. Thus, the task transformation is the main focus for the rest of this section.

Note that the countdowns can be restricted to integer values, since we are only interested in urgent timed paths through $G(T)$, which in turn only contain integer delays. However, even with integers, the set of vertices of $G(T')$ may grow rapidly in size compared to $G(T)$, although this growth is polynomially bounded in the constraint values γ_i. Further, we introduce some optimizations in Section V that greatly reduce the number of vertices in T' and the actual overhead during graph traversal.

A. Task Transformation Details

We start by illustrating the task transformation using a very basic example.



(a) EDRT task T (b) (Parts of) DRT task T'

Fig. 4. A basic example of a 1-EDRT task T being transformed into an equivalent DRT task T'. Only a subset of the vertices of T' is shown.

Example IV.2. *Consider the example in Figure 4. The shown task T has three vertices with the constraint $(v_1, v_3, 5)$. Since there is one constraint, we extend each vertex with* one *countdown:*

- *Vertex v_1 is extended to $(v_1, 5)$ in T' since it is the starting vector of the constraint. The constraint value is 5, so the countdown gets the value 5.*
- *From $(v_1, 5)$, we create an edge to $(v_2, 3)$ since there is an edge (v_1, v_2) in T with label $p(v_1, v_2) = 2$. The constraint does not involve v_2, so the countdown is just decreased by 2, which is also the edge label.*
- *From $(v_2, 3)$, we want to create an edge to a vertex involving v_3, since there is an edge (v_2, v_3) in T. However, v_3 is the target vertex of the given constraint. Thus, the countdown needs to decrease to 0, to ensure constraint satisfaction. Consequently, the edge goes to vertex $(v_3, 0)$ in T'. The edge label is 3 which is the required waiting time.*
- *Finally, if one starts at v_2, the constraint is not active, which we model by another vertex $(v_2, 0)$ in T'. Its countdown is 0 and thus allows visiting $(v_3, 0)$ after just $p(v_2, v_3) = 2$ time units.*

Note that the actual transformation T' contains more vertices for v_2 and v_3 with other countdown values. They are however not essential and therefore left out for this example.

We now give the full details of how to construct an equivalent plain DRT model T' from a k-EDRT model T. We need to describe the set of vertices, their labels, the set of edges and their labels. Note that this is a theoretical description of the transformation and sufficient for the theoretical complexity result. We introduce some powerful optimizations in Section V for efficient implementations.

Vertices: For each vertex $v \in G(T)$, we create vertices (v,t) for $G(T')$, where $t = (t_1, \ldots, t_k)$ is a *countdown vector*. A priori, we do not know which of the possible values will actually be used, so we need to create vertices for all possible combinations of values for countdowns t_i. A countdown t_i is associated with constraint $(\mathit{from}_i, \mathit{to}_i, \gamma_i)$. It has value γ_i if v is the starting vertex of the associated constraint, i.e., $v = \mathit{from}_i$. This expresses that we want to record that T just visited from_i and at least γ_i time units must pass before to_i may be visited. Otherwise, t_i can have any integer value between 0 and γ_i, since we do not know beforehand how long ago from_i has been visited last when visiting v in T. All possible (v,t) consistent with this description are being created. Formally, for each $v \in G(T)$, we create as vertices for T' all (v,t) such that:

$$\forall i : \begin{cases} t_i = \gamma_i, & \text{if } v = \mathit{from}_i \\ t_i \in \{0, \ldots, \gamma_i\}, & \text{otherwise} \end{cases}$$

Vertex labels: Each new vertex (v,t) in $G(T')$ has the same label as vertex v in $G(T)$, since it represents the release of jobs of the same type. Formally:

$$\forall (v,t) \in G(T') : \begin{cases} e\big((v,t)\big) := e(v) \\ d\big((v,t)\big) := d(v) \end{cases}$$

Edges: Given two vertices (u,s) and (v,t) in $G(T')$ we need to decide whether there should be an edge between them in $G(T')$. For each combination of u, s and v, there will be exactly one countdown vector t for which we create such an edge:

1) $t_i = \gamma_i$ for all i such that $v = \mathit{from}_i$, expressing that v is resetting a countdown since it is the *starting* vertex of the associated constraint.
2) Otherwise, t_i is s_i decremented by some waiting time. For all i, the countdowns are decremented by the *same* amount of time δ. Countdowns below 0 are set to 0.
3) The waiting time δ is at least $p(u,v)$ and also not smaller than any s_i for all i such that $v = \mathit{to}_i$. This expresses that all constraints involving v as *target* vertex need to be satisfied and the waiting time sufficiently large to guarantee that.

