A Generalized Parallel Task Model for Recurrent Real-Time Processes

VINCENZO BONIFACI, IASI–Consiglio Nazionale delle Ricerche
ANDREAS WIESE, Max Planck Institute for Informatics
SANJOY K. BARUAH, University of North Carolina
ALBERTO MARCHETTI-SPACCAMELA, Sapienza Università di Roma
SEBASTIAN STILLER, Technische Universität Braunschweig
LEEN STOUGIE, CWI & Vrije Universiteit Amsterdam

A model is considered for representing recurrent precedence-constrained tasks that are to execute on multiprocessor platforms. A recurrent task is specified as a directed acyclic graph (DAG), a period, and a relative deadline. Each vertex of the DAG represents a sequential job, while the edges of the DAG represent precedence constraints between these jobs. All the jobs of the DAG are released simultaneously and need to complete execution within the specified relative deadline of their release. Each task may release jobs in this manner an unbounded number of times, with successive releases occurring at least the specified period apart. Conditional control structures are also allowed. The scheduling problem is to determine whether a set of such recurrent tasks can be scheduled to always meet all deadlines upon a specified number of identical processors.

This problem is shown to be computationally intractable, but amenable to efficient approximate solutions. Earliest Deadline First (EDF) and Deadline Monotonic (DM) are shown to be good approximate global scheduling algorithms. Polynomial and pseudo-polynomial time schedulability tests, of differing effectiveness, are presented for determining whether a given task set can be scheduled by EDF or DM to always meet deadlines on a specified number of processors.

CCS Concepts: • **Theory of computation → Scheduling algorithms**; • **Software and its engineering** → *Real-time schedulability*;

Research by Leen Stougie was partially supported by the Netherlands Organisation for Scientific Research (NWO) through the Gravitation Programme Networks (024.002.003).
Earlier versions of this work appeared in the proceedings of the IEEE Real-Time Systems Symposium, San Juan, Puerto Rico, 2012, the EuroMicro Conference on Real-Time Systems, Paris, France, 2013, and the EuroMicro Conference on Real-Time Systems, Lund, Sweden, 2015.
Authors' addresses: V. Bonifaci, Istituto di Analisi dei Sistemi ed Informatica, Consiglio Nazionale delle Ricerche, via dei Taurini 19, 00185 Rome, Italy; email: vincenzo.bonifaci@iasi.cnr.it; A. Wiese, Max Planck Institute for Informatics, Saarbrücken, Germany; email: awiese@mpi-inf.mpg.de; S. K. Baruah, Computer Science Department, University of North Carolina at Chapel Hill, NC, USA; email: baruah@cs.unc.edu; A. Marchetti-Spaccamela, Dipartimento di Ingegneria Informatica Automatica e Gestionale, Sapienza Università di Roma, via Ariosto 25, 00185 Rome, Italy; email: alberto@diag.uniroma1.it; S. Stiller, Institute for Mathematical Optimization, Technische Universität Carolo-Wilhelmina zu Braunschweig, Universitätsplatz 2, 38106 Braunschweig, Germany; email: sebastian.stiller@tu-bs.de; L. Stougie, Centrum Wiskunde & Informatica, P.O. Box 94079, 1090 GB Amsterdam, Netherlands; email: Leen.Stougie@cwi.nl.
Author's current mailing address: A. Wiese, Facultad de Ciencias Fisicas y Matematicas, Universidad de Chile, Beauchef 851 Of. 705 Piso 7, Santiago Centro, Chile; S. K. Baruah, McKelvey School of Engineering, Washington University in St. Louis, Campus Box 1100, 1 Brookings Drive, St. Louis, MO 63130-4899, USA.

2329-4949/2019/06-ART3 $15.00
https://doi.org/10.1145/3322809

Additional Key Words and Phrases: Precedence constraints, multiprocessor platform, directed acyclic graph, conditional control-flow, schedulability test, approximation algorithm

ACM Reference format:
Vincenzo Bonifaci, Andreas Wiese, Sanjoy K. Baruah, Alberto Marchetti-Spaccamela, Sebastian Stiller, and Leen Stougie. 2019. A Generalized Parallel Task Model for Recurrent Real-Time Processes. *ACM Trans. Parallel Comput.* 6, 1, Article 3 (June 2019), 40 pages.
https://doi.org/10.1145/3322809

1 INTRODUCTION

Many real-time systems can be modeled as being composed of a finite number of independent recurrent processes or tasks, each of which generates a potentially infinite sequence of jobs that need to be executed. Since its origins in the late 1960s up until the early 1990s, the discipline of real-time scheduling has primarily dealt with determining how multiple recurrent processes, each with a relatively simple internal structure, can be scheduled upon a shared processor (Liu 1969a, 1969b; Liu and Layland 1973; Dertouzos 1974; Leung and Merrill 1980). More recently, the convergence of two important long-term trends in real-time computing—the increasing prevalence of multiprocessor platforms and increased complexity in the internal structure of individual recurrent processes—has thrown up significant challenges to this traditional perspective of real-time scheduling.

- **Multiprocessors.** As processor manufacturers seek to provide large improvements in performance without corresponding increases in power and energy requirements, scaling trends in processor design have tended to move away from increasing clock frequencies to increasing the number of cores per processor. This trend toward multicore platforms has exposed a significant shortcoming of earlier recurrent task models—they fail to expose the parallelism that may exist within the workload generated by individual recurrent tasks, thereby preventing the exploitation of such parallelism by runtime scheduling mechanisms. This motivates the need for more expressive task models for recurrent processes that allow for the modeling of partial parallelism within individual tasks, as well as for representing precedence dependencies between different parts of a single task. Unfortunately, such issues are not particularly well understood; as Saifullah et al. (2013) have observed, "the growing importance of parallel task models for real-time applications poses new challenges to real-time scheduling theory that has mostly focused on sequential task models."
- **More complex recurrent processes.** Accompanying this trend toward increased parallelism has been a trend toward increased complexity in the functionalities implemented within individual recurrent processes. In particular, the early models for recurrent real-time processes, such as the *Liu and Layland model* (Liu and Layland 1973) and the *three-parameter sporadic tasks model* (Mok 1983) assume that each task models straight-line code; however, all but the simplest real-time code today contains conditional constructs (e.g., *if-then-else*'s) governed by guards whose values cannot always be determined prior to runtime. This trend toward more complex control-flow structures in recurrent real-time processes has indeed been somewhat addressed in the prior real-time scheduling literature (see, e.g., Baruah (1998) and Stigge et al. (2011)) but in the context of models for *uniprocessor* platforms.

The concurrent consideration of both parallelism and conditional execution gives rise to a plethora of important and interesting challenges; this article reports upon our efforts at addressing some of these challenges. We present a parallel task model that we call the *sporadic DAG model* that is designed with the explicit purpose of exposing parallelism that may be present within the workload generated by individual recurrent processes. In this model, a recurrent task is specified as a

directed acyclic graph (DAG), a period, and a relative deadline. Each vertex of the DAG represents a sequential job, while the edges of the DAG represent precedence constraints between these jobs. We additionally propose the *conditional sporadic DAG model* as an enhancement of the sporadic DAG model, to enable the modeling of situations in which control structures within the code being modeled by the task may mean that different activations of the task cause different parts of the code to be executed. We consider real-time workloads that can be modeled as a collection of independent conditional sporadic precedence-constrained tasks that execute upon a platform comprising m identical processors. We assume that the platform is fully *preemptive* and that it allows *global interprocessor migration*, although we assume that each job within a task may execute on at most one processor at each instant of time. We study the behavior of two well known global scheduling policies: global *Earliest Deadline First (EDF)* and global *Deadline Monotonic (DM)* (Leung and Whitehead 1982; Liu and Layland 1973).

2 BACKGROUND CONCEPTS AND RELATED WORK

2.1 Feasibility, Schedulability, and Speedup Bounds

We will see that a single (regular or conditional) sporadic DAG task set may in general generate infinitely many different individual collections of jobs during different runs; a task set $\mathcal{T}$ is said to be *feasible* upon a specified platform if a valid schedule exists on that platform for every collection of jobs that may be generated by the task set $\mathcal{T}$.

Given a scheduling algorithm ALG, a task set is said to be *ALG-schedulable* upon a specified platform if ALG meets all deadlines when scheduling any collection of jobs that may be generated by the task set upon that platform.

An important requirement of hard real-time systems is to guarantee prior to runtime that all deadlines will be met; such guarantees are given by *schedulability tests*. A schedulability test for a given scheduling algorithm ALG is an algorithm that takes as input a description of a task set and the specifications of an execution platform and provides as output an answer to whether the system is ALG-schedulable upon the specified platform. A schedulability test is exact if it correctly identifies all schedulable and unschedulable task sets. It is sufficient if it correctly identifies all unschedulable task systems but may misidentify some schedulable systems as being unschedulable. For any scheduling algorithm to be useful for hard-deadline real-time applications, it must have at least a sufficient schedulability test that can verify that a given job system is schedulable. It has been observed (Baker and Baruah 2007, p. 3-3) that the quality of the scheduling algorithm and the schedulability test are inseparable, since there is no practical difference between a system that is not schedulable and one that cannot be proven to be schedulable.

We say that a scheduling algorithm ALG has a *speedup bound* α, where α is a real number greater or equal than 1 if any task system that is feasible on m unit speed processors is ALG-schedulable on m speed-α processors. With respect to schedulability tests, we define speedup bounds as follows. We say *an ALG-schedulability test has speedup bound* α if any task system that is feasible on m unit speed processors is determined by the test to be ALG-schedulable on m speed-α processors. (Note that such a test may also give a positive answer for systems for which there is no schedule on m unit speed processors but that are ALG-schedulable on m speed-α processors. In this sense, the value α, that is, the *speedup bound* of the test, is a metric for quantifying the quality of the approximation of the test.)

2.2 Results in This Paper

The major contribution of this article is to give bounded-speedup and efficient schedulability tests for sets of unconditional or conditional multiple precedence-constrained tasks, where each task's

precedence constraints are specified by a DAG, and the tasks' deadlines are arbitrary. The main results of the article are to show a $2 - 1/m$ *speedup bound for EDF and a corresponding pseudopolynomial time schedulability test with speedup bound* $2 - 1/m + \epsilon$ (for any $\epsilon > 0$), thus improving and extending previous results; the bound matches the best bound for a sequential sporadic task set (Baruah et al. 2010; Bonifaci et al. 2012), showing that parallel threads and precedence constraints do not influence the effectiveness of EDF. Moreover, in addition to EDF, the analysis is extended to the Deadline Monotonic scheduling algorithm, showing a $3 - 1/m$ *speedup bound and a corresponding pseudopolynomial time schedulability test with speedup bound* $3 - 1/m + \epsilon$ (for any $\epsilon > 0$); also in this case, the speedup bound matches the best bound known for a sequential sporadic task set (Baruah et al. 2010).

Our tests described above have pseudopolynomial time complexity; we complement these pseudopolynomial tests with simpler, polynomial time sufficient conditions to test EDF and DM schedulability. We show how the conditional case can be reduced to the unconditional case by means of an appropriate transformation.

Last but not least, we present an extension to precedence-constrained tasks in which each individual sequential operation may have *a distinct intra-task deadline*, as opposed to a unique deadline per task. This is a significant enhancement of the expressiveness of the model; it allows the specification of more and less stringent timing requirements for different parts of the same task, resulting in a conditional sporadic DAG model that is even more flexible.

2.3 Prior Results

It is known (Ullman 1975) that the preemptive scheduling of a given collection of precedence-constrained jobs on a multiprocessor platform is NP-hard in the strong sense; this intractability result is easily seen to hold for the sporadic DAG model as well.

For the *sequential* sporadic task model (Mok 1983) there exist schedulability tests with speedup factor of $2 - 1/m$ when the scheduling algorithm is Earliest Deadline First (EDF) and $3 - 1/m$ when the scheduling algorithm is Deadline Monotonic (DM) (Baruah et al. 2010; Bonifaci et al. 2012).[1]

Our model also generalizes the *fork-join* model that has been introduced by Lakshmanan et al. (2010) and further generalized in subsequent work (Andersson and de Niz 2012; Nelissen et al. 2012; Saifullah et al. 2013). In the fork-join model, the execution requirement of a task is an alternate sequence of parallel and sequential threads that are represented as sequential and parallel segments; parallel segments need to synchronize before starting the execution of the next sequential segment. Saifullah et al. (2013) analyzed the case of implicit deadlines (i.e., $D_i = T_i$) and all parallel threads with the same worst-case execution requirement and showed a global EDF-schedulability test with speedup 4 and a partitioned DM-schedulability test with speedup 5. The article also extends the above results to the special case of the DAG model where each vertex of the DAG has unit execution time.

Nelissen et al. (2012) considered the same model consisting of a sequence of parallel and sequential threads, with the assumption that relative deadlines of the tasks are not larger than the corresponding periods. The authors showed a speedup bound of 2 for a certain class of algorithms, which includes PD^2, U-EDF, LLREF, and DP-Wrap. Andersson and de Niz (2012) considered a similar model and showed that EDF has a speedup bound of $2 - 1/m$. We remark that the schedulability test provided by Andersson and de Niz is not efficient and no bound on its running time is provided. Independently of our work, Li et al. (2013, 2015) also showed a speedup bound of $2 - 1/m$ for global EDF for arbitrary deadline precedence-constrained tasks, though without a schedulability test. For

[1]Note, however, that in Baruah et al. (2010) and Bonifaci et al. (2012), it is assumed that subsequent jobs generated by the same task *cannot* be parallelized.

the special case of implicit deadline tasks, they additionally provided a linear-time schedulability test with speedup bound $4 - 2/m$ (a similar result is discussed by Saifullah et al. 2014). In subsequent work, Li et al. (2014) focus on implicit deadline precedence-constrained tasks and present a federated scheduling strategy and a corresponding linear-time schedulability test with speedup $2 - 1/m$, as well as linear-time schedulability tests with speedup 2.619 for global EDF and 3.733 for global Rate Monotonic, respectively.

With regards to conditional execution of parallel code. to the best of our knowledge the *multi-DAG model* proposed by Fonseca et al. (2015) represents the first attempt at concurrently modeling both intra-task parallelism and conditional execution in recurrent real-time task systems. The multi-DAG model models each recurrent task as a collection of "execution flows," each of which represents a different flow of control through the code being modeled by the task; each such execution flow is explicitly modeled as a separate DAG.

Although the multi-DAG model does indeed succeed in achieving its goal of generalizing the sporadic DAG model to represent conditional control-flow constructs, this generalization comes at a significant price in terms of computational complexity. As stated above, each possible flow of control (called "execution flow") through the code modeled by an individual task is explicitly represented by a separate DAG, and the *number* of such flows is an important parameter in determining the efficiency of the schedulability analysis and runtime scheduling algorithms proposed in Fonseca et al. (2015). But there may in general be exponentially many different flows through a graph. Consider, for example, code structured like this:

```
if (C1) then {S11} else {S12}
if (C2) then {S21} else {S22}
if (C3) then {S31} else {S32}
. . . . . .
. . . . . .
if (Cn) then {Sn1} else {Sn2}
```

where each `(Ci)` represents a Boolean condition, and each `{Sij}` a block of straight-line code. It is evident that such a code fragment may have 2^n different execution flows through it; hence, requiring explicit enumeration of all execution flows and having the number of such flows be a determinant in the computational complexity of scheduling and schedulability analysis algorithms mean that these algorithms all have exponential worst-case runtime.

We model a recurrent conditional precedence-constrained task by introducing conditional vertices, next to job-vertices. These conditional vertices come in pairs $c, \bar{c}$. Informally speaking, vertex c can be thought of representing a point in the code where a conditional expression is evaluated and, depending upon the outcome of this evaluation, control will subsequently flow along exactly one of several different possible paths in the code. It is required that all these different paths meet again at a common point in the code, represented by the vertex $\bar{c}$. We will define this requirement formally in Section 3. Tasks without conditional vertices will be often referred to as *unconditional* tasks.

2.4 Outline of the Paper

The remainder of this article is organized as follows. In Section 3, we formally define the notation and terminology used in describing our task model. In Section 4, we show that the problems we seek to solve are intractable; hence, we are unlikely to obtain efficient algorithms for solving them exactly. (This fact provides justification for the fact that our algorithms provide approximate solutions, rather than exact ones.) We also show how parallel scheduling anomalies can complicate the analysis even when the task set comprises only one task. In Section 5, we present a general result for work-conserving algorithms (Section 5.2), and we use it to derive the speedup bounds for EDF

(Section 5.3) and DM (Section 5.4). We present and analyze a corresponding pseudopolynomial time EDF- and DM-schedulability test in Section 6 that for arbitrary $\epsilon > 0$ either guarantees that a task set is EDF-schedulable (respectively, DM-schedulable) on m processors of speed $2 - 1/m + \epsilon$ (respectively, $3 - 1/m + \epsilon$) or proves that the task set is infeasible on m processors of unit speed. Section 7 presents a transformation from conditional tasks to unconditional tasks that allows us to apply the tests from Section 6 also in the conditional case. In Section 8, we move on to simpler sufficient schedulability conditions that can easily be tested in polynomial time. Section 9 presents the extension to the case of distinct intra-task deadlines. We conclude with some remarks in Section 10.

3 MODEL AND DEFINITIONS

In the *sporadic DAG model*, the input consists of a *task set* (or *task system*) $\mathcal{T} = (\tau_1, \tau_2, \ldots, \tau_n)$, a list of n *dag-tasks* (or simply *tasks*). A dag-task τ_i $(i = 1, \ldots, n)$ is specified by a tuple $(\mathcal{G}_i, D_i, T_i)$, where $\mathcal{G}_i$ is a vertex-weighted directed acyclic graph and D_i and T_i are positive integers.

The DAG $\mathcal{G}_i$ is specified as $\mathcal{G}_i = (V_i, E_i)$, where V_i is a set of vertices and E_i a set of directed edges between these vertices; it is required that these edges do not form any oriented cycle. We assume that $\mathcal{G}_i$ has exactly one source vertex and one sink vertex. Each set V_i is the disjoint union of three sets $R_i, C_i, \bar{C}_i$ representing three types of vertices. Each $v \in R_i$ denotes a sequential operation (to which we will refer as a "*job*" throughout the article). Each job $v \in R_i$ is characterized by a processing time $e_v \in \mathbb{N}$, also known as *worst-case execution time* (wcet). The *conditional* vertices $(c \in C_i, \bar{c} \in \bar{C}_i)$ come in pairs. They are used to represent an "*if-then-else*" test. Each $c \in C_i$ represents an "*if*," having two edges going out to job-vertices: one to a job-vertex $v_1 \in R_i$ representing the case that the "if" in c holds and one to a vertex $v_2 \in R_i$ otherwise. Thus, from c either the subgraph starting in v_1 is taken, and this subgraph ends in some job-vertex $\bar{v}_1 \in R_i$, or the subgraph starting in v_2 is taken, which ends in some other job-vertex $\bar{v}_2 \in R_i$. From each of $\bar{v}_1$ and $\bar{v}_2$, an edge is going into the "*end if*" conditional vertex $\bar{c} \in \bar{C}_i$. We assume that each of the subgraphs between $v_1, \bar{v}_1$ and between $v_2, \bar{v}_2$ have only one incoming edge (the edge (c, v_k), $k = 1, 2$) and only one outgoing edge (the edge $(\bar{v}_k, \bar{c})$, $k = 1, 2$). More formally,

(1) For each $k \in \{1, 2\}$, let $V'_k \subseteq V_i$ and $E'_k \subseteq E_i$ denote the vertices and edges on paths reachable from v_k that do not include vertex $\bar{c}$. By definition, v_k is the sole source vertex of the DAG $\mathcal{G}'_k = (V'_k, E'_k)$. It must hold that $\bar{v}_k$ is the sole sink vertex of $\mathcal{G}'_k$.
(2) It must hold that $V'_1 \cap V'_2 = \emptyset$. Additionally, with the exception of (c, v_k), there should be no edges in E_i into vertices in V'_k from vertices not in V'_k, for each $k \in \{1, 2\}$. That is, $E_i \cap ((V_i \setminus V'_k) \times V'_k) = \{(c, v_k)\}$ should hold for all k.

In what follows, we will often treat the conditional vertices as job-vertices and refer to all vertices of V_i as job-vertices. The edges represent precedence relations between the jobs: If $(v_1, v_2) \in E_i$, then job v_1 must complete execution before job v_2 can begin execution. In this case, v_1 is called a *predecessor* of v_2. We emphasize the special role of the conditional vertices: After the conditional job $c \in C_i$ has been executed, exactly one of its two successor jobs becomes eligible for execution, and $\bar{c} \in \bar{C}_i$ can be executed only after exactly one of its two predecessor jobs has completed.

The sub-DAG between a pair $c \in C_i, \bar{c} \in \bar{C}_i$ is called a *conditional construct*. Conditional constructs can be nested. We will use the term *inner-most conditional construct* to denote a conditional construct that does not contain any other one.

Notice that in the construction of the DAG, we assumed that either outcome of a conditional test leads to a subgraph that starts and ends with a single job. We can always ensure this by introducing, if necessary, dummy job-vertices with processing time 0. Similarly, the assumption that $\mathcal{G}_i$ has a

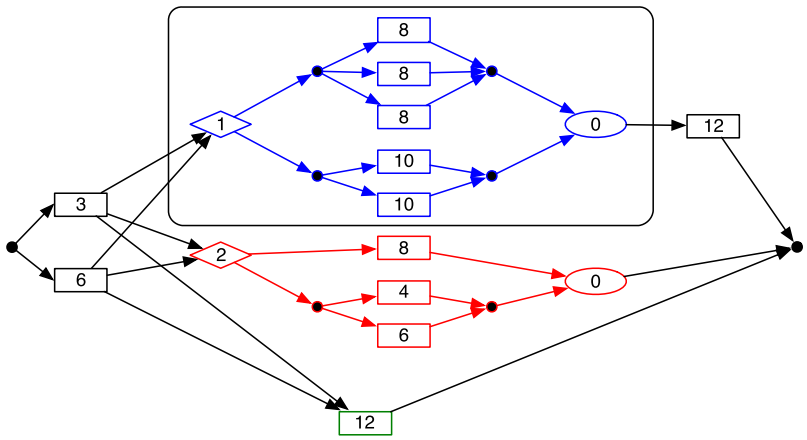

Fig. 1. The DAG of an example conditional sporadic DAG task. Vertices denote jobs; the numbers within vertices denote the wcets of the jobs. Small solid circles denote jobs with wcet equal to zero. Diamonds and ovals denote conditional start and end vertices, respectively, and rectangles denote non-conditional vertices. (The large rectangle encloses a single conditional construct in the DAG that will be referenced later in this document.)

single source and a single sink is without loss of generality. Also, similarly, the out-degree and in-degree of 2 of c and $\bar{c}$, respectively, is without loss of generality, by using dummy conditional vertices. If desired, then we can also associate a processing time $e_c \in \mathbb{N}$ to a start conditional vertex c, meaning that the evaluation of the logical condition is itself a sequential operation requiring a wcet of e_c; this is equivalent to the insertion of a regular job vertex with wcet equal to e_c in front of the conditional vertex c. End conditional vertices typically have an associated processing time equal to zero, but a nonzero processing time could be similarly accomodated.

An example conditional sporadic dag-task is depicted in Figure 1. Small solid circles denote "dummy" vertices, which correspond to jobs with wcet equal to zero. Diamond and oval vertices denote start and end conditional vertices, respectively; there are two pairs of conditional vertices in this DAG, each with branching factor equal to two. (For those reading this on a color medium, the upper conditional construct is represented in blue; the lower conditional construct in red.) The semantics of this dag-task are as follows. Whenever a dag-job is released, the dummy source vertex has zero execution requirement and therefore immediately completes execution. Two vertices, with wcets 3 and 6, respectively, both become eligible for execution. Once both these jobs have completed, three jobs become eligible simultaneously.

- —A conditional expression with a wcet of 1 is evaluated; depending upon the outcome of this evaluation, either three jobs each with wcet equal to 8, or two jobs each with wcet equal to 10, are executed. After these jobs complete, a single job with wcet equal to 12 is executed.
- —Another conditional expression that has a wcet of 2 is evaluated. Depending upon the outcome of this evaluation, either a single job with a wcet equal to 8, or two jobs with wcet equal to 4 and 6, respectively, are executed.
- —A single job with wcet equal to 12 is executed.

The positive integer T_i is called the *period* of task τ_i. A *release* or arrival of a *dag-job* of task τ_i at time-instant t means that all $|V_i|$ jobs $v \in V_i$ are released at time-instant t; t is called the *release date* of both the dag-job and the jobs that compose it. The period denotes the minimum amount of time that must elapse between the release of successive dag-jobs of task τ_i: If a dag-job is released

at t, then the next dag-job of of task τ_i cannot be released prior to time-instant $t + T_i$. We also assume all release dates to be integer. We say a job v is *available* at time t if all the predecessor jobs of v (if any) have completed execution at time t; v itself has not been completed at time t and t is greater than or equal to the release date of v.

The positive integer D_i is called the (*relative*) *deadline*. If a dag-job is released at time-instant t, then all $|V_i|$ jobs that constitute it must complete execution by time-instant $t + D_i$ (the *absolute deadline* of those jobs).

Remark 3.1. If $D_i > T_i$, then task τ_i may release a dag-job prior to the completion of its previously-released dag-jobs. We do *not* require that all jobs of a dag-job complete execution before jobs of the next dag-job can start executing.

Remark 3.2. We assume that each job requires an integer number of units of execution time (less than or equal to its wcet). Note, however, that even though we assume the execution times to be integers, when analyzing algorithms with increased speed (as we will, for example, do for EDF with speed $2 - 1/m$ in Section 5), a job could be completed at a non-integral point in time, even if it is never preempted. Therefore, in the analysis it is possible for jobs to be started or preempted at fractional time instants.

Some additional notation and terminology:

- A *chain* in $\mathcal{G}_i$ is a sequence of vertices $v_1, v_2, \ldots, v_k \in V_i$ such that (v_j, v_{j+1}) is an edge in $\mathcal{G}_i$, $1 \leq j < k$. The *length* of this chain is defined to be the sum of the wcets of all its vertices: $\sum_{j=1}^{k} e_{v_j}$.
- We denote by $\text{len}(\mathcal{G}_i)$ the length of the longest chain in $\mathcal{G}_i$. Notice that for this definition the existence of conditional vertices is irrelevant. It is easy to see that $\text{len}(\mathcal{G}_i)$ can be computed in time linear in the number of vertices and the number of edges in the acyclic graph $\mathcal{G}_i$ by first obtaining a topological order of the vertices of the graph and then running a straightforward dynamic program.
- We define $\text{vol}(\mathcal{G}_i)$ as the maximum total wcet of a single dag-job that can be generated by task τ_i. Contrary to $\text{len}(\mathcal{G}_i)$, in this notion conditional constructs do play a crucial role. In every possible realization of the DAG either of the two pathways in every conditional construct is taken. Clearly, for computing $\text{vol}(\mathcal{G}_i)$ the largest of the two, of every conditional construct, has to be taken into account. We observe that a polynomial time algorithm for computing the volume of a conditional dag-task can be obtained as a consequence of the main result of Section 7.

For our example task τ_1 of Figure 1, a dag-job has maximum total wcet if the upper branch is taken for the upper conditional and the lower branch for the lower conditional, meaning that $\text{vol}(\mathcal{G}) = 70$; for this graph, we also have $\text{len}(\mathcal{G}) = 29$.

Throughout the article, we denote the length of a time interval I by $|I|$. We summarize in Table 1 the most common notations used throughout the article.

4 HARDNESS OF THE DAG MODEL

4.1 Computational Complexity

As mentioned in the Introduction, the multiprocessor scheduling of sporadic dag-tasks is an NP-hard problem. Indeed, even for a single unconditional dag-task for which $D = T$, determining feasibility is easily seen to be equivalent to the makespan minimization problem for preemptive scheduling of a set of precedence constrained jobs on identical processors, or $P|prec, pmtn|C_{\max}$, using scheduling notation (Graham et al. 1979) (diverging, only in this subsection, a bit from notation

Table 1. Notation

n	number of tasks
m	number of processors
α, β	speedup factors
ϵ	error factor
$\mathcal{T}$	task set
τ_i	ith task
D_i	relative deadline of τ_i
T_i	period of τ_i
$\mathcal{G}_i$	DAG of τ_i
V_i	vertex set of $\mathcal{G}_i$
E_i	edge set of $\mathcal{G}_i$
v, v'	job-vertices
e_v	wcet of v
R_i	regular vertex set
C_i	start conditional vertex set
$\bar{C}_i$	end conditional vertex set
$\text{len}(\mathcal{G}_i)$	length of $\mathcal{G}_i$
$\text{vol}(\mathcal{G}_i)$	volume of $\mathcal{G}_i$
t, t'	time instants
I, I'	time intervals
$\|I\|$	length of I

J, J'	job collections
j, j'	jobs
r_j	release date of j
d_j	absolute deadline of j
e_j	(unknown) execution time of j
S_∞	idealized schedule
t^*	critical safe instant
t'_f	deadline miss instant
x	fully-busy time in I'
y	non-fully-busy time in I'
Y	non-fully-busy instants set
$\text{gen}(\mathcal{T})$	set of job collections that can be generated by $\mathcal{T}$
$\text{work}^J(I)$	work of J done by S_∞ due in I
$\text{work}_i(t)$	worst-case workload of τ_i due in an interval of length t
$\hat{w}_i(t)$	approximation of $\text{work}_i(t)$
$\lambda_\mathcal{T}$	workload density of $\mathcal{T}$
$\text{rdem}^J(t)$	remaining demand of J in S_∞ after t time units
$\text{rdem}_i(t)$	worst-case remaining demand of τ_i in S_∞ after t time units

introduced in the previous section). Therefore, the problem is NP-hard in the strong sense (Ullman 1975).

Proposition 4.1. *The problem of determining the feasibility of a set of dag-tasks is* NP*-hard in the strong sense, even when* $n = 1$ *and* $D = T$.

The above hardness result can be strengthened in some cases. If job preemptions are only allowed at integer multiples of the processor's clock cycle, then the problem $P|prec, pmtn|C_{\max}$ is equivalent to $P|prec, p_j = 1|C_{\max}$, for which a result of Lenstra and Rinnooy Kan (1978) implies that determining the feasibility of a dag-task set is NP-hard in the strong sense even when allowing a processor speedup of $4/3 - \epsilon$, for any $\epsilon > 0$.

We also remark that, when the number of processors m is not part of the input (that is, it is a constant), the complexity of the feasibility problem for a single unconditional task with $D = T$ is related to that of a long-standing open problem, known in scheduling notation as $Pm|prec, p_j = 1|C_{\max}$, which appears as open problem "OPEN8" from the original list of Garey and Johnson (1979) and is still open.

4.2 Parallel Scheduling Anomalies

Since a dag-task may legally generate infinitely many different collections of dag-jobs, it would be helpful to identify a single collection as representing the "worst-case" collection, in the sense that if this worst-case collection is feasible (respectively, ALG-schedulable by some algorithm ALG), then all legal collections are also feasible (respectively, ALG-schedulable). One reasonable candidate for the status of such worst-case behavior is the *synchronous arrival sequence*, in which

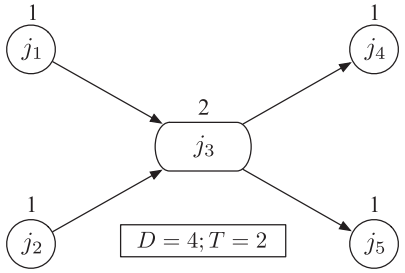

Fig. 2. An example dag-task. The number above each vertex denotes the wcet of the corresponding job.

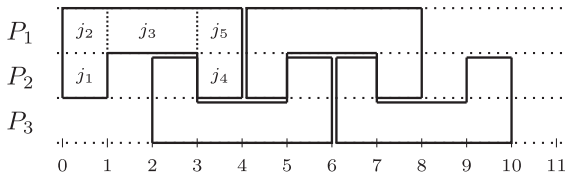

Fig. 3.

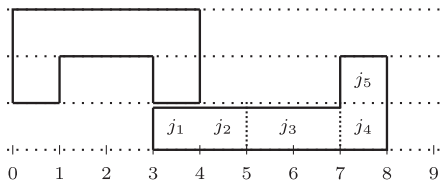

Fig. 4. A parallel scheduling anomaly.

successive dag-jobs arrive as soon as they are able to do so, i.e., exactly T time units apart. (Intuitively, the synchronous arrival sequence is a reasonable candidate, since it maximizes the amount of execution that needs to be completed over a given interval.) However, the synchronous arrival sequence does not consider the parallelism between different jobs within the same dag-job; it turns out that, as a consequence, the synchronous arrival sequence need not in fact correspond to the worst-case behavior of a dag-task set, even when the task set consists of a single unconditional task.

Proposition 4.2. *A dag-task set $\mathcal{T}$ might be infeasible on m processors even when $n = 1$ and the synchronous arrival sequence of $\mathcal{T}$ is feasible on m processors.*

Proof. Consider the unconditional dag-task with $T = 2$, $D = 4$, consisting of five jobs, depicted in Figure 2. Suppose we wish to schedule this task on three processors P_1, P_2, P_3. We present below (Figure 3) a schedule in Gantt-chart like notation for the synchronous arrival sequence.