Formally, we first calculate the waiting time δ:

$$\delta(u,s,v) := \max\Big(p(u,v), \{s_i \mid v = \mathit{to}_i\}_{i=1}^{k}\Big)$$

Second, the following must hold for the new countdown vector t:

$$\forall i : t_i = \begin{cases} \gamma_i, & \text{if } v = \mathit{from}_i \\ \max\big(0, s_i - \delta(u,s,v)\big), & \text{otherwise.} \end{cases}$$

Edge labels: For an edge from (u,s) to (v,t), the delay is just the δ we computed above.

$$p\big((u,s),(v,t)\big) := \delta(u,s,v)$$

It's important to note that the countdown vector does *not* represent dynamic information during "run-time", but is a rather static property of the vertices in T'. To illustrate this, consider again the task from Figure 4. In the transformed task in Figure 4(b), assume a timed path starts in $(v_1, 5)$. Assume further that the system now waits for more than the minimum time, e.g., 3 instead of 2 time units. When moving to the successor vertex representing v_2, we *still* arrive at $(v_2, 3)$. This is because the edge represents only the *minimal* possible waiting time. Therefore, the countdowns as a static part of the vertices only record the effect of the edges, not of the actual "run-time behavior". Consequently, the system in this example is required to wait again for 3 time units (not just 2) before the next step, which would be to $(v_3, 0)$. In other words, the timed path $(v_1, 3, v_2, 2, v_3)$ which is legal in T does not have a corresponding timed path in T', i.e., with the same delays. However, this is not a problem, since it is not an *urgent* path. The objective of our presented task transformation is to preserve urgent paths of T (which indeed all have a counterpart in T') since only these define the dbf.

Example IV.3. *Consider again the running example task T from Figure 2. After applying the described transformation to T, we get an equivalent plain DRT task T' of which the most essential parts are shown in Figure 5. In fact, after the vertex removal optimization discussed below in Section V, the vertices from Figure 5 are the only remaining ones.*

Note that the cycle $(v_5, v_2, v_3, v_5, v_4, v_5)$ in $G(T)$ that we identified in Example IV.1 as the one with the highest density translates to a simple cycle in T'.

B. Correctness

For the task transformation method from above, we show now its correctness and summarize the whole k-EDRT analysis method in order to establish our first main technical result.

We use a central correctness lemma to prove that the demand bound functions of the given k-EDRT task T and the transformed DRT task T' coincide. Intuitively, every urgent timed path in T has a legal corresponding path through T' by extending the vertices with appropriate countdown vectors. Further, all urgent timed paths in T' are legal in T (after removing countdown vectors from the vertices) since the countdowns ensure satisfaction of all inter-release separation constraints. A formal proof is given in Appendix A.

Lemma IV.4. *For a k-EDRT task T and its transformation T', their demand bound functions coincide, i.e.,*