As the schedule demonstrates, the synchronous arrival sequence is feasible. Jobs j_1 and j_2 first execute in parallel for one time unit. This is followed by job j_3 executing sequentially for two time units, after which jobs j_4 and j_5 execute in parallel for one time unit. All jobs complete execution within four time units of the release of the dag-job to which they belong.

However, if one dag-job is released at time 0 and the second dag-job is released at time 3 (instead of 2), then one of the two dag-jobs cannot complete on time.

The problem, as illustrated in Figure 4 the schedule above, is that if the first dag-job completes by its deadline, then the second dag-job cannot exploit the parallelism of j_1 and j_2 by executing

them in parallel. This requires that these jobs of the second dag-job execute sequentially one after the other, thereby causing the dag-job to miss its deadline (which is at time-instant 7). □

5 SPEEDUP BOUNDS FOR COLLECTIONS OF JOBS

This section considers what we call a *consistent collection J* of jobs. The set of jobs in a sequence of dag-jobs generated by a dag-task set $\mathcal{T}$ will be an example of a consistent collection of jobs. Arguing about job collections instead of DAG task sets makes the results somewhat more general and—more importantly—easier to present. We now define the consistent collection model.

Definition 5.1. A *collection of jobs J* is a sequence of jobs that are revealed online over time, i.e., a job $j \in J$ becomes known upon the release of j. Each job $j \in J$ is characterized by a release date $r_j \in \mathbb{N} \cup \{0\}$, an absolute deadline $d_j \in \mathbb{N}$, an unknown execution time $e_j \in \mathbb{N}$, and a set of *predecessor* jobs J_j, which are the jobs that have to be finished before j can become available. The actual execution time e_j of a job is discovered by the scheduler only when the job signals completion.

We call such a collection of jobs J a *consistent* collection of jobs if we also have, for every predecessor job j of job k, that $r_j = r_k$ and $d_j \leq d_k$. Observe that every collection of jobs generated by a dag-task set is consistent, since all jobs that constitute a given dag-job have identical release date and, if $d_j > d_k$ with j preceding k, then we can without loss of generality reset d_j to d_k without affecting schedulability. Recall that a job j is *available at time t* if $t \geq r_j$ and all predecessors of j have been completed, while j is not yet completed.

5.1 The Idealized Schedule

Given a collection of jobs J, suppose that infinitely many (or, say, cardinality of J) processors of unit speed were available. In this case, the following scheduling algorithm would be optimal: Allocate one processor to each job and schedule each job as soon as it becomes available. Denote by S_∞ the resulting schedule; it is easy to see that the following claims hold:

- (K1) S_∞ starts and ends processing jobs at integral time instants.
- (K2) S_∞ dominates all valid schedules of J, in the following sense: At any point in time and for any job, S_∞ has processed at least as much of that job as any valid schedule of J did upon a platform of m unit speed processors.

Since, usually, the number of available processors will be smaller than the number of jobs, the schedule S_∞ cannot be implemented in general. Still, in light of property (K2) above, S_∞ is useful to establish comparisons with valid schedules.

Example 5.2. Consider the dag-task of Figure 2. Its synchronous arrival sequence is scheduled by S_∞ as depicted in Figure 5.

In the following subsections, we analyze EDF and DM by comparing them to S_∞. To this end, we need some additional definitions.

Definition 5.3. Let ALG be a scheduling algorithm and $\alpha \geq 1$. An *α-counterexample* for ALG is a consistent job collection J such that J is correctly scheduled by S_∞ on unit speed processors but incorrectly scheduled by ALG on m speed-α processors (that is, S_∞ completes each job of J by its deadline, while ALG does not). A *minimal α-counterexample* is an α-counterexample J such that no $J' \subsetneq J$ is an α-counterexample.

Definition 5.4. Let ALG be a scheduling algorithm and J an α-counterexample for ALG, for some $\alpha \geq 1$. A *safe instant* is a time t such that ALG has processed at least the same amount of every job

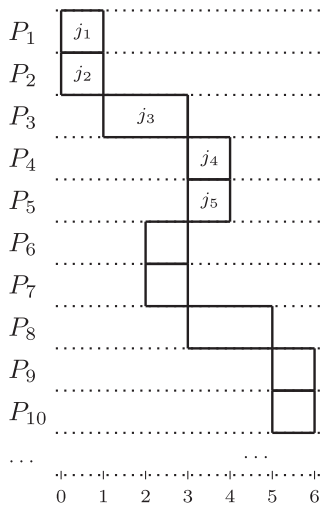

Fig. 5. The idealized schedule S_∞ for the synchronous arrival sequence of the DAG task in Figure 2 (only a prefix of the schedule is shown).

during $[0, t]$ as S_∞ did during $[0, t]$. The *critical safe instant* t^* is the largest safe instant that is not larger than the release date of the first job on which ALG fails.

Note that the definition is well posed: The critical safe instant of ALG exists since 0 is, trivially, a safe instant.

Example 5.5. Consider again the parallel scheduling anomaly considered in Section 4.2. The job collection shown in Figure 4 is a 1-counterexample for EDF. However, it is not minimal, since removing the second occurrence of either j_5 or j_4 (but not both) creates a smaller 1-counterexample. The critical safe instant t^* here is 3.

5.2 Analysis of Work-Conserving Algorithms

A *work-conserving algorithm* is a scheduling algorithm that never leaves processors idle while there are available jobs. We give a general lemma concerning work-conserving algorithms. From this lemma, speedup bounds for EDF and DM will directly follow.

Lemma 5.6. *Let ALG be a work-conserving algorithm and $\alpha, \beta \geq 1$ be such that whenever ALG admits a minimal α-counterexample J^*, there exist intervals $I = [t^*, t_f]$, $I' = [t^*, t'_f]$ such that:*

- *t^* is the critical safe instant;*
- *t'_f is the time at which ALG misses a deadline;*
- *$|I|/\beta \leq |I'| \leq |I|$;*
- *during I', ALG only executes jobs with deadline in I.*

Then for any consistent job collection J and any integer $m \geq 1$, at least one of the following holds:

(1) *all jobs in J are completed within their deadlines under ALG on m processors of speed α, or*
(2) *J is not schedulable by S_∞ (with unit speed), or*
(3) *there is an interval I such that any valid speed-1 schedule for J on m processors must process more than $(\alpha m - m + 1) \cdot |I|/\beta$ units of work within I.*

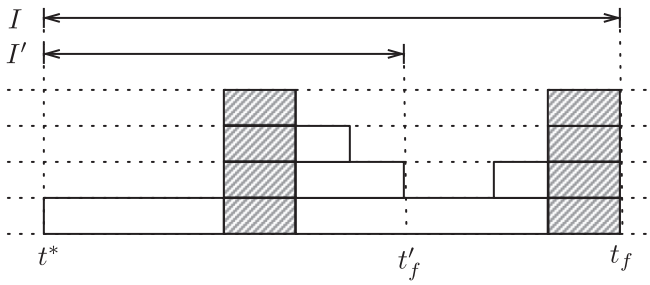

Fig. 6. Illustration of the analysis in Lemma 5.6. The overall time span of the shaded intervals in I' is x. The overall time span of the non-shaded intervals in I' is $y = |I'| - x$. The total work processed by ALG during I' is at least $\alpha m x + \alpha y$.

Proof. Let J be a consistent collection of jobs and assume that neither (1) nor (2) hold; that is, under ALG on m speed-α processors, some job $j \in J$ fails to meet its deadline d_j, while J is feasible with unit speed on a sufficiently large number of processors. In other words, J is a α-counterexample for ALG.

Consider a minimal α-counterexample $J^* \subseteq J$. By the assumption, there must exist intervals I, I' with the properties required in the hypothesis. We now claim that, within I', ALG processes more than $(\alpha m - m + 1)|I'|$ units of work. This claim gives the lemma due to the following reasoning: If ALG processes more than $(\alpha m - m + 1)|I'|$ units of work within I', then the idealized scheduler S_∞ processes at least the same amount of work during I, because, by the assumptions, t^* is itself a safe instant and, again by assumption, ALG is only processing jobs with deadline in I. Hence, *every* valid schedule for J^* (by property (K2) of S_∞) has to process more than

$$(\alpha m - m + 1)|I'| \geq (\alpha m - m + 1)|I|/\beta$$

units of work during I. Therefore, the same must be true for any valid schedule for J (since $J^* \subseteq J$).

To prove the claim on the amount of work done by ALG during I', let x denote the total length of intervals within I', where in ALG's schedule all m processors are busy, and let $y = |I'| - x$. Observe that in ALG's schedule at least one processor is always busy during I', because ALG is work conserving (this is illustrated in Figure 6).

We now claim that $\alpha y < |I'|$. First assume that this is not the case and $\alpha y \geq |I'|$. Denote by $Y_1, \ldots, Y_k \subseteq I'$ the subintervals of I' where not all processors are busy, and let $Y = Y_1 \cup \ldots \cup Y_k$. We define t_0 to be the least time $t \geq \lceil t^* \rceil$ such that

$$\alpha \cdot \Big|[t^*, t] \cap Y\Big| \geq \lceil t^* \rceil - t^*.$$

By property (K1) of S_∞, during all time instants within $[t^*, t_0) \cap Y$ all jobs are available for ALG that are scheduled by S_∞ during $[t^*, \lceil t^* \rceil)$. Since during all these instants ALG, being a work-conserving algorithm, does not use all processors and runs the processors with speed α, and by time t_0 it has processed at least as much of every job as S_∞ has by time $\lceil t^* \rceil$.

We define instants t_h, $h = 1, \ldots, t'_f - \lceil t^* \rceil$ using an analogous pattern: t_h is defined as the least $t \geq \lceil t^* \rceil + h$ such that

$$\alpha \cdot \Big|[t^*, t] \cap Y\Big| \geq \lceil t^* \rceil - t^* + h. \tag{1}$$

Note that we are still assuming that $\alpha y \geq |I'|$, and hence $h^* = t'_f - \lceil t^* \rceil$ is well defined. In fact, $t_{h^*} = t'_f$, since t'_f satisfies Equation (1) for $h = h^*$:

$$\alpha y = \alpha \cdot \left| [t^*, t'_f] \cap Y \right| \geq \lceil t^* \rceil - t^* + t'_f - \lceil t^* \rceil = |I'|.$$

We prove by induction that up to time t_h ALG has processed as much of every job as S_∞ has by time $\lceil t^* \rceil + h$. The case $h = 0$ was proven above. Now suppose that the claim is true for some $h \geq 0$. Then, again by property (K1), at each time instant during $[t_h, t_{h+1}) \cap Y$ all jobs are available for ALG that S_∞ works on during $[\lceil t^* \rceil + h, \lceil t^* \rceil + h + 1)$. Since during all these instants ALG does not use all processors and runs the processors with speed α, by time t_{h+1} it has processed at least as much of every job as S_∞ by time $\lceil t^* \rceil + h + 1$. By induction, the claim is true for $h^* = t'_f - \lceil t^* \rceil$, and hence at time $t_{h^*} = t'_f$ ALG has processed as much of every job as S_∞. This yields a contradiction, since we assumed that S_∞ constructs a valid schedule while ALG does not.

Therefore, we must have $\alpha y < |I'|$. Then during I' ALG processes at least

$$\begin{aligned} \alpha m \cdot x + \alpha y &= \alpha m(|I'| - y) + \alpha y \\ &= \alpha m |I'| - \alpha m y + \alpha y \\ &= \alpha m |I'| - \alpha(m-1)y \\ &> (\alpha m - m + 1)|I'| \end{aligned}$$

units of work, and by construction of I', any valid schedule has to process during the interval I all work that ALG processes during I'. □

Theorem 5.7. *Let ALG be a work-conserving algorithm satisfying the hypothesis of Lemma 5.6 with $\alpha \geq 1 + \beta - 1/m$. Then any consistent job collection J that is feasible on m processors of unit speed is ALG-schedulable on m processors of speed α.*

Proof. Let J be a consistent job collection that is feasible on m processors of unit speed. Since ALG satisfies the hypothesis of Lemma 5.6, we know that at least one of the following holds:

(1) all jobs in J are completed within their deadline under ALG on m processors of speed α, or
(2) J is not schedulable by S_∞ (with unit speed), or
(3) there is an interval I such that any valid schedule for J must process more than $(\alpha m - m + 1)|I|/\beta$ units of work within I.

Since J is assumed feasible on m processors of unit speed, it must also be schedulable by S_∞, therefore case (2) is excluded.

Also, since J is feasible on m processors of unit speed, J admits a valid schedule that processes during any interval I at most $m \cdot |I|$ units of work. This excludes case (3), because

$$(\alpha m - m + 1)|I|/\beta \geq m|I|.$$

Hence case (1) must hold. □

Since every collection of jobs generated by a dag-task set is consistent, we obtain the following corollary.

Corollary 5.8. *Let ALG be a work-conserving algorithm satisfying the hypothesis of Lemma 5.6 with $\alpha \geq 1 + \beta - 1/m$. Then any dag-task set that is feasible on m processors of unit speed is ALG-schedulable on m processors of speed α.*

5.3 Analysis of EDF

At any time, the EDF scheduler processes the m jobs with minimum deadline that are currently available (breaking ties arbitrarily).

We show that EDF satisfies the hypothesis of Lemma 5.6 with $\beta = 1$, $\alpha = 2 - 1/m$.

Lemma 5.9. *Let J^* be a minimal α-counterexample for EDF, where $\alpha = 2 - 1/m$. Then there is an interval $I = I' = [t^*, t'_f]$ such that:*

—t^* is the critical safe instant;
—t'_f is time at which EDF misses a deadline;
—during I, EDF only executes jobs with deadline in I.

Proof. Let $j \in J^*$ be the job for which EDF fails to meet the deadline. We let $t'_f = d_j$. By the minimality of J^*, J^* does not contain any job j' with deadline larger than d_j; otherwise, such a job could be removed from J^* to obtain a smaller α-counterexample (we are using the fact that J^* is consistent: If $d_{j'} > d_j$, then j' cannot precede j, nor any indirect predecessor of j). Therefore, all the jobs that EDF processes during I must have their deadline in I. □

Theorem 5.10. *Any consistent collection of jobs that is feasible on m processors of unit speed is EDF-schedulable on m processors of speed $2 - 1/m$.*

Proof. Follows by Lemma 5.9 and Theorem 5.7. □

Corollary 5.11. *Any dag-task set that is feasible on m processors of unit speed is EDF-schedulable on m processors of speed $2 - 1/m$.*

The above bound is tight: Examples are known, even without precedence constraints, of feasible collections of jobs that are not EDF-schedulable unless EDF's speedup is at least $2 - 1/m$ (Phillips et al. 2002).

5.4 Analysis of DM

Recall that the relative deadline of a job j is the difference $(d_j - r_j)$ between its absolute deadline and release date. At any time, the DM scheduler processes the m jobs with minimum relative deadline that are currently available (breaking ties arbitrarily).

We can show that DM satisfies the hypothesis of Lemma 5.6 with $\beta = 2$, $\alpha = 3 - 1/m$.

Lemma 5.12. *Let J^* be a minimal α-counterexample for DM when run at speed $\alpha = 3 - 1/m$. Then there are intervals $I = [t^*, t_f]$, $I' = [t^*, t'_f]$ such that:*

—t^* is the critical safe instant;
—t'_f is the time at which DM misses a deadline;
—$|I|/2 \le |I'| \le |I|$;
—during I', DM only executes jobs with deadline in I.

Proof. Let $j \in J^*$ be the job for which DM fails to meet the deadline. We let $t_f = 2d_j - r_j$ and $t'_f = d_j$. By definition of critical safe instant we have $t^* \le r_j$. Hence,

$$\frac{|I'|}{|I|} = \frac{d_j - t^*}{2d_j - r_j - t^*} \ge \frac{d_j - r_j}{2d_j - 2r_j} = \frac{1}{2}.$$

The other inequality, $|I'| \le |I|$, follows from $t'_f \le t_f$.

Finally, by the minimality of J^*, J^* does not contain any job j' with deadline $d_{j'}$ larger than $2d_j - r_j$. In fact, assume there was such a job j'. If the relative deadline of j' is at most $d_j - r_j$,

then j' is released after d_j and therefore its removal would yield a smaller counterexample; if the relative deadline of j' is larger than $d_j - r_j$, then j' never interferes with the execution of job j (by definition of DM) and again by removing j' from J^* we would obtain a smaller counterexample. Therefore, all the jobs that DM processes during I' must have their deadline in I. □

Theorem 5.13. *Any consistent collection of jobs that is feasible on m processors of unit speed is DM-schedulable on m processors of speed $3 - 1/m$.*

Proof. By Lemma 5.12 and Theorem 5.7. □

Corollary 5.14. *Any dag-task set that is feasible on m processors of unit speed is DM-schedulable on m processors of speed $3 - 1/m$.*

6 PSEUDOPOLYNOMIAL TIME TESTS WITH BOUNDED SPEEDUP

In this section, we present a pseudopolynomial time test for both EDF- and DM-schedulability that is based on a characterization of the amount of work that a feasible dag-task set requires. In fact, after starting with basic preliminaries that hold for any dag-task, in this section we will derive the tests for unconditional dag-tasks. In the next section, we will then extend the tests to conditional dag-tasks, essentially by transforming conditional dag-tasks into unconditional dag-tasks that are equivalent in terms of relevant parameters (length, volume, and workload density).

6.1 Derivation of the Schedulability Test(s)

In the following, we present a pseudopolynomial time test for both EDF- and DM-schedulability that is based on a characterization of the amount of work that a feasible dag-task set requires.

Recall the definition of S_∞ from Section 5. Suppose we are given a set $\mathcal{T}$ of dag-tasks. Lemmata 5.6 and 5.9 imply that, to assert that EDF feasibly schedules any consistent job collection J that can be generated by $\mathcal{T}$ when given speed α, it suffices to ensure that for any such job collection J,

- **Condition 1:** J is S_∞-schedulable, and
- **Condition 2:** there is no interval I during which every valid schedule for J must finish more than $(\alpha m - m + 1) \cdot |I|$ units of work.

However, if any of the two conditions fails (with $\alpha \geq 2 - 1/m$) then the task set is infeasible on m unit speed processors. Using Lemmata 5.6 and 5.12 allows a similar reasoning for DM (with $\alpha \geq 3 - 1/m$). Therefore, a natural schedulability test will check both conditions and return "schedulable" if and only if both of them are satisfied.

Remark 6.1. Observe that both conditions are monotone in the execution times of the job collection. That is, if they are satisfied by a collection of jobs with some execution times, they are also satisfied by a similar collection of jobs with decreased execution times. This allows us to focus on the wcets of the jobs when verifying the conditions.

Verifying Condition 1. It is easy to check whether every job collection that can be generated by $\mathcal{T}$ is correctly scheduled by S_∞: This is the case if and only if $\text{len}(\mathcal{G}_i) \leq D_i$ for all $i = 1, \ldots, n$. This can be tested in linear time by computing each $\text{len}(\mathcal{G}_i)$ as discussed in Section 3.

Verifying Condition 2. For the remainder of this section, we can focus on verifying the second condition. For a collection of jobs J and an interval I, we denote by $\text{work}^J(I)$ the amount of work done by S_∞ during I on the jobs in J whose deadlines are in I. The motivation for this quantity is that any valid schedule with unit speed machines has to execute at least $\text{work}^J(I)$ units of work during I.

Example 6.2. Consider again the collection of jobs J that was analyzed in Figure 4. Let $I = [3, 7]$; then the deadlines of all jobs are in I. The amount of work done by S_∞ during I is 8: 2 for the last two jobs of the first dag-job, plus 6 for the jobs of the second dag-job. Therefore, $\text{work}^J(I) = 8$.

Definition 6.3. Given a dag-task set $\mathcal{T}$, let $\text{gen}(\mathcal{T})$ be the set of job collections that may be generated by $\mathcal{T}$ and define

$$\text{work}_{\mathcal{T}}(t) = \sup_{J \in \text{gen}(\mathcal{T})} \sup_{t_0 \geq 0} \text{work}^J([t_0, t_0 + t]).$$

$$\lambda_{\mathcal{T}} = \sup_{t \in \mathbb{N}} \frac{\text{work}_{\mathcal{T}}(t)}{t}.$$

Intuitively, the quantity $\lambda_{\mathcal{T}}$ denotes the maximum "workload density" that an interval can have. In particular, if $\lambda_{\mathcal{T}} > m$, then the system is infeasible, since there is a job sequence for $\mathcal{T}$ and an interval I during which more than $m \cdot |I|$ units of work have to be finished by any schedule.

We want to compute the maximum workload density, to test Condition 2; that is, to test whether or not $\lambda_{\mathcal{T}} \leq m$. We cannot afford to compute $\lambda_{\mathcal{T}}$ with perfect precision: That would be an NP-hard task already in the case of sequential tasks (Eisenbrand and Rothvoss 2010). However, computing the workload density up to an ϵ error factor, for some small $\epsilon > 0$, is sufficient, as shown in the next lemma. The lemma states that with a certain extra speedup EDF and DM are still feasible if the workload density is slightly higher than m. So, if we can at least distinguish whether $\lambda_{\mathcal{T}}$ is greater than m, or less than or equal to slightly more than m, then we can either conclude the task set is EDF-schedulable (or DM-schedulable) with an appropriate speedup or that for some job sequence in $\text{gen}(\mathcal{T})$, no valid unit speed schedule exists.

Lemma 6.4. *Let $\mathcal{T}$ be a dag-task set. Let $\epsilon \geq 0$ and suppose that $\lambda_{\mathcal{T}} \leq (1 + \epsilon)m$ for any $t \in \mathbb{N}$ and that $\mathcal{T}$ is feasible on a sufficiently large number of unit-speed processors. Then $\mathcal{T}$ is EDF-schedulable on m processors of speed $2 - 1/m + \epsilon$ and DM-schedulable on m processors of speed $3 - 1/m + 2\epsilon$.*

Proof. We give the proof for EDF; the one for DM follows by exactly the same arguments and is therefore omitted. Suppose that EDF fails on some job collection $J \in \text{gen}(\mathcal{T})$ when running at speed $\alpha = 2 - 1/m + \epsilon$; i.e., J is an α-counterexample. Then by Lemmata 5.6 and 5.9 we can choose $\beta = 1$ and conclude that there is an interval I in which any valid schedule must finish more than

$$(\alpha m - m + 1) \cdot |I| = (2m - 1 + \epsilon m - m + 1)|I| = (1 + \epsilon)m|I|$$

units of work. This contradicts that $\lambda_{\mathcal{T}} \leq (1 + \epsilon)m$. □

Therefore, to approximately test the feasibility of $\mathcal{T}$, it suffices to estimate $\lambda_{\mathcal{T}}$. We summarize this in the following lemma.

Lemma 6.5. *Let $\epsilon \geq 0$ and $\hat{\lambda}_{\mathcal{T}}$ be such that $\lambda_{\mathcal{T}}/(1 + \epsilon) \leq \hat{\lambda}_{\mathcal{T}} \leq \lambda_{\mathcal{T}}$. Assume that $\mathcal{T}$ is feasible on a sufficiently large number of unit-speed processors. Then*

(1) *if $\hat{\lambda}_{\mathcal{T}} > m$, $\mathcal{T}$ is infeasible on m unit speed processors;*
(2) *if $\hat{\lambda}_{\mathcal{T}} \leq m$, $\mathcal{T}$ is EDF-schedulable on m speed-$(2 - 1/m + \epsilon)$ processors and DM-schedulable on m speed-$(3 - 1/m + \epsilon)$ processors.*

Proof. In case (1), we have $\lambda_{\mathcal{T}} \geq \hat{\lambda}_{\mathcal{T}} > m$. Thus, there is a job collection $J \in \text{gen}(\mathcal{T})$ and an interval I such that $\text{work}^J(I) > m|I|$, and hence $\mathcal{T}$ is not feasible on m unit speed machines.

In case (2), we have $\lambda_{\mathcal{T}} \leq (1 + \epsilon)\hat{\lambda}_{\mathcal{T}} \leq (1 + \epsilon)m$. Thus, Lemma 6.4 yields the claim. □

Given a dag-task set $\mathcal{T}$, a $(1+\epsilon)$-*approximation algorithm for* $\lambda_{\mathcal{T}}$ is an algorithm computing a value $\hat{\lambda}_{\mathcal{T}}$ that fulfills

$$\lambda_{\mathcal{T}}/(1+\epsilon) \leq \hat{\lambda}_{\mathcal{T}} \leq \lambda_{\mathcal{T}}.$$

In other words, it computes a value not larger than the true maximum work density but also not much smaller than it. Note that these are exactly the conditions required in Lemma 6.5. Thus, we can reformulate the lemma:

Corollary 6.6. *Let $\epsilon \geq 0$. A $(1+\epsilon)$-approximation algorithm for $\lambda_{\mathcal{T}}$ yields an EDF-schedulability test for $\mathcal{T}$ with speedup $2 - 1/m + \epsilon$ and a DM-schedulability test for $\mathcal{T}$ with speedup $3 - 1/m + \epsilon$.*

Corollary 6.7. *Let $\mathcal{T} = (\tau_1, \tau_2, \ldots, \tau_n)$ be a dag-task set such that:*

(1) $\text{len}(\mathcal{G}_i) \leq D_i$ for each $i = 1, 2, \ldots, n$;
(2) $\lambda_{\mathcal{T}} \leq m$.

Then $\mathcal{T}$ is EDF-schedulable with speed $2 - 1/m$ on m processors and DM-schedulable with speed $3 - 1/m$ on m processors.

Approximation of $\lambda_{\mathcal{T}}$. From here on in this section, we concentrate on dag-tasks that do not contain any conditional vertices. We construct a $(1+\epsilon)$-approximation algorithm for $\lambda_{\mathcal{T}}$ for any given $\epsilon > 0$. By Corollary 6.6, this allows us to test the second condition for speedup factors arbitrarily close to $2 - 1/m$ for EDF and arbitrarily close to $3 - 1/m$ for DM. The running-time of the $(1+\epsilon)$-approximation algorithm will be proportional to $1/\epsilon$. Thus, by increasing the running time of the test, one can decrease the required speedup factor.

Recall that $\lambda_{\mathcal{T}}$ represents the maximum relative load of an interval (over all possible job collections). Given an interval, its total workload is the sum of the workloads caused by the tasks $\tau_1, \ldots, \tau_n$. Since the tasks are independent, we can equivalently write

$$\lambda_{\mathcal{T}} = \sup_{t \in \mathbb{N}} \frac{\sum_{i=1}^{n} \text{work}_i(t)}{t},$$

where $\text{work}_i(t)$ is the maximum amount of work that may be done by S_∞ on jobs of task τ_i that are due in an interval of length t (i.e., the maximum workload caused by task τ_i during an interval of length t). This maximum is achieved when the deadline of some dag-job of τ_i coincides with the rightmost endpoint of the interval; in fact if this is not the case we can increase its release time without decreasing $\text{work}_i(t)$. Moreover, the other dag-jobs of τ_i are released as closely as possible. That is, if the interval is (without loss of generality) $[t_0, t_0 + t]$, then there is

— one dag-job with release date $t_0 + t - D_i$ and deadline $t_0 + t$,
— one dag-job with release date $t_0 + t - D_i - T_i$ and deadline $t_0 + t - T_i$,
— one dag-job with release date $t_0 + t - D_i - 2T_i$ and deadline $t_0 + t - 2T_i$,
— …
— in general, one dag-job with release date $t_0 + t - D_i - kT_i$, up to a k such that $t_0 + t - (k+1)T_i \leq t_0$ (more dag-jobs would not contribute to the amount of work done by S_∞ during $[t_0, t_0 + t]$).

We call such a sequence of dag-jobs the *backward-aligned sequence* for τ_i.

As a consequence of this structure, $\text{work}_i(t)$ is piecewise linear as a function of t, with a number of pieces that is $O(|V_i| \cdot t/T_i)$, as each dag-job is responsible for at most $|V_i|$ pieces.

Example 6.8. Consider the task set $\mathcal{T}$ consisting only of the dag-task of Figure 2. Its backward-aligned sequence is depicted in Figure 7. The corresponding function work is illustrated in

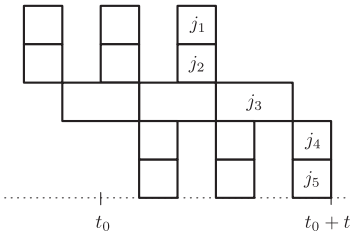

Fig. 7. The backward-aligned sequence for the work(t) function for the dag-task of Figure 2. For compactness, distinct dag-jobs are vertically aligned.

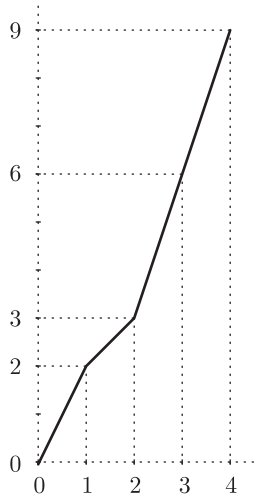

Fig. 8. The work function for the dag-task of Figure 2.

Figure 8. Moving backward from the most recently released to the least recently released dag-job, we can see that the work(t) function takes value 2 for $t = 1$, value $2 + 1 = 3$ for $t = 2$, value $3 + 3 = 6$ for $t = 3$, and value $3(t - 1)$ for $t > 3$. The quantity $\lambda_{\mathcal{T}}$ is equal to

$$\sup \left\{ \frac{2}{1}, \frac{3}{2}, \frac{6}{3}, \frac{9}{4}, \ldots, \frac{3(t-1)}{t}, \ldots \right\} = 3.$$

A simple pseudopolynomial time algorithm to compute $\text{work}_i(t)$ is Algorithm 1. The algorithm accumulates, for each $t' = 1, \ldots, t$, the number of processors that are busy during $[t_0 + t - t', t_0 + t - t' + 1)$ in the S_∞ schedule of the backward-aligned sequence.

For the purpose of obtaining a good approximation of $\lambda_{\mathcal{T}}$, it suffices to approximately compute $\sup_{t \in \mathbb{N}} \text{work}_i(t)/t$ for each task τ_i. In the next lemma, we first prove some (rough) upper and lower bounds for the quantity $\text{work}_i(t)$.

Lemma 6.9. *For any unconditional dag-task* $\tau_i = (\mathcal{G}_i, D_i, T_i)$,

$$\text{work}_i(t) \geq \max \left(\left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor, 0 \right) \cdot \text{vol}(\mathcal{G}_i), \tag{2}$$

$$\text{work}_i(t) \leq \left\lceil \frac{t}{T_i} \right\rceil \cdot \text{vol}(\mathcal{G}_i). \tag{3}$$

ALGORITHM 1: Workload computation

work$_i(t)$:
  $W \leftarrow 0$
  for $t' \leftarrow 1$ **to** t
    do
      $W \leftarrow W + B(t_0 + t - t')$
  ($B(x)$ = n. of busy processors in $[x, x+1)$ during
  S_∞'s schedule of the backward-aligned sequence)
  return W

Proof. Equation (2): There can be as many as $\lfloor (t + T_i - D_i)/T_i \rfloor$ releases of τ_i-dag-jobs in an interval of length t whose release date and deadline fall within the interval; each of them contributes $\mathrm{vol}(\mathcal{G}_i)$ to the work function.

Equation (3): There cannot be more than $\lceil t/T_i \rceil$ releases of τ_i-dag-jobs in an interval of length t whose deadlines fall within the interval. These dag-jobs are the only ones that contribute an amount of work larger than 0. □

The number of linear pieces of the function work$_i(t)$ can be large. Therefore, we approximate work$_i(t)$ by a function $\hat{\mathrm{w}}_i(t)$ defined as follows:

$$\hat{\mathrm{w}}_i(t) = \begin{cases} \mathrm{work}_i(t) & \text{if } t \le T_i/\epsilon + (1 + 1/\epsilon)D_i \\ \frac{t - D_i}{T_i}\,\mathrm{vol}(\mathcal{G}_i) & \text{if } t > T_i/\epsilon + (1 + 1/\epsilon)D_i. \end{cases}$$

We also define $\hat{\mathrm{w}}(t) = \sum_{i=1}^{n} \hat{\mathrm{w}}_i(t)$.

Lemma 6.10. *The function $\hat{\mathrm{w}}_i$ is piecewise linear and has*

$$O\left(\frac{1}{\epsilon} \cdot |V_i| \cdot \left(1 + \frac{D_i}{T_i}\right)\right)$$

many linear pieces.

Proof. Define $T_i^* = T_i/\epsilon + (1 + 1/\epsilon)D_i$. On interval (T_i^*, ∞), the function $\hat{\mathrm{w}}_i(t)$ is linear by construction (that is, it has only one linear piece). For the interval $[0, T_i^*]$, observe that $\hat{\mathrm{w}}_i(t)$ is piecewise linear and continuous and it can change its slope only when a job has finished processing in S_∞. The number of jobs of τ_i released during $[0, T_i^*]$ is bounded by $|V_i| \cdot \lceil T_i^*/T_i \rceil = O(\frac{1}{\epsilon} \cdot |V_i| \cdot (1 + \frac{D_i}{T_i}))$, which implies the claim. □

We will now use the function $\hat{\mathrm{w}}(t) = \sum_{i=1}^{n} \hat{\mathrm{w}}_i(t)$ instead of the term $\sum_{i=1}^{n} \mathrm{work}_i(t)$ to (approximately) compute $\lambda_{\mathcal{T}}$. Summing the bound of Lemma 6.10 for all tasks, we get:

Corollary 6.11. *The function $\hat{\mathrm{w}}$ is piecewise linear and has*

$$O\left(\frac{1}{\epsilon} \cdot \sum_{i=1}^{n} |V_i| \cdot \max_{i=1}^{n}\left(1 + \frac{D_i}{T_i}\right)\right)$$

many linear pieces.