$$\forall t \geqslant 0 : \mathit{dbf}_T(t) = \mathit{dbf}_{T'}(t).$$

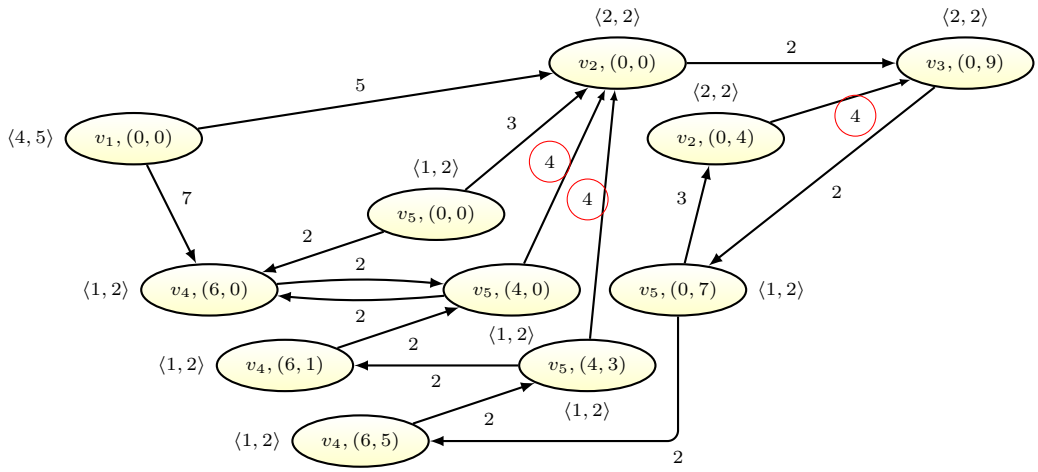


Fig. 5. Plain DRT task T' after applying the transformation to the example EDRT T task from Figure 2. Encircled edge labels are larger than labels of corresponding edges in $G(T)$ since extra waiting time is required by the additional constraints from $C(T)$, detected via countdowns. Note that only the most essential vertices are shown, i.e., those necessary for the dbf. All 216 vertices removed by the vertex removal optimization in Section V-B are hidden.

Given this lemma, the main theorem follows directly, since the result from [2] can be applied to the set of transformed tasks.

Theorem IV.5. *For constants $k \in \mathbb{N}$ and $c < 1$, feasibility for all k-EDRT task sets τ with $U(\tau) \leqslant c$ can be decided in pseudo-polynomial time.*

Proof: Given τ, we apply the described transformation to all tasks in order to obtain τ'. Lemma IV.4 guarantees that their demand bound functions coincide, which implies that their utilizations are also the same (Definition III.5). Thus, we can apply the main result from [2], guaranteeing feasibility to be decidable for τ' (and therefore τ) in pseudo-polynomial time, if the following two conditions hold:

1) The number of vertices in τ' as well as the values in τ' (vertex and edge labels) are pseudo-polynomially bounded in the description of τ.
2) The transformation itself runs in pseudo-polynomial time.

For the first property, all labels are bounded by the old labels and the constraint values. Further, the number of new vertices per old vertex is bounded by $\gamma_1 \cdot \gamma_2 \cdots \gamma_k$, since this is the number of possible countdown vectors. With k being (bounded by a) constant, this is a polynomial in the values from τ. The second property is trivially satisfied since creating each of these vertices and edges (and their labels) is inexpensive. ∎

Note that we didn't assume any property resulting from assuming constrained deadlines. Thus, the result holds for task sets with arbitrary deadlines, since Lemma IV.4 only relies on the property that urgent timed paths are defining the demand bound function. This is true for the case of arbitrary deadlines as well. Further, the result in [2] also holds for arbitrary deadlines and thus can be transferred to the k-EDRT case.

V. Optimizations

While the described method is sufficient to show the theoretical pseudo-polynomial complexity bound, there are a few ways to optimize implementations for higher efficiency. First, all optimizations discussed in [2] are applicable when analyzing the transformed task set. Further, we discuss three optimizations that are specific to the task transformation and may have drastic impact on analysis runtime: A *refined domination relation*, *removal of unnecessary vertices* and a *countdown compression* method.

A. Refined Domination Relation

In [2], a *domination relation* between demand triples is introduced. A demand triple $\langle e, d, v\rangle$ dominates $\langle e', d', v'\rangle$, if the abstracted paths can create a higher demand ($e \geqslant e'$) within a shorter interval ($d \leqslant d'$) and end in the same vertex ($v = v'$). The motivation is that in this case, $\langle e', d', v'\rangle$ does not contribute new information to the dbf calculation. Further, the same is true for all future extensions of $\langle e', d', v'\rangle$, since $\langle e, d, v\rangle$ ends in the same vertex and can thus be extended in the exact same way. Consequently, $\langle e', d', v'\rangle$ does not need to be considered further and can be discarded.

This concept can be refined for the analysis of a translated T'. We have *domain specific* information about the involved vertices: they represent countdown vectors. A countdown vector constrains future behavior of the path, since it may impose additional waiting times. Thus, if we have two vertices (v, s) and (v, t) in T' with $\forall i : s_i \leqslant t_i$, all path extensions possible from (v, t) are also possible from (v, s), in potentially shorter time. This relation can be added to the domination relation in order to implement further discarding opportunities.