For piecewise linear functions ϕ with few pieces, one can compute $\sup_{t \in \mathbb{N}} \phi(t)/t$ efficiently, which allows us to compute $\hat{\lambda}_{\mathcal{T}}$ and eventually to infer EDF- or DM-schedulability.

Lemma 6.12. *Let $\phi : \mathbb{N} \to \mathbb{N}$ be a piecewise linear function with ℓ_ϕ linear pieces and assume the value of the limit $\lim_{t\to\infty} \phi(t)/t$ is known. Then the value $\sup_{t \in \mathbb{N}} \phi(t)/t$ can be found by evaluating ϕ in $O(\ell_\phi)$ points.*

Proof. Let $[a, b]$ be a piece of ϕ, that is, a maximal interval in which ϕ is linear. Then $\phi(t)/t$ is monotone in $[a, b]$, so that $\max(\phi(a)/a, \phi(b)/b) \geq \phi(t)/t$ for all $t \in [a, b]$. Therefore, to compute $\sup_{t\in\mathbb{N}} \phi(t)/t$ it suffices to compute the value of ϕ in $\ell_\phi + 1$ points (one of these "points" is $t = \infty$). □

With this preparation, it remains to show that each $\hat{\mathrm{w}}_i(t)$ approximates $\mathrm{work}_i(t)$ sufficiently well, implying that also $\hat{\mathrm{w}}(t)$ is close to $\mathrm{work}(t)$.

Lemma 6.13. *For all $i = 1, \ldots, n$ and all $t \in \mathbb{N}$,*

$$\frac{1}{1+\epsilon}\mathrm{work}_i(t) \leq \hat{\mathrm{w}}_i(t) \leq \mathrm{work}_i(t).$$

Proof. First observe that $\mathrm{work}_i(t) \geq \hat{\mathrm{w}}_i(t)$, since for all $t > T_i/\epsilon + (1 + 1/\epsilon)D_i$, by Equation (2),

$$\begin{aligned}\frac{\mathrm{work}_i(t)}{\mathrm{vol}(\mathcal{G}_i)} &\geq \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor \geq \frac{t + T_i - D_i}{T_i} - 1 \\ &= \frac{t - D_i}{T_i} = \frac{\hat{\mathrm{w}}_i(t)}{\mathrm{vol}(\mathcal{G}_i)}.\end{aligned}$$

Moreover, again for $t > T_i/\epsilon + (1 + 1/\epsilon)D_i$ we have, using Equation (3),

$$\begin{aligned}\frac{\mathrm{work}_i(t)}{\hat{\mathrm{w}}_i(t)} &\leq \frac{\lceil t/T_i \rceil}{\frac{t-D_i}{T_i}} \leq \frac{t/T_i + 1}{t/T_i - D_i/T_i} \\ &= \frac{t + T_i}{t - D_i} \leq \frac{(D_i + T_i)/\epsilon + D_i + T_i}{(D_i + T_i)/\epsilon + D_i - D_i} = 1 + \epsilon.\end{aligned}$$ □

Corollary 6.14. *For all $t \in \mathbb{N}$,*

$$\frac{1}{1+\epsilon}\mathrm{work}(t) \leq \hat{\mathrm{w}}(t) \leq \mathrm{work}(t).$$

Lemma 6.15. *For any $\epsilon > 0$ there is a pseudopolynomial time $(1 + \epsilon)$-approximation for $\lambda_{\mathcal{T}}$.*

Proof. By Lemma 6.12, to compute $\hat{\lambda}_{\mathcal{T}}$ we need to invoke the function $\hat{\mathrm{w}}$ a number of times proportional to the number of its linear pieces. Such a number is bounded by Corollary 6.11 and it is pseudopolynomial. Note also that the limit $\lim_{t\to\infty} \hat{\mathrm{w}}(t)/t$ is simply $\sum_{i=1}^{n} \mathrm{vol}(\mathcal{G}_i)/T_i$. We can conclude that $\hat{\lambda}_{\mathcal{T}}$ can be computed in pseudopolynomial time (Algorithm 2), as long as one knows how to determine the set Q_i of endpoints of intervals in which each function work_i is linear. This will be discussed in Section 6.2. By Corollary 6.14, $\hat{\lambda}_{\mathcal{T}}$ is a good approximation to $\lambda_{\mathcal{T}}$. □

ALGORITHM 2: Load estimation

```
Load(T, ε):
    for i ← 1 to n
        do
            Q_i ← LinInt(τ_i, ε)
            (see Section 6.2)
    Q ← ⋃_{i=1}^{n} Q_i
    λ̂_T ← max( max_{t∈Q} Σ_{i=1}^{n} ŵ_i(t)/t, Σ_{i=1}^{n} vol(G_i)/T_i )
    return λ̂_T
```

Theorem 6.16. *Let $\epsilon > 0$. There is a pseudopolynomial time EDF-schedulability test with speedup $2 - 1/m + \epsilon$, and a pseudopolynomial time DM-schedulability test with speedup $3 - 1/m + \epsilon$.*

Proof. Follows by Corollary 6.6 and Lemma 6.15. The pseudocode description of the test is summarized in Algorithm 3. □

Example 6.17. Let us see how Algorithm 3 behaves for the dag-task set consisting of the task in Figure 2, when taking $\epsilon = 1/3$ and $m = 3$. Notice that, as discussed in Section 4.2, at least 4 *unit* speed processors are necessary to schedule the task; but the test is still allowed to return "EDF-schedulable with speed 2" as long as the task is indeed EDF-schedulable on 3 speed-2 processors (since $2 - 1/m + \epsilon = 2 - 1/3 + 1/3 = 2$). This is indeed the case, as the reader can easily check.

ALGORITHM 3: EDF/DM-schedulability test

```
SCHED(𝒯, m, ε):
    for i ← 1 to n
        do
            len(𝒢_i) ← length of the
            longest chain in 𝒢_i
    if ∃i : len(𝒢_i) > D_i then return "infeasible"
    λ̂_𝒯 ← LOAD(𝒯, ε)
    if λ̂_𝒯 > m then return "infeasible"
    return "EDF-schedulable with speed 2 − 1/m + ε
        and DM-schedulable with speed 3 − 1/m + ε"
```

Recall that in this example $D = 4$, $T = 2$. Since $\text{len}(\mathcal{G}) = 4 \leq D$, the first test passes. The approximate work $\hat{\text{w}}(t)$ is equal to $\text{work}(t)$ for $t \leq T/\epsilon + (1 + 1/\epsilon)D = 22$ and equal to $3(t - 4)$ for $t > 22$. The supremum of $\hat{\text{w}}(t)/t$ is 3, which is achieved for $t \to \infty$. So $\hat{\lambda}_{\mathcal{T}} = 3$ and the second test also passes. The Algorithm therefore returns "EDF-schedulable with speed 2" (as well as "DM-schedulable with speed 3").

6.2 Determination of the Linearity Intervals

As discussed in the proof of Lemma 6.15, the only missing part of the test is a subroutine to determine, for each task τ_i, the linearity intervals of the function work_i (up to the threshold T_i^*—recall the proof of Lemma 6.10). Recall that the evaluation of $\text{work}_i(t)$ requires considering a generic interval $[t_0, t_0 + t]$ and the S_∞ schedule of the backward-aligned sequence of τ_i's dag-jobs inside this interval.

Consequently, the endpoints of the linearity intervals are given by the following expressions:

$$-a_{q,v} = D_i + q \cdot T_i - \text{RT}(v),$$
$$-b_{q,v} = D_i + q \cdot T_i - (\text{RT}(v) - e_v),$$

where q ranges on nonnegative integers, v ranges on V_i (the nodes of τ_i's DAG $\mathcal{G}_i$), e_v is the wcet of v, and $\text{RT}(v)$ corresponds to the response time of v in the S_∞ schedule. The quantity $\text{RT}(v)$ can be easily computed as follows:

$$-\text{RT}(v) = e_v + \max_{u:\, u \text{ precedes } v} \text{RT}(u), \text{ if } v \text{ has some predecessor,}$$
$$-\text{RT}(v) = e_v, \text{ if } v \text{ has no predecessors.}$$

Note that we need only consider the values of $q \geq 0$ such that $a_{q,v}$ or $b_{q,v}$ are less than the threshold T_i^* (by the proof of Lemma 6.10). We summarize the resulting procedure in Algorithm 4.

ALGORITHM 4: Determination of linearity intervals

LinInt(τ_i, ϵ):

do

$T_i^* \leftarrow T_i/\epsilon + (1 + 1/\epsilon)D_i$

$A_i \leftarrow \cup_{q\geq 0, v\in V_i}\{a_{q,v} \ : \ a_{q,v} \leq T_i^*\}$

$B_i \leftarrow \cup_{q\geq 0, v\in V_i}\{b_{q,v} \ : \ b_{q,v} \leq T_i^*\}$, where

$a_{q,v} = D_i + q \cdot T_i - \mathrm{RT}(v)$,

$b_{q,v} = D_i + q \cdot T_i - (\mathrm{RT}(v) - e_v)$.

return $A_i \cup B_i$

7 FROM UNCONDITIONAL TASKS TO CONDITIONAL TASKS

In this section, we will extend the schedulability tests of the previous section for unconditional tasks to conditional tasks by transforming any conditional task into an equivalent unconditional task, in the sense that they have the same relevant parameters len, vol, and the same work function.

Before presenting the transformation we find it helpful to define an additional function for conditional tasks called the *remaining demand function*, denoted rdem. Let $\mathcal{J}_i$ denote all possible *complete* collections of jobs that comprise a single dag-job of τ_i; each collection $J \in \mathcal{J}_i$ corresponds to a different conditional flow of execution (by J being "complete", we mean that all job-vertices of $\mathcal{G}_i$ arising from that flow of execution are included in J). Then for each $J \in \mathcal{J}_i$, all jobs in J have a common release date and a common deadline—the release date and deadline of the dag-job that generates them. In line with notation used so far, we use $S_\infty(J)$ to denote, for any job collection J, the idealized schedule obtained by allocating a unit speed processor to each job in J the instant it is available for execution, and executing this job upon its allocated processor until it completes.

Definition 7.1. Consider any $J \in \mathcal{J}_i$ for a dag-job of a given conditional dag-task τ_i, and let t denote any positive real number. Let $\mathrm{rdem}^J(t)$ denote the amount of work remaining to be executed in schedule $S_\infty(J)$ a duration t time units after the release date of the dag-job. Moreover, $\mathrm{rdem}_i(t)$ is defined to be the maximum value of $\mathrm{rdem}^J(t)$ over all collections of jobs $J \in \mathcal{J}_i$.

Note that $\mathrm{rdem}_i(t)$ expresses the maximum amount of work that can remain to be executed, if a single dag-job of task τ_i were to execute for t time units upon infinitely many unit speed processors (that is, the maximum difference between the wcet's of all the jobs in J and the amount of execution that has already occurred).

The significance of the remaining demand function arises from its relationship with the work function, as captured by the following lemma.

Lemma 7.2. *Consider a conditional dag-task τ_i. For all $t \geq 0$,*

(i) if $0 \leq t \leq D_i$, then we have

$$\mathrm{work}_i(t) = \sum_{h=0}^{\lfloor D_i/T_i \rfloor} \mathrm{rdem}_i(D_i - t + hT_i).$$

(ii) if $0 < D_i < t$, then for $k = \lfloor (t - D_i)/T_i \rfloor + 1$, we have

$$\mathrm{work}_i(t) = k \cdot \mathrm{vol}(\mathcal{G}) + \sum_{h=0}^{\lfloor D_i/T_i \rfloor} \mathrm{rdem}_i(D_i - t + (k + h)T_i).$$

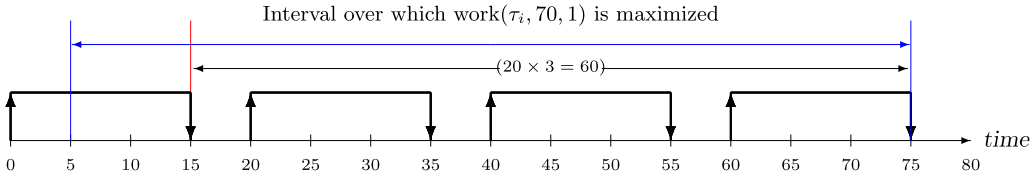

Fig. 9. Computing work(τ_i, 70, 1) for the task of Figure 11.

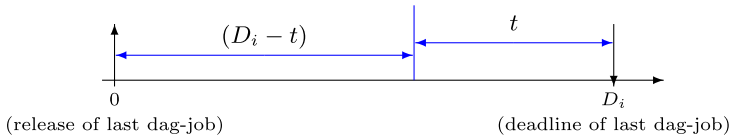

Fig. 10.

Proof. Recall that $\text{work}_i(t)$ is defined as the *maximum* value, over all job collections J that may be generated by τ_i, of the amount of execution occurring within some interval of duration t in the schedule $S_\infty(J)$ of jobs in J that have deadlines within this interval.

First, observe that for all $t \geq D_i$ according to the definition of the rdem function, with respect to a dag-task τ_i that can be successfully scheduled by S_∞, we have $\text{rdem}_i(t) = 0$.

As in the case of unconditional tasks in the previous section, we now observe that the maximum of $\text{work}_i(t)$ is achieved when the dag-job of τ_i that contributes to J with highest deadline has a deadline that coincides with the rightmost endpoint of the interval, so w.l.o.g. assume that the interval on which $\text{work}_i(t)$ is achieved is of the form $[D_i - t, D_i]$. Analogously as we have seen in Section 6.1, the maximum workload is achieved when we have a backward-aligned sequence of dag-jobs, that is, the deadlines of the dag-jobs contributing to $\text{work}_i(t)$ are $D_i - hT_i$, $h = 0, 1, \ldots$ (the only difference being that now distinct dag-jobs may result in different job-vertices of $\mathcal{G}_i$ being released). An example is illustrated in Figure 9.

We now consider the case $t \leq D_i$ and prove *(i)*. When $t \leq D_i \leq T_i$, for intervals of duration t, there is exactly one dag-job that contributes to $\text{work}_i(t)$ (Figure 10).

Upon infinitely many unit speed processors, the maximum work remaining to be done $(D_i - t)$ time units after the dag-job's arrival is, by definition, $\text{rdem}_i(D_i - t)$. It is evident from the definition of the work function and from the picture above that this is also equal to $\text{work}_i(t)$. If $D_i < T_i$ since $\lfloor D_i/T_i \rfloor = 0$ *(i)* is proven; if $D_i = T_i$, then we observe that when $h = 1$ then $D_i - t + hT_i > D_i$ and hence $\text{rdem}_i(D_i - t + hT_i) = 0$ and hence *(i)* is also proven.

If, however, $t \leq T_i < D_i$, then there may be multiple dag-jobs contributing to $\text{work}_i(t)$. Let $h = 0, 1, \ldots$, index the dag-jobs backward so that the last released dag-job has index 0. Then the maximum work remaining to be done on the hth dag-job in the $S_\infty(J)$ schedule at time $D_i - t$ is exactly $\text{rdem}_i(D_i - t + hT_i)$. It follows from the definition of the work function that the sum of such terms equals $\text{work}_i(t)$. Finally, note that only the first $\lceil D_i/T_i \rceil$ such terms can be nonzero. This implies

$$\begin{aligned}\text{work}_i(t) &= \sum_{h=0}^{\lceil D_i/T_i \rceil} \text{rdem}_i(D_i - t + hT_i) \\ &= \sum_{h=0}^{\lfloor D_i/T_i \rfloor} \text{rdem}_i(D_i - t + hT_i) + \text{rdem}_i(D_i - t + \lceil D_i/T_i \rceil T_i).\end{aligned}$$

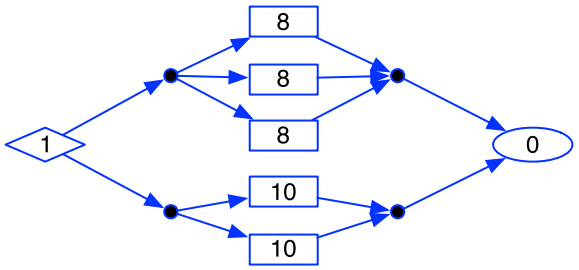

Fig. 11. The DAG $\mathcal{G}_i$ of an example conditional dag-task τ_i. For this task, $D_i = 15$ and $T_i = 20$. Note that $\text{len}(\mathcal{G}_i) = 11$ and corresponds to taking the lower branch of the conditional, while $\text{vol}(\mathcal{G}_i) = 25$ and corresponds to taking the upper branch.

We now show that $\text{rdem}_i(D_i - t + \lceil D_i/T_i \rceil T_i) = 0$. Let $h = \lceil D_i/T_i \rceil$ and note that $0 \le t \le T_i < D_i$ implies $hT_i \ge D_i$ and $D_i - t + hT_i \ge D_i - t + D_i \ge D_i$. Therefore when $h = \lceil D_i/T_i \rceil$ and $0 \le t \le D_i$ we have $\text{rdem}_i(D_i - t + hT_i) = 0$. This concludes the proof of *(i)*.

To prove *(ii)*, we reduce the computation of $\text{work}_i(t)$, $t > D_i$, to the computation of $\text{work}_i(t - T_i)$. When $t > D_i$, at least one entire scheduling window of a dag-job of τ_i fits in the interval considered by the work function, and therefore $\text{work}_i(t) = \text{work}_i(t - T_i) + \text{vol}(\mathcal{G}_i)$. This can be repeated $\lfloor (t - D_i)/T_i \rfloor + 1$ times, thus proving *(ii)* and completing the proof of the Lemma. □

We have thus reduced the problem of computing $\text{work}_i(t)$ for all t to that of computing $\text{rdem}_i(t)$ for values of $t \le D_i$. We first illustrate with an example how to compute $\text{rdem}_i(t)$. Consider a task τ_i with parameters $D_i = 15$ and $T_i = 20$, and a DAG $\mathcal{G}_i$ depicted in Figure 11.[2] According to $\mathcal{G}_i$, a conditional expression having wcet equal to 1 is evaluated each time a dag-job of τ_i is released. Depending upon the outcome of this evaluation, we must execute either three jobs of wcet 8, each of them executable in parallel, or two jobs of wcet 10, each of them executable in parallel. There is no execution cost (and hence no wcet) associated with recombining the branches; hence, the conditional vertex depicting the end of the conditional construct has a wcet of zero.

Notice that $|\mathcal{J}_i| = 2$ for this task; i.e., there are two possible flows of execution through this DAG for a single dag-job of τ_i, depending upon whether the upper or the lower branch is taken upon evaluation of the conditional expression. We consider these two cases separately; the resulting functions are depicted graphically in Figure 12.

— *The upper branch is taken.* The amount of work remaining is depicted as the line beginning at the point $(0, 25)$ in Figure 12 (the blue line, for those reading this on a color medium). At the beginning, there are 25 units of work remaining to be done. Only one job—the one corresponding to the conditional vertex—executes over the interval $[0, 1)$; hence the slope of the line during this interval is -1. Once the conditional expression has been evaluated, three jobs execute in parallel for eight time units; hence the slope is -3.

— *The lower branch is taken.* The amount of work remaining is depicted as the line beginning at the point $(0, 21)$ in Figure 12 (the red line). At the beginning, there are 21 units of work remaining to be done. As in the case above, only the job corresponding to the conditional vertex executes over the interval $[0, 1)$; hence the slope of the line during this interval is -1. Once the conditional expression has been evaluated, two jobs execute in parallel for 10 time units; hence, the slope is -2.

[2]Observe *en passant* that this DAG is the same as the part of the DAG in Figure 1 enclosed in a large rectangle, representing the upper conditional construct of that task.

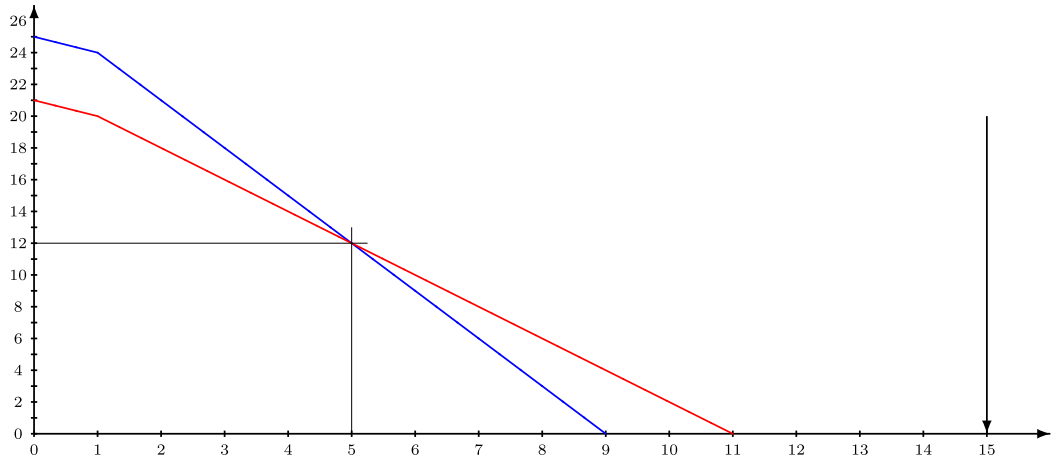

Fig. 12. Remaining work (y-axis) as a function of time elapsed (x-axis) since the release of a dag-job of the task τ_i.

It is immediately evident from Figure 12 that for values of $t \leq 5$, executing the upper branch leaves more work remaining to be done after t time units (this is depicted as the blue line); for values of $t \geq 5$, executing the lower branch leaves more work remaining to be done (the red line); i.e., the upper envelope of the two individual rdem functions[3] corresponding to the two different paths through the conditional code represents the maximum amount of remaining work for all values of t, and $\text{rdem}_i(t)$ is therefore the upper envelope of the two individual rdem lines plotted in Figure 12.

The above discussion exemplifies that it is not possible to identify one particular path through the code such that simply evaluating this path suffices for determining the worst-case behavior of the task for all values of t. In fact, for different values of t there are different paths through the conditional code that represent the "worst case"; i.e., leaving the maximum amount of work remaining to be done.

Let us now apply Lemma 7.2 to compute $\text{work}_i(t)$ for the example task τ_i for values of $t = 65, 70, 72$, and 78. Observe that $k = 1 + \lfloor (t - D_i)/T_i \rfloor = 3$ for $55 \leq t < 75$, and $k = 4$ for $t \geq 75$, and hence $k \cdot \text{vol}(\mathcal{G}_i) = 25 \times 3 = 75$ for $t = 65$, 70, 72, while $k \cdot \text{vol}(\mathcal{G}_i) = 25 \times 4 = 100$ for $t = 78$. Therefore, according to Lemma 7.2,

$$\begin{aligned}
\text{work}_i(65) &= 75 + \text{rdem}_i(10) = 75 + 2 = 77 \\
\text{work}_i(70) &= 75 + \text{rdem}_i(5) = 75 + 12 = 87 \\
\text{work}_i(72) &= 75 + \text{rdem}_i(3) = 75 + 18 = 93 \\
\text{work}_i(78) &= 100 + \text{rdem}_i(17) = 100 + 0 = 100.
\end{aligned}$$

The approach considered for computing the rdem function of the example dag-task has an obvious generalization:

—Determine the function $\text{rdem}^J(t)$ for each $J \in \mathcal{J}_i$.
—Take the upper envelope: $\text{rdem}_i(t) = \max_{J \in \mathcal{J}_i} \text{rdem}^J(t)$.

Notice that this approach suffers from the same problem as the multi-DAG model of Fonseca et al. (2015): The number of distinct possible flows of execution, and hence the number of

[3]The *upper envelope* of a collection of functions is defined to be the pointwise maximum of these functions.

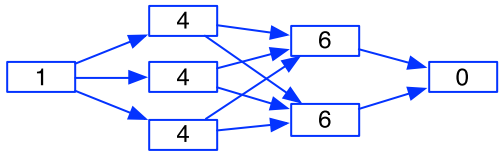

Fig. 13. The DAG obtained by transforming the upper conditional construct of Figure 1.

distinct collections of jobs J for which we would need to compute $\text{rdem}^J(t)$, may be exponential in the size of the DAG. Therefore, the overall algorithm suggested would require exponential time. We will develop a more efficient approach that avoids explicitly computing the rdem function for all possible $J \in \mathcal{J}_i$.

7.1 Schedulability of Conditional Dag-task Sets

Given a task system of conditional dag-tasks, we will efficiently *transform* each conditional dag-task τ_i into an "equivalent" unconditional one $\hat{\tau}_i$, in the sense that both have the same len, vol, deadline, and period parameters, and they will satisfy the property that

$$\text{work}_i(t) = \widehat{\text{work}}_i(t) \text{ for all } t \geq 0. \tag{4}$$

It will then follow that a conditional dag-task set $\mathcal{T}$ is feasible if and only if the equivalent unconditional dag-task set $\hat{\mathcal{T}}$ is feasible. Therefore, we can then apply the pseudo-polynomial time schedulability tests of Section 6 to unconditional dag-task sets to obtain a pseudo-polynomial time EDF schedulability test and DM schedulability test for conditional dag-task sets that has speedup factor $(2 - 1/m + \epsilon)$ and $(3 - 1/m + \epsilon)$, respectively, for any constant $\epsilon > 0$.

We first illustrate this transformation for our example task of Figure 11. Consider a task $\hat{\tau}_i$ with $\hat{D}_i = D_i = 15$ and $\hat{T}_i = T_i = 20$ that has the DAG $\hat{\mathcal{G}}_i$ depicted in Figure 13. It is readily verified that the remaining work function of this task is identical to the upper envelope of the two remaining work functions depicted in Figure 12. Hence, *tasks τ_i and $\hat{\tau}_i$ have identical* rdem *functions* (and, therefore, identical work functions). Hence $\hat{\tau}_i$ is an unconditional dag-task that is "equivalent" to τ_i in the sense that both have the same work functions.

To obtain $\hat{\tau}_i$, we set out to construct an unconditional dag-task with rdem function equal to the upper envelope of the two rdem functions plotted in Figure 12:

- —Since the upper envelope has a slope of -1 over the interval $[0, 1)$, we introduced a single vertex with wcet $= (1 - 0) = 1$.
- —The slope of the upper envelope is then -3 over the interval $[1, 5)$; this is modeled by adding a second "layer" of three vertices, each with wcet $= (5 - 1) = 4$, as successor vertices.
- —The slope of the upper envelope is subsequently -2 over the interval $[5, 11)$; this is modeled by adding a third layer of two vertices, each with wcet $= (11 - 5) = 6$, as successor vertices.
- —The final layer with a single vertex with wcet $= 0$ represents the end of the conditional construct.

Notice that we have added edges from each vertex in each layer to all vertices in the immediately succeeding layer and that the volume and the length of $\hat{\tau}_i$ are equal to the volume and length of τ_i, respectively.

As mentioned, we should avoid explicitly computing all rdem functions for all the (possibly exponentially many) possible execution flows. How to do so in transforming a conditional dag-task into an equivalent unconditional one, we illustrate first by another more complete example, prior to formally proving it.

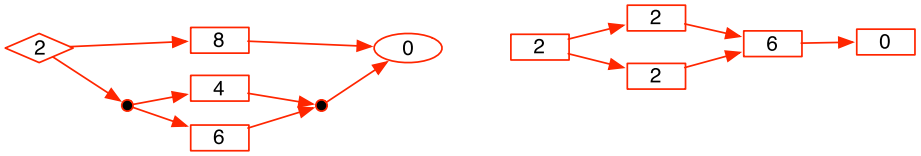

Fig. 14. Transforming the lower conditional construct of Figure 1.

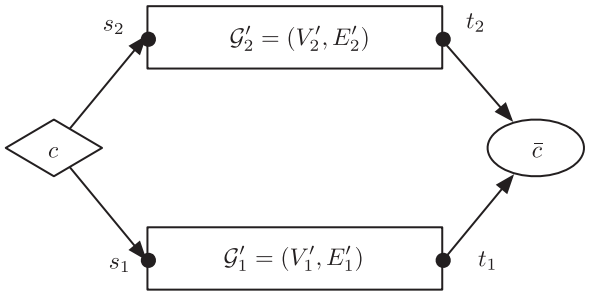

Fig. 15. A conditional construct. Vertices s_1 and t_1 (vertices s_2 and t_2, respectively) are the sole source vertex and sink vertex of $\mathcal{G}'_1$ ($\mathcal{G}'_2$, respectively).

We already observed that the DAG in Figure 11 appears as one conditional construct in the larger DAG of Figure 1—the one that is enclosed within a larger rectangle. If we were to replace that entire conditional construct in the DAG of Figure 1 with the DAG of Figure 13, then we would obtain a conditional DAG with *one fewer conditional construct*, for which (by Theorem 7.3 below) the work function is identical to the work function of the DAG of Figure 1.

We can do likewise for the other (lower) conditional construct in the DAG of Figure 1; Figure 14 depicts the application of a similar transformation to this lower conditional construct. Finally, Figure 16 depicts the unconditional DAG resulting from applying both transformations (and some cosmetic changes—deletions of the dummy source and sink vertices).

The transformation algorithm. We now describe our algorithm for transforming a conditional dag-task τ_i into an equivalent unconditional dag-task $\hat{\tau}_i$, with $\hat{D}_i = D_i$ and $\hat{T}_i = T_i$. To obtain $\hat{\mathcal{G}}_i$, we start out with the DAG $\mathcal{G}_i$ and repeatedly

(1) identify an innermost conditional construct;
(2) construct an unconditional DAG that is equivalent to this innermost conditional construct (we describe below how to do so); and
(3) replace the identified innermost conditional construct with the constructed equivalent unconditional DAG,

until there are no remaining conditional constructs in the DAG.

We now describe how to construct an equivalent unconditional DAG from a single innermost conditional construct that is notated as in Figure 15.

—Separately construct the rdem functions for the collections of jobs corresponding to all the vertices in $\{c, \bar{c}\} \cup V'_1$ and $\{c, \bar{c}\} \cup V'_2$, respectively. Each rdem is piecewise linear, with the number of linear segments bounded from above by the number of vertices in the graph, and each linear segment has a negative integer slope.

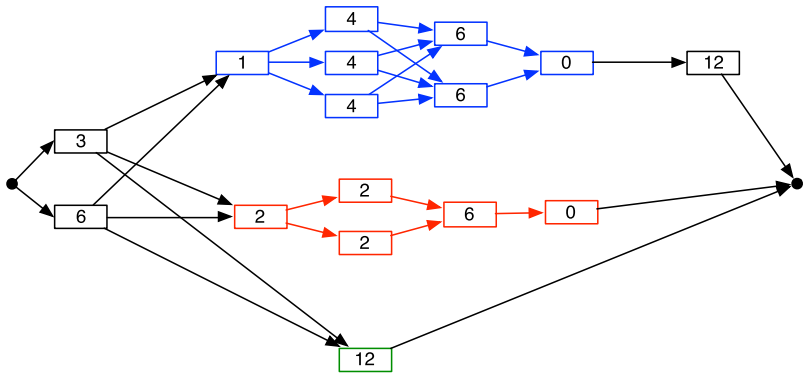

Fig. 16. The DAG of Figure 1 with both conditional constructs removed.

—Determine the upper envelope of these two rdem functions. This upper envelope is piecewise linear, each linear segment has a negative integer slope, and the total number of linear segments is bounded from above by twice the number of vertices in $\{c, \bar{c}\} \cup V_1' \cup V_2'$.
—Construct a DAG $\hat{\mathcal{G}} = (\hat{V}, \hat{E})$ that has the same rdem function as the upper envelope determined above. This graph is constructed as a "layered" one, with as many layers as there are linear segments in the upper envelope plus 1. The number of vertices in the kth layer is equal to the (negation of the) slope of the kth segment of the upper envelope, and each vertex is labeled with a wcet equal to the duration of the time axis spanned by this kth segment. The last layer consists of a single sink vertex with wcet = 0. There is an edge from each vertex in a layer to each vertex in the immediately succeeding layer.

The resulting DAG $\hat{\mathcal{G}} = (\hat{V}, \hat{E})$ is the equivalent unconditional DAG. Before stating the theorem we observe two simple properties of the basic transformation defined above:

—(VP) The volume of $\hat{\mathcal{G}}$ is equal to the volume of $\mathcal{G}$;
—(LP) The length of $\hat{\mathcal{G}}$ is equal to the length of $\mathcal{G}$.

Theorem 7.3. *Let τ_i denote a conditional dag-task, and $\hat{\tau}_i$ denote the (perhaps conditional) dag-task obtained by replacing an innermost conditional construct in the DAG $\mathcal{G}_i$ of τ_i by an equivalent unconditional DAG $\hat{\mathcal{G}}_i$ as described above.*

Then, for all $t, 0 \leq t \leq D_i$, $\text{rdem}_i(t) = \widehat{\text{rdem}}_i(t)$ (and therefore $\text{work}_i(t) = \widehat{\text{work}}_i(t)$ for all t).