Example V.1. *Consider the following two paths through $G(T')$ in Figure 5, together with their demand triple abstrac-*

tions:

$$\big((v_5,(4,3)),(v_4,(6,1))\big) \quad \rightsquigarrow \quad \langle 2,4,(v_4,(6,1))\rangle$$
$$\big((v_5,(0,7)),(v_4,(6,5))\big) \quad \rightsquigarrow \quad \langle 2,4,(v_4,(6,5))\rangle$$

Strictly speaking, both demand triples are involving different vertices from T', so no optimization from [2] could be applied for discarding one of them. However, we know that they represent the same vertex in the original EDRT task T (Figure 2). Further, comparing the countdown vector $(6,1)$ to $(6,5)$, we notice that the first one is less restrictive than the second one, since all countdowns are smaller or equal. Consequently, the second demand triple can be discarded.

B. Vertex Removal

For a second optimization, we note that the first optimization described above applies particularly to the initial demand triples. In our running example from Figure 2, the transformation will produce, among others, the two vertices $(v_4,(6,0))$ and $(v_4,(6,8))$. (Only the first one is shown in Figure 5.) They only differ in the countdown for the second constraint, and whatever timed path is possible from $(v_4,(6,8))$ has a corresponding timed path starting in $(v_4,(6,0))$, which is less restrictive, as discussed above in Section V-A. Thus, with the above optimization, already the initial demand triple containing $(v_4,(6,8))$ can be discarded. We additionally notice that the vertex $(v_4,(6,8))$ does not have any incoming edges, since these would reduce the second countdown by at least 2 (and it has a maximum of 9). This means that the graph exploration of the transformed task T' does not *start* at that vertex and actually will never *visit* it. Thus, we can remove that vertex from T' altogether without influencing the represented timed paths through T and thus the dbf.

In general, we can remove from T' all vertices (v,t) which do *not* have any incoming edges and for which t does *not* have the following shape:

$$\forall i : t_i = \begin{cases} \gamma_i, & \text{if } v = from_i \\ 0, & \text{otherwise.} \end{cases}$$

We also remove their outgoing edges. This removal procedure can be repeated until no such vertex exists anymore. (Note that by removing vertices, other vertices may lose their incoming edges and be thus also eligible for removal.)

The described vertex removal procedure is quite effective. In the example from Figure 2, the EDRT task has 5 vertices which results in 227 vertices after task transformation to a plain DRT task. However, after applying the described optimization, only 11 vertices remain (those shown in Figure 5), resulting in an analysis speed-up of several orders of magnitude.

A different way of realizing this optimization is to create vertices only *on the fly* during analysis. This means that an implementation would *not* start the analysis by first creating all vertices and edges in memory, removing unnecessary ones afterwards. Instead, only the vertices with countdown vectors of the shape described above are created, i.e., one per vertex in the original EDRT model. As graph exploration proceeds, new vertices with updated countdowns are created. By doing so, only the necessary vertices are actually created and memory use may be reduced significantly. However, with this approach, it is necessary to adjust the way the method from [2] calculates its bound up to which the dbf must be computed.

C. Countdown Compression

For our third optimization, we note that a special situation occurs if constraints are located "far away" from each other in T'. Imagine a task with a rather big directed graph, where two vertices u and v are involved in constraints $(u,u,10)$ and $(v,v,10)$. However, assume all timed paths between both vertices involve accumulated delays of more than 10. Clearly, at most one of the two constraints is active at any time. They do not *overlap*. Thus, their corresponding countdowns in T' after the vertex removal described above are never non-zero at the same time.

In other words, their countdown could be re-used and thus we could *compress* the countdown vector. In the simple example of the two constraints $(u,u,10)$ and $(v,v,10)$, we would just use one countdown and an additional bit indicating which of the two constraints is being counted. In fact, if T contains more than k constraints, e.g., linearly many, it could still be analyzed by the presented method for k-EDRT, by using the sketched compression optimization. Consequently, Theorem IV.5 can be generalized to EDRT task sets with up to k *non-overlapping* constraints. Note that the non-overlapping property can be verified by a simple graph traversal.

VI. Unbounded Case: Strong coNP-hardness

In order to show $coNP$-hardness in the strong sense of the feasibility problem for general EDRT models, we provide a reduction from the classical *Hamiltonian Path Problem* (or rather its complement).

Definition VI.1. *The problem of deciding whether a directed graph G with n vertices contains a simple path with n vertices, i.e., n unique vertices, is called the* Hamiltonian Path Problem.