Before presenting the proof we provide some intuition, restricting to the case of a constrained deadline task τ_i. Let $\mathcal{J}_i$ denote all possible complete collections of jobs that comprise dag-jobs of τ_i and that have their deadlines in the considered interval. If $D_i \leq T_i$, then, for any given t, there is at most one dag-job that is released and due within the considered interval. We will see that the volume of $\mathcal{G}_i$ and $\hat{\mathcal{G}}_i$ are the same; for this reason we can now focus on collections of jobs that belong to just one dag-job of τ_i.

Let us assume that the innermost conditional construct that is replaced in τ_i is notated as in Figure 15.

Consider any pair $J_1 \in \mathcal{J}_i, J_2 \in \mathcal{J}_i$ of such complete collections of jobs comprising a single dag-job of τ_i, that differ only in the choices they make with regard to the conditional construct that is selected for replacement. That is, exactly one of J_1, J_2 contains (jobs corresponding to) all the vertices in V_1', while the other contains (jobs corresponding to) all the vertices in V_2'; other than these differences, they both contain (jobs corresponding to) exactly the same collection of vertices.

Let Δ be the difference between the sum of the wcets of all the jobs in V_1' and the sum of the wcets of all the jobs in V_2'. Consider the functions $\text{rdem}^{J_1}(t)$ and $\text{rdem}^{J_2}(t)$ as functions of t. At time-instant $t = 0$, $\text{rdem}^{J_1}(t) - \text{rdem}^{J_2}(t) = \Delta$.

Observe that the schedules $S_\infty(J_1)$ and $S_\infty(J_2)$ are identical prior to the instant, t_o say, the time at which both begin the execution of the job corresponding to the conditional vertex c of Figure 15. Hence, at t_o, we still have that $\text{rdem}^{J_1}(t_o) - \text{rdem}^{J_2}(t_o) = \Delta$.

Examining the schedules $S_\infty(J_1)$ and $S_\infty(J_2)$ at times $> t_o$ and prior to the execution of vertex $\bar{c}$ in either schedule, we observe that

- —Those jobs that belong in both J_1 and J_2 execute at the same times in both schedules; hence, the decrease in the remaining demand due to the execution of these jobs proceeds in exactly the same manner both in $\text{rdem}^{J_1}(t)$ and in $\text{rdem}^{J_2}(t)$.
- —Of course, the execution of the jobs belonging to exactly one of J_1 or J_2 proceeds differently in the schedules $S_\infty(J_1)$ and $S_\infty(J_2)$; the manner in which these executions happen is represented in the rdem functions that were separately constructed for the collections of jobs corresponding to the vertices in $\{c, \bar{c}\} \cup V_1'$ and $\{c, \bar{c}\} \cup V_2'$, respectively.

At times $> t_o$ and prior to the execution of vertex $\bar{c}$ in either schedule, we can therefore consider the rdem functions for each of J_1 and J_2 as the sum of a part that is identical in both, and a part that is equal to the rdem functions that were separately constructed for the collections of jobs corresponding to the vertices in $\{c, \bar{c}\} \cup V_1'$ and $\{c, \bar{c}\} \cup V_2'$, respectively. And as was argued in the case when we considered a single conditional construct in isolation (and therefore had only two possible flows of execution), the upper envelope of both individual rdem functions represents a tight upper bound on the rdem function over both the flows of execution that are represented by the part of J_1 and J_2 that differ from each other.

Note that both len and vol of $\mathcal{G}_i$ and $\hat{\mathcal{G}}_i$ are the same.

The correctness follows by observing that by construction, the DAG $\hat{\mathcal{G}} = (\hat{V}, \hat{E})$ that replaces the conditional construct has an rdem function exactly equal to this upper envelope. We are now ready to prove Theorem 7.3.

Proof. We use the shorthand $f^J(t) = \text{rdem}^J(t)$. We already observed that

$$\text{rdem}_i(t) = \max_{J \in \mathcal{J}_i} f^J(t), \tag{5}$$

where $\mathcal{J}_i$ is the set of possible complete collections of jobs comprising dag-jobs of the original task τ_i. Similarly,

$$\widehat{\text{rdem}}_i(t) = \max_{J \in \hat{\mathcal{J}}_i} f^J(t), \tag{6}$$

where $\hat{\mathcal{J}}_i$ is the set of possible complete collections of jobs comprising dag-jobs of the transformed task $\hat{\tau}_i$. To prove the theorem we need to show that for all t the right-hand sides of Equations (5) and (6) are equal.

Given τ_i and t there are at most $k_i(t) = \lceil t/T_i \rceil$ dag-jobs of τ_i with deadlines in an interval of length t that have deadlines that are T_i far apart.

Let $\mathcal{J}_i^j$ ($\hat{\mathcal{J}}_i^j$) be the collection of jobs comprising a single dag-job j of $\mathcal{J}_i$ ($\hat{\mathcal{J}}_i$), $j = 1, 2 \ldots, k_i(t)$. Clearly, if $J \in \mathcal{J}_i$, then there exist disjoint sets $J^j \in \mathcal{J}_i^j$, $j = 1, 2 \ldots, k_i(t)$ such that $J = \bigcup_{j=1}^{k_i(t)} J^j$. Hence, $\bigcup_{j=1}^{k_i(t)} \mathcal{J}_i^j = \mathcal{J}_i$ and $\bigcup_{j=1}^{k_i(t)} \hat{\mathcal{J}}_i^j = \hat{\mathcal{J}}_i$; since jobs of $\mathcal{J}_i^j$ and $\mathcal{J}_i^{j'}$ ($\hat{\mathcal{J}}_i^j$ and $\hat{\mathcal{J}}_i^{j'}$) belong to

different dag-jobs of τ_i we can write

$$\max_{J \in \mathcal{J}_i} f^J(t) = \sum_{j=1}^{k_i(t)} \max_{J \in \mathcal{J}_i^j} f^J(t), \tag{7}$$

$$\max_{\hat{J} \in \hat{\mathcal{J}}_i} f^{\hat{J}}(t) = \sum_{j=1}^{k_i(t)} \max_{\hat{J} \in \hat{\mathcal{J}}_i^j} f^{\hat{J}}(t). \tag{8}$$

Therefore to prove that the right-hand sides of Equations (5) and (6) are equal, it is sufficient to show that for all t and all j, $j = 1, 2 \ldots, k_i(t)$, $\max_{J \in \mathcal{J}_i^j} f^J(t) = \max_{\hat{J} \in \hat{\mathcal{J}}_i^j} f^{\hat{J}}(t)$.

In the proof of Lemma 7.2, we have shown that the maximum value of rdem is achieved when jobs of $\mathcal{J}_i^1$ ($\hat{\mathcal{J}}_i^1$) have deadline at t and jobs of $\mathcal{J}_i^j$ ($\hat{\mathcal{J}}_i^j$) have deadline at $t - (j-1)T_i$ (i.e., that dag-jobs of τ_i are backwardly aligned). Therefore, for $j = 1, 2 \ldots, k_i(t)$, we have

$$\max_{J \in \mathcal{J}_i^j} f^J(t) = \max_{J \in \mathcal{J}_i^1} f^J(t - (j-1)T_i),$$

$$\max_{\hat{J} \in \hat{\mathcal{J}}_i^j} f^{\hat{J}}(t) = \max_{\hat{J} \in \hat{\mathcal{J}}_i^1} f^{\hat{J}}(t - (j-1)T_i).$$

The above implies that to prove the theorem it is sufficient to show that for all t

$$\max_{J \in \mathcal{J}_i^1} f^J(t) = \max_{\hat{J} \in \hat{\mathcal{J}}_i^1} f^{\hat{J}}(t). \tag{9}$$

There is a natural bijection between job collections in $\hat{\mathcal{J}}_i^1$ and pairs of job collections in $\mathcal{J}_i^1$: We associate to $\hat{J} \in \hat{\mathcal{J}}_i^1$ the pair $J_1, J_2 \in \mathcal{J}_i^1$ such that all the jobs generated from conditional branches, except the one being transformed, are the same in J_1, J_2 as they are in $\hat{J}$. Then, J_1 and J_2 correspond to the two possible completions of these set of jobs with the jobs generated inside the conditional branch being transformed. In particular, J_1, J_2 differ only in the jobs belonging to the conditional construct being transformed.

In the sequel, we will use this bijection: Given $\hat{J}$, then J_1 and J_2 are the sets associated to $\hat{J}$. Hence, to prove Equation (9), it is sufficient to show that for all $\hat{J} \in \mathcal{J}_i^1$

$$f^{\hat{J}}(t) = \max\{f^{J_1}(t), f^{J_2}(t)\} \qquad \text{for all } t.$$

We now analyze in more detail the structure of the functions $f^{\hat{J}}$, f^{J_1}, f^{J_2} and of the corresponding schedules $S_\infty(\hat{J})$, $S_\infty(J_1)$, $S_\infty(J_2)$. To do so, we need to distinguish the jobs according to whether they (1) precede—or are unrelated to—the conditional section being transformed, (2) belong to the conditional section or to its replacement, or (3) follow the conditional section. More formally, if $\mathcal{G}$ and $\hat{\mathcal{G}}$ stand for the original and transformed DAGs, respectively, then we define:

(1) A (A_1, A_2, respectively) to be the set of jobs of $\hat{J}$ (J_1, J_2, respectively) whose corresponding vertices are *not reachable from* c in $\hat{\mathcal{G}}$ ($\mathcal{G}$); and $\alpha(t)$ ($\alpha_1(t)$, $\alpha_2(t)$) to be the cumulative remaining demand of A (A_1, A_2) at time t in schedule $S_\infty(\hat{J})$ ($S_\infty(J_1)$, $S_\infty(J_2)$).
(2) B (B_1, B_2) to be the set of jobs of $\hat{J}$ (J_1, J_2) whose corresponding vertices are *reachable from* c *but not from* $\bar{c}$, *plus* $\bar{c}$, in $\hat{\mathcal{G}}$ ($\mathcal{G}$); and $\beta(t)$ ($\beta_1(t)$, $\beta_2(t)$) to be the cumulative remaining demand of B (B_1, B_2) at time t in schedule $S_\infty(\hat{J})$ ($S_\infty(J_1)$, $S_\infty(J_2)$).
(3) C (C_1, C_2) to be the set of jobs of $\hat{J}$ (J_1, J_2) whose corresponding vertices are *reachable from* $\bar{c}$, *excluding* $\bar{c}$, in $\hat{\mathcal{G}}$ ($\mathcal{G}$); and $\gamma(t)$ ($\gamma_1(t)$, $\gamma_2(t)$) to be the cumulative remaining demand of C (C_1, C_2) at time t in schedule $S_\infty(\hat{J})$ ($S_\infty(J_1)$, $S_\infty(J_2)$);

(4) t_1 (t_2, respectively) to be the instant at which $S_\infty(J_1)$ ($S_\infty(J_2)$, respectively) completes the job corresponding to the conditional vertex $\bar{c}$.

Note that the sets A, B, C form a partition of $\hat{J}$ and therefore, for all t,

$$f^{\hat{J}}(t) = \alpha(t) + \beta(t) + \gamma(t).$$

Similarly, for all t it holds that

$$f^{J_1}(t) = \alpha_1(t) + \beta_1(t) + \gamma_1(t)$$
$$f^{J_2}(t) = \alpha_2(t) + \beta_2(t) + \gamma_2(t).$$

Without loss of generality, assume that $t_1 \geq t_2$. We observe that:

(P1) $\alpha(t) = \alpha_1(t) = \alpha_2(t)$ for all t. The jobs in A, A_1, and A_2 correspond to the same vertices of the original and transformed DAGs and are scheduled at exactly the same times in all three schedules $S_\infty(\hat{J})$, $S_\infty(J_1)$, and $S_\infty(J_2)$.

(P2) $\beta(t) = \max(\beta_1(t), \beta_2(t))$ for all t. This follows by construction of the transformed inner DAG and by the fact that execution of conditional vertex c starts at the same time in each of the three schedules.

(P3) $\beta(t) = \beta_1(t) \geq \beta_2(t)$ for all $t \geq t_2$. At any time t with $t_2 \leq t < t_1$, the jobs in B_1 have not all completed in $S_\infty(J_1)$, while all jobs in B_2 have completed in $S_\infty(J_2)$, so $\beta_1(t) > 0 = \beta_2(t)$. And for $t \geq t_1$, $\beta(t) = \beta_1(t) = \beta_2(t) = 0$.

(P4) $\gamma(t) = \gamma_1(t) \geq \gamma_2(t)$ for all t. The jobs in C, C_1 and C_2 correspond to the same vertices of the original and transformed DAGs. The start time of any job j is given by the length of the longest chain of jobs from the source to j. The start time of a job of C_1 in $S_\infty(J_1)$ *cannot be earlier* than the start time of the corresponding job of C_2 in $S_\infty(J_2)$, since $t_1 \geq t_2$. Using property (LP) of the basic transformation, we also have that the start time of a job $j \in C$ in $S_\infty(\hat{J})$ *equals* the start time of the corresponding job $j_1 \in C_1$ in $S_\infty(J_1)$.

(P5) $\gamma(t) = \gamma_1(t) = \gamma_2(t)$ for all $t < t_2$. Before t_2, *no* job of C, C_1, or C_2 has yet received execution in any of the three schedules.

Combining the above points we obtain that, when $t < t_2$,

$$\begin{aligned} f^{\hat{J}}(t) &= \alpha(t) + \beta(t) + \gamma(t) \\ &= \alpha(t) + \max(\beta_1(t), \beta_2(t)) + \gamma(t) && \text{(by (P2))} \\ &= \max(\alpha(t) + \beta_1(t) + \gamma(t), \alpha(t) + \beta_2(t) + \gamma(t)) \\ &= \max(\alpha_1(t) + \beta_1(t) + \gamma_1(t), \alpha_2(t) + \beta_2(t) + \gamma_2(t)) \\ & && \text{(by (P1), (P5))} \\ &= \max(f^{J_1}(t), f^{J_2}(t)), \end{aligned}$$

while for $t \geq t_2$,

$$\begin{aligned} f^{\hat{J}}(t) &= \alpha(t) + \beta(t) + \gamma(t) \\ &= \alpha_1(t) + \beta_1(t) + \gamma_1(t) \ (= f^{J_1}(t)) \\ & && \text{(by (P1), (P3), (P4))} \\ &\geq \alpha_2(t) + \beta_2(t) + \gamma_2(t) \ (= f^{J_2}(t)). \\ & && \text{(by (P1), (P3), (P4))} \end{aligned}$$

Therefore, we have shown that $f^{\hat{J}}(t) = max f^{J_1}(t), f^{J_2}(t)$ holds for all t and the theorem is proved. □

7.2 Preservation of Schedulability Properties by the Transformation

As we have seen, the transformation from a conditional task to an equivalent unconditional task preserves the vol and len parameters, as well as the work and remaining demand functions. Therefore, it allows to approximately test schedulability using the approach for unconditional tasks presented in Section 6.

We note, however, that the transformation does not preserve schedulability. That is, it may happen that the conditional task is schedulable where the unconditional is not. This is rather intuitive, since it may happen that the volume of the conditional task occurs on a completely different conditional branch of the graph than the one on which the critical path occurs, whereas the equivalent unconditional task combines these two in one graph.

An example is a conditional task consisting of only one conditional construct with, in one branch, 3 parallel jobs with wcet 8, and in the other branch a single job with wcet 12. If the deadline is 12, then whatever branch is taken, a preemptive schedule is always able to complete the jobs on 2 machines within the deadline. The transformation yields an equivalent unconditional task that starts with 3 parallel jobs with wcet 6 all preceding a single job with wcet 6. Clearly, this task cannot be scheduled within the deadline on 2 machines.

The transformation neither preserves non-schedulability. As an example, think of the conditional task consisting of only one conditional construct with in one branch 3 parallel jobs with wcet 8 all preceding a job with wcet 9. In the other branch, there are two parallel jobs with wcet 20 each. If the deadline is 20, then if the first branch is taken it cannot meet the deadline on 2 machines. However, within a S_∞ schedule the second branch has always the highest remaining demand. Thus, the equivalent unconditional task consists simply of the second branch of the conditional task, which is in fact schedulable within the deadline on two machines.

7.3 Time Complexity of the Transformation

Before concluding this section, we briefly discuss the time complexity of the transformation procedure. We bound the time complexity in terms of two parameters: the size (number of nodes and edges) of the DAG $\mathcal{G}$ being transformed, which we denote by $|\mathcal{G}|$, and the maximum nesting level of the conditionals in $\mathcal{G}$, which we denote by $\nu(\mathcal{G})$. Trivially, $\nu(\mathcal{G}) \leq |\mathcal{G}|$, but typically we expect $\nu(\mathcal{G})$ to be much smaller than $|\mathcal{G}|$.