Proposition VI.2 ([8])**.** *The Hamiltonian Path Problem is NP-hard in the strong sense.*

We provide a reduction from the Hamiltonian Path Problem as follows. Given a directed graph G, we construct a task set τ with the following properties:

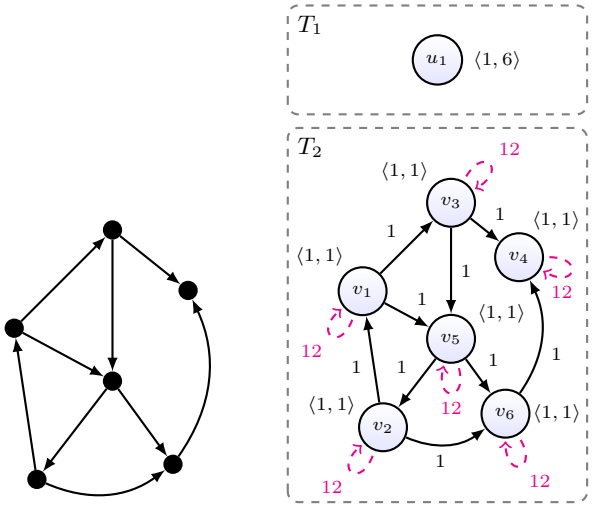
1) If G contains a Hamiltonian Path, τ is infeasible.
2) If G does not contain a Hamiltonian Path, τ is feasible.
3) The number of vertices in the tasks of τ and all involved values (labels and constraints) need to be polynomially bounded in the size of G.
4) Given a constant $c < 1$, we must be able to construct τ such that $U(\tau) \leqslant c$.

The third requirement is necessary to establish $coNP$-hardness in the *strong* sense. The fourth requirement is also necessary since we usually restrict ourselves to a class of task sets with a utilization bounded by a constant $c < 1$. We want to show that for any choice of c, the problem stays strongly $coNP$-hard.

For presentation reasons, we first assume $c = 1/2$. A simple generalization to arbitrary $c < 1$ is given afterwards. We construct the task set τ from graph G with n vertices as follows. The task set contains two tasks:

- The first task T_1 contains just one vertex u_1 without any edges, and the label $\langle e(u_1), d(u_1)\rangle = \langle 1, n\rangle$.
- The second task T_2 uses G as underlying graph. All vertices are labeled with $\langle 1, 1\rangle$ and all edges with 1. Further, its constraint set contains self-loops with label $2n$, i.e., $C(T_2) := \{(v, v, 2n) \mid v \in G(T_2)\}$.

Note that all n constraints are potentially overlapping, so this is not a case in which the *countdown compression* optimization from Section V-C applies. We illustrate the construction with the following example.



(a) Given digraph G (b) Constructed task set τ

Fig. 6. Example for construction of task set $\tau = \{T_1, T_2\}$ from a given digraph G containing 6 vertices. This example contains a Hamiltonian Path, making τ infeasible.

Example VI.3. *Consider graph G in Figure 6(a) with 6 vertices. We note that G contains a Hamiltonian Path. Figure 6(b) shows the constructed task set. We see that T_2 may release 6 jobs along the timed path $(v_1, 1, v_3, 1, v_5, 1, v_2, 1, v_6, 1, v_4)$ which corresponds to the Hamiltonian Path from G. This causes an execution demand of 6 within 6 time units (5 for releasing all jobs, 1 for the last deadline). Together with the job from T_1, this task set is clearly infeasible.*

We now check the four properties from above. First, if there is a Hamiltonian Path in G, task T_2 may release n jobs along that path within $n-1$ time units. Together with T_1, the system is overloaded in an interval of length n. A formal proof is given in Appendix B.

Lemma VI.4. *If graph G contains a Hamiltonian Path, then $dbf(n) \geqslant n + 1$.*

Second, if there is no Hamiltonian Path in G, then the constraints from the constructed $C(T_2)$ prevent the system from overloading, since within up to $2n$ time units, T_2 can only create a demand of at most $n - 1$, one per time unit. A formal proof is given in Appendix C.

Lemma VI.5. *If graph G does not contain a Hamiltonian Path, then $\forall t \geqslant 0 : dbf(t) \leqslant t$.*

Finally, we summarize our second main result in the following theorem.

Theorem VI.6. *For any constant $c < 1$, the feasibility problem for EDRT task sets τ with $U(\tau) \leqslant c$ is coNP-hard in the strong sense.*

Proof: We first assume $c \in [1/2, 1)$. In that case, the construction introduced above provides a task set τ with $U(\tau) \leqslant 1/2 \leqslant c$. The task set's utilization is at most $1/2$, since $U(T_1) = 0$ with T_1 being acyclic, and $U(T_2) \leqslant 1/2$ because any of the n vertices can only be re-visited after at least $2n$ time units. Therefore, even in the presence of a Hamiltonian Path, $dbf_{T_2}(t) \leqslant dbf_{T_2}(t - 2n) + n$ for $t \geqslant 2n$, resulting in the claimed utilization.

The first two properties from above for a proper reduction are satisfied by Lemmas VI.4 and VI.5 using the exact dbf characterization from Proposition III.4. The third property is also clear since all numbers and values are linearly bounded in n. Thus, for $c \in [1/2, 1)$, we are done.

Finally, in order to satisfy also the last property, note that for $c < 1/2$ we can change the task construction by setting $C(T_2) := \{(v, v, n/c) \mid v \in G(T_2)\}$ instead. This reduces $U(T_2)$ to at most c and leaves all other properties satisfied. ■

We note that the presented reduction constructs a task set with constrained deadlines. (For arbitrary deadlines, only one task is necessary and the construction can be even simpler.) Thus, even when restricting to constrained deadlines, the problem remains strongly *coNP*-hard.

Remark VI.7. *Theorem VI.6 holds even for the restricted class of task sets with constrained deadlines.*

VII. Conclusions and Future Work

In this paper we have studied the tractability border of the feasibility problem for task models with digraph-based release structures. We have introduced global inter-release separation constraints as a natural extension of the digraph real-time task model (DRT). Based on these, we have shown that the feasibility problem stays pseudo-polynomial if the number of constraints is bounded by a constant. The analysis technique uses the basic DRT model as a back-end to which tasks containing global constraints are transformed. The flexible structure of the DRT model proved useful for allowing such general transformations. This makes it possible to transfer results for the DRT model to other settings. As a second technical result, we have shown *coNP*-hardness if the number of global constraints is not bounded by a constant. This was done via a reduction from the Hamiltonian Path Problem, showing that graphs as a basis for modeling job releases in general add high complexity. Consequently, we have established a precise tractability borderline for the feasibility problem of digraph-based task models.

With this clear picture of the expressiveness of graph-based models, we plan to develop a tool based on our prototype implementation incorporating the described methods. We will do case studies to evaluate the practical applicability of the models and their analysis, and study other extensions, e.g. global deadlines.

REFERENCES

[1] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[2] M. Stigge, P. Ekberg, N. Guan, and W. Yi, "The Digraph Real-Time Task Model," in *Proc. of RTAS 2011*, pp. 71–80.

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

[4] S. K. Baruah, "Dynamic- and Static-priority Scheduling of Recurring Real-time Tasks," *Real-Time Syst.*, vol. 24, no. 1, pp. 93–128, 2003.

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

[6] M. Anand, A. Easwaran, S. Fischmeister, and I. Lee, "Compositional Feasibility Analysis of Conditional Real-Time Task Models," in *Proc. of ISORC 2008*, pp. 391–398.

[7] F. Eisenbrand and T. Rothvoß, "EDF-schedulability of synchronous periodic task systems is coNP-hard," in *Proc. of SODA 2010*, pp. 1029–1034.

[8] R. Karp, "Reducibility Among Combinatorial Problems," in *Complexity of Computer Computations*, R. Miller and J. Thatcher, Eds. Plenum Press, 1972, pp. 85–103.

APPENDIX

A. Proof of Lemma IV.4

Proof: In order to show that the demand bound functions of T and T' are identical, it suffices to show the following two properties about urgent timed paths through their graphs:

1) An urgent timed path in T has a corresponding legal timed path in T'.
2) An urgent timed path in T' has a corresponding legal timed path in T.

Both together imply that there can not be a t for which either of $dbf_T(t)$ and $dbf_{T'}(t)$ is larger: both values correspond to some urgent timed path each. If each of them corresponds to a legal timed path in the other model, the other demand bound function must be at least as large at that point.

For the first claim, let $\tilde{\pi} = (\pi_0, \delta_0, \ldots, \pi_l)$ be an urgent timed path in T. We iteratively construct a corresponding path $\tilde{\pi}' = \big((\pi_0, t^{(0)}), \delta_0, \ldots, (\pi_l, t^{(l)})\big)$. All we need to do is to define all countdown vectors $t^{(i)}$ appropriately, since all other components are taken directly from $\tilde{\pi}$. We start with $t^{(0)}$, in which we set all countdowns as small as possible:

$$\forall m : t_m^{(0)} = \begin{cases} \gamma_m, & \text{if } \pi_0 = \mathit{from}_m \\ 0, & \text{otherwise.} \end{cases}$$

All succeeding $t^{(i)}$ are constructed by applying the rules presented in Section IV for edge construction. Since $\tilde{\pi}$ is urgent in T, all δ_i correspond exactly to the δ calculated in these rules. The resulting $\tilde{\pi}'$ must be a legal path in T' since all δ_i are equal to the edge labels, by construction.

For the second claim, let $\tilde{\pi}' = \big((\pi_0, t^{(0)}), \delta_0, \ldots, (\pi_l, t^{(l)})\big)$ be an urgent timed path in T'. We want to show that $\tilde{\pi} = (\pi_0, \delta_0, \ldots, \pi_l)$ is legal in T. From the construction of the waiting time δ in the transformation of T to T' it is clear that $\delta_i \geqslant p(\pi_i, \pi_{i+1})$ for all i. Further, for all constraints $(\mathit{from}_m, \mathit{to}_m, \gamma_m) \in C(T)$, any visit of to_m in $\tilde{\pi}'$ can not occur before the corresponding countdown is 0. Since the countdown is reset to γ_m whenever from_m is visited, the accumulated waiting time between from_m and to_m must be at least γ_m. Consequently, for any two constrained vertices in $\tilde{\pi}'$, i.e., $\pi_i = \mathit{from}_m$ and $\pi_j = \mathit{to}_m$, we have $\delta_i + \ldots + \delta_{j-1} \geqslant \gamma_m$. In summary, $\tilde{\pi}$ must be legal in T. ∎

B. Proof of Lemma VI.4

Proof: Let G be a digraph with n vertices containing a Hamiltonian Path $\pi = (\pi_0, \ldots, \pi_{n-1})$ and let $\tau = \{T_1, T_2\}$ be the constructed task set. We already have $dbf_{T_1}(n) = 1$ since the only job that can be released by T_1 has an execution demand of 1 and a deadline of n.

For $dbf_{T_2}(n)$, we consider the timed path $\tilde{\pi} = (\pi_0, 1, \pi_1, \ldots, 1, \pi_{n-1})$ that is constructed from π by inserting delays of 1. Since π is simple, all vertices are unique in $\tilde{\pi}$, making $\tilde{\pi}$ a legal timed path in $G(T_2)$, because all edge labels are 1 and all additional timing constraints only concern revisiting of vertices. Now we have with $l = n - 1$:

$$e(\tilde{\pi}) := \sum_{i=0}^{l} e(\pi_i) = n$$

$$d(\tilde{\pi}) := \sum_{i=0}^{l-1} \delta_i + d(\pi_l) = (n-1) + 1 = n$$

Thus, $dbf_{T_2}(n) \geqslant n$ since $\tilde{\pi}$ shows that T_2 can create an execution demand of n within a time interval of n.

Together, $dbf(n) = dbf_{T_1}(n) + dbf_{T_2}(n) \geqslant n + 1$. ∎

C. Proof of Lemma VI.5

Proof: Assume G does *not* contain a Hamiltonian Path. We want to show $dbf(t) \leqslant t$ for all $t \geqslant 0$ and do a case distinction for t. Recall that $dbf(t)$ includes only jobs that can be both released *and* have their deadlines within an interval of size t.

- $t \in [0, n)$: The job of T_1 does not count into such an interval since its deadline is n. Further, T_2 can only release up to $\lfloor t \rfloor$ jobs within an interval of length $t < n$ which also have their deadlines within the interval. Thus, $dbf(t) \leqslant \lfloor t \rfloor \leqslant t$.
- $t \in [n, 2n)$: Task T_1 can release its job and contributes 1 to the dbf. Further, task T_2 can only release up to $n-1$ jobs within an interval of that size, since vertices can not be revisited (recall constraints from $C(T_2)$) and there is no simple path that visits all vertices, by assumption. Consequently, $dbf(t) \leqslant n \leqslant t$.
- $t \geqslant 2n$: Within every additional $2n$ time units, T_2 can only release up to $n - 1$ more jobs. Together with the previous insight, we derive also in this case that $dbf(t) \leqslant t$.

∎