Theorem 7.4. *The time complexity of the transformation algorithm applied to a conditional DAG $\mathcal{G}$ is bounded above by some polynomial in $2^{\nu(\mathcal{G})} \cdot |\mathcal{G}|$.*

Proof. Recall that the transformation algorithm in Section 7.1 operates in phases, by repeatedly transforming one conditional construct at a time (an innermost one), thus producing a sequence of equivalent DAGs $\mathcal{G}^{(0)}(=\mathcal{G}), \mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \ldots, \mathcal{G}^{(k)}, \ldots$ with less and less conditional constructs, until an unconditional DAG $\hat{\mathcal{G}}$ is arrived at. The proof of the claim is based on the following observations:

- In each phase, exactly one conditional construct is removed, therefore the number of intermediate equivalent DAGs produced is at most the number of conditional constructs of $\mathcal{G}$ (which is clearly less than the size of $\mathcal{G}$);
- Each step (1), (2), and (3) of a transformation phase described in Section 7.1 can be carried out in time that is polynomial in the size of the conditional sub-DAG being transformed in that phase. This is easily seen for steps (1) and (3); for step (2), note that the complexity

is dictated by the number of linear pieces of the upper envelope of the rdem functions of the two sub-DAGs spanned by V_1' and V_2', which is at most twice the size of the conditional sub-DAG being replaced;

— Thus, the size of $\mathcal{G}^{(k+1)}$ is at most twice the size of $\mathcal{G}^{(k)}$. Moreover, as long as $\mathcal{G}^{(k+1)}, \mathcal{G}^{(k+2)}, \ldots, \mathcal{G}^{(k')}$ are obtained by transforming conditional constructs that are not nested in $\mathcal{G}^{(k)}$, the size of $\mathcal{G}^{(k')}$ is at most twice the size of $\mathcal{G}^{(k)}$. It follows that the size of $\hat{\mathcal{G}}$, and of each intermediate DAG $\mathcal{G}^{(k)}$, is at most $2^{v(\mathcal{G})}$ times the size of $\mathcal{G}$. The claim follows. □

While the bound in Theorem 7.4 may be exponentially large for dag-tasks for which $v(\mathcal{G})$ is comparable to $|\mathcal{G}|$, note that alternative approaches based on enumerating execution flows have an exponential complexity even when $v(\mathcal{G}) = 1$ (recall the example in Section 2.3). In contrast, whenever $v(\mathcal{G})$ is constant, Theorem 7.4 provides a polynomial-time bound.

8 SIMPLER SUFFICIENT CONDITIONS FOR SCHEDULABILITY

We complement the results of the previous sections with two simpler sufficient conditions for EDF- and DM-schedulability, respectively, that can be easily checked in polynomial time.

In this section we assume, without loss of generality, that the DAG tasks τ_i are ordered according to nondecreasing D_i (breaking ties arbitrarily). Since all conditions are stated only in terms of the parameters $\text{len}(\mathcal{G}_i)$, $\text{vol}(\mathcal{G}_i)$, D_i, T_i, they apply equally well to unconditional and conditional DAG task sets.

8.1 EDF-Schedulability

Theorem 8.1. *Let $\mathcal{T} = (\tau_1, \ldots, \tau_n)$ be a DAG task set satisfying the following conditions, for some $\delta \in (0, 1]$:*

(i) $\text{len}(\mathcal{G}_k) \le \delta D_k$, $k = 1, 2, \ldots, n$,
(ii) *for each k, $k = 1, 2, \ldots, n$, either*

$$\sum_{i:T_i \le D_k} \frac{\text{vol}(\mathcal{G}_i)}{T_i} + \sum_{i:T_i > D_k} \frac{\text{vol}(\mathcal{G}_i)}{2D_k} \le \frac{(1-\delta)m + \delta}{2}$$

or

$$\sum_{i:T_i \le D_k} \frac{\text{vol}(\mathcal{G}_i)}{T_i} + \sum_{i=1}^{n} \frac{\text{vol}(\mathcal{G}_i)}{D_k} \le (1-\delta)m + \delta.$$

Then $\mathcal{T}$ is EDF-schedulable on m unit-speed processors.

Proof. Suppose by contradiction that EDF fails to complete a dag-job of a task τ_k. Let j be the first dag-job of task τ_k that misses its deadline d_j. W.l.o.g., we assume that there are no dag-jobs with a deadline later than d_j. Consider the interval $I = [r_j, d_j)$. Denote by x the total amount of time during I where all processors are busy. Let $y = (d_j - r_j) - x = D_k - x$, i.e., y denotes the total amount of time in I during which not all processors are busy.

We first observe that $y \le \delta D_k$. This follows from the observation that whenever a processor is idle, EDF must be executing a job belonging to the longest chain of the last dag-job released by τ_k and, hence, $y \le \text{len}(\mathcal{G}_k)$, which is assumed to be at most δD_k (condition (i)).

Condition $y \le \delta D_k$ implies that $x \ge (1-\delta)D_k$. Now, since the total amount of execution occurring over the interval I is greater than or equal to $(mx + y)$, we conclude that the total work done by EDF during I is at least $((1-\delta)m + \delta)D_k$.

Now recall inequality Equation (3) from Section 6 and observe that the total amount of work due in I is bounded above by

$$\begin{aligned}
&\sum_{i:T_i \le D_k} \left\lceil \frac{D_k}{T_i} \right\rceil \mathrm{vol}(\mathcal{G}_i) + \sum_{i:T_i > D_k} \mathrm{vol}(\mathcal{G}_i) \\
&\le 2D_k \left(\sum_{i:T_i \le D_k} \frac{\mathrm{vol}(\mathcal{G}_i)}{T_i} + \sum_{i:T_i > D_k} \frac{\mathrm{vol}(\mathcal{G}_i)}{2D_k} \right) \\
&\le ((1-\delta)m + \delta)D_k,
\end{aligned}$$

or, in case the alternative assumption is satisfied,

$$\begin{aligned}
&\sum_{i:T_i \le D_k} \left\lceil \frac{D_k}{T_i} \right\rceil \mathrm{vol}(\mathcal{G}_i) + \sum_{i:T_i > D_k} \mathrm{vol}(\mathcal{G}_i) \\
&\le D_k \left(\sum_{i:T_i \le D_k} \frac{\mathrm{vol}(\mathcal{G}_i)}{T_i} + \sum_{i=1}^{n} \frac{\mathrm{vol}(\mathcal{G}_i)}{D_k} \right) \\
&\le ((1-\delta)m + \delta)D_k,
\end{aligned}$$

where we have used either of condition (ii) and the fact that $\lceil z \rceil \le 2z$ when $z \ge 1$. This contradicts the assumption that EDF fails and completes the proof of the theorem. □

8.2 DM-Schedulability

Theorem 8.2. *Let $\mathcal{T} = (\tau_1, \tau_2, \ldots, \tau_n)$ be a DAG task set satisfying the following conditions, for some $\delta \in (0, 1]$:*

(i) $\mathrm{len}(\mathcal{G}_k) \le \delta D_k$, $k = 1, 2, \ldots, n$,
(ii) *for each k, $k = 1, 2, \ldots, n$, either*

$$\sum_{i:T_i \le 2D_k} \frac{\mathrm{vol}(\mathcal{G}_i)}{T_i} + \sum_{i:T_i > 2D_k} \frac{\mathrm{vol}(\mathcal{G}_i)}{4D_k} \le \frac{(1-\delta)m + \delta}{4}$$

or

$$\sum_{i:T_i \le 2D_k} \frac{\mathrm{vol}(\mathcal{G}_i)}{T_i} + \sum_{i=1}^{n} \frac{\mathrm{vol}(\mathcal{G}_i)}{2D_k} \le \frac{(1-\delta)m + \delta}{2}.$$

Then $\mathcal{T}$ is DM-schedulable on m unit-speed processors.

Proof. Suppose by contradiction that DM fails to complete a dag-job of a task τ_k. Let j be the first dag-job of task τ_k that misses its deadline d_j in the minimal instance J that violates the theorem, i.e., J is an instance with the smallest number of dag-jobs that violates the theorem. W.l.o.g., assume that there are no dag-jobs with a deadline later than $2d_j - r_j$.

Consider the intervals $\hat{I} = [r_j, d_j)$ and $I = [r_j, 2d_j - r_j)$; the crucial observation in this case is that, during $\hat{I}$, DM processes jobs that have their deadline in I.

Denote by x the total amount of time during $\hat{I}$ when all processors are busy according to the DM schedule. Let $y = (d_j - r_j) - x = D_k - x$, i.e., y denotes the total amount of time in $\hat{I}$ during which not all processors are busy.

We first observe that $y \le \delta D_k$. This follows from the observation that whenever a processor is idle, DM must be executing a job belonging to the longest chain of the last dag-job released by τ_k and hence $y \le \mathrm{len}(\mathcal{G}_k)$, which is assumed to be at most δD_k.

Condition $y \leq \delta D_k$ implies that $x \geq (1-\delta)D_k$. Now, since the total amount of execution occurring over the interval $\hat{I}$ is greater than or equal to $(mx + y)$, we conclude that the total work done by DM during $\hat{I}$ is greater than or equal to $((1-\delta)m + \delta)D_k$.

Again recall inequality Equation (3) from Section 6 and observe that the total amount of work due in I is bounded above by

$$\begin{aligned}
&\sum_{i:T_i \leq 2D_k} \left\lceil \frac{2D_k}{T_i} \right\rceil \mathrm{vol}(\mathcal{G}_i) + \sum_{i:T_i > 2D_k} \mathrm{vol}(\mathcal{G}_i) \\
&\leq 4D_k \left(\sum_{i:T_i \leq 2D_k} \frac{\mathrm{vol}(\mathcal{G}_i)}{T_i} + \sum_{i:T_i > 2D_k} \frac{\mathrm{vol}(\mathcal{G}_i)}{4D_k} \right) \\
&\leq ((1-\delta)m + \delta)D_k,
\end{aligned}$$

or, in case the alternative assumption is satisfied,

$$\begin{aligned}
&\sum_{i:T_i \leq 2D_k} \left\lceil \frac{2D_k}{T_i} \right\rceil \mathrm{vol}(\mathcal{G}_i) + \sum_{i:T_i > 2D_k} \mathrm{vol}(\mathcal{G}_i) \\
&\leq 2D_k \left(\sum_{i:T_i \leq 2D_k} \frac{\mathrm{vol}(\mathcal{G}_i)}{T_i} + \sum_{i=1}^{n} \frac{\mathrm{vol}(\mathcal{G}_i)}{2D_k} \right) \\
&\leq ((1-\delta)m + \delta)D_k,
\end{aligned}$$

where we have used either of condition (ii) and the fact that $\lceil 2z \rceil \leq 4z$ when $z \geq 1/2$. This contradicts the assumption that DM fails and completes the proof of the theorem. □

9 TASKS WITH INTRA-TASK DEADLINES

In this section, we present a schedulability test for unconditional tasks with intra-task deadlines. Throughout this section we assume that the given dag-tasks do not have any conditional vertices. Additionally, we extend our model by defining a specific deadline for each job of the task. Formally, we assume that each task τ_i is specified by a tuple $(\mathcal{G}_i, T_i)$ where as before T_i is a positive integer denoting the period length of the task. Since we do not allow conditional vertices, the DAG for each task τ_i is specified as $\mathcal{G}_i = (R_i, E_i)$ where each vertex $v \in R_i$ corresponds to a sequential operation (a job in our terminology) that is characterized by a processing time $e_v \in \mathbb{N}$ and now additionally a relative deadline D_v. If a dag-job of task τ_i is released at time instant t, then each job $v \in R_i$ constituting it must complete execution by time $t + D_v$.

In this model, we can assume without loss of generality that the jobs generated by a task are consistent, meaning that whenever there are two jobs $v, v' \in R_i$ such that v is a predecessor of v', then $D_v \leq D_{v'}$. If this was not the case, then we could simply change D_v to $D_{v'}$ without affecting schedulability.

Similarly as in Section 6, we need to check whether the following two conditions are satisfied for any collection of jobs J that can be released by the given task set:

- —**Condition 1:** J is S_∞-schedulable, and
- —**Condition 2:** there is no interval I during which every valid schedule for J must finish more than $(\alpha m - m + 1) \cdot |I|$ units of work.

We explain now how to check these conditions.

Condition 1. It suffices to check for every task whether one single dag-job is correctly scheduled on S_∞. To this end, we need to check for each job $v \in R_i$ of a dag-task τ_i whether all its predecessors

finish before time $D_v - e_v$ on S_∞. This can be done easily in polynomial time by simulating S_∞'s schedule; we omit the details.

Condition 2. As in Section 6, we analyze the workload density that an interval can have and define, for any collection J of jobs that might be generated from task set $\mathcal{T}$, and any interval I, $\text{work}^J(I)$ as the amount of work done by S_∞ during I on the jobs of J that are released during I and have their deadlines in I. Note that it might be that of two jobs in J belonging to the same dag-job, one of them contributes to $\text{work}^J(I)$ and the other does not. We define $\text{work}_i(t)$, $\text{work}_\mathcal{T}(t)$, and $\lambda_\mathcal{T}$ accordingly: $\text{work}_i(t)$ denotes the maximum amount of work that may be done by S_∞ on jobs of task τ_i that are due in an interval of length t (i.e., the maximum workload caused by task τ_i during an interval of length t). Similarly, $\text{work}_\mathcal{T}(t)$ denotes the same for the set of tasks $\mathcal{T}$.

Approximation of $\lambda_\mathcal{T}$. As before, we define $\lambda_\mathcal{T} = \sup_{t\in\mathbb{N}}(\text{work}_\mathcal{T}(t)/t)$, which can be written as

$$\lambda_\mathcal{T} = \sup_{t\in\mathbb{N}} \frac{\sum_{i=1}^{n} \text{work}_i(t)}{t}.$$

Again, since it is coNP-hard to compute $\lambda_\mathcal{T}$ exactly (Eisenbrand and Rothvoss 2010), we compute an approximation $\hat{\lambda}_\mathcal{T}$ instead. In the setting of intra-task deadlines, this maximum is achieved when the deadline of some *job* of a dag-job of τ_i coincides with the rightmost endpoint of the interval and the other dag-jobs of τ_i are released as closely as possible. As before, this is the case because if this was not true, then we could increase the release time of the last dag-job without decreasing $\text{work}_i(t)$, and the same argumentation works in case that the other dag-jobs of τ_i are not released as closely as possible.

Computing $\text{work}_i(t)$ is now more complicated than before. Even though we know that the maximum workload in an interval is achieved when *some* job v of a dag-job has its deadline on the rightmost endpoint of the interval, we do not know which one this is. Even more, for different values of t this job might be different. Therefore, when computing $\text{work}_i(t)$ we have to try all jobs of τ_i as a candidate v. This is shown in Algorithm 5, which, for a given t, iterates over all $v \in R_i$. For each candidate v, the algorithm computes (similarly to Algorithm 1) the maximum work performed by S_∞ in $[t_0, t_0 + t]$ for jobs due before $t_0 + t$, assuming that v has its deadline at $t_0 + t$; finally, it returns the highest value found.

ALGORITHM 5: Workload computation for intra-task deadlines

```
work_i(t):
    W_max ← 0
    foreach v ∈ R_i
        do
            W ← 0
            for t' ← 1 to t
                do
                    W ← W + B_v(t_0 + t − t')
                    (B_v(x) = n. of busy processors in [x, x + 1) during
                    S_∞'s schedule of a backward-aligned sequence
                    where an instance of job-vertex v has its deadline at time t_0 + t)
            if W > W_max
              then W_max ← W
    return W_max
```

As before, it suffices to approximately compute $\sup_{t\in\mathbb{N}} \text{work}_i(t)/t$ for each task τ_i. For a dag-job of τ_i we define $D_i^{\max} := \max_{v\in R_i} D_v$. Then, the statements of Lemma 6.9 hold accordingly and we have that

$$\text{work}_i(t) \geq \max\left(\left\lfloor \frac{t + T_i - D_i^{\max}}{T_i} \right\rfloor, 0\right) \cdot \text{vol}(\mathcal{G}_i), \tag{10}$$

$$\text{work}_i(t) \leq \left\lceil \frac{t}{T_i} \right\rceil \cdot \text{vol}(\mathcal{G}_i). \tag{11}$$

As in Section 6, we define the function $\hat{\text{w}}_i$ for each task τ_i by

$$\hat{\text{w}}_i(t) = \begin{cases} \text{work}_i(t) & \text{if } t \leq T_i/\epsilon + (1+1/\epsilon)D_i^{\max} \\ \frac{t-D_i^{\max}}{T_i}\,\text{vol}(\mathcal{G}_i) & \text{if } t > T_i/\epsilon + (1+1/\epsilon)D_i^{\max} \end{cases},$$

and we define $\hat{\text{w}}(t) := \sum_{i=1}^n \hat{\text{w}}_i(t)$for all t.

Lemma 9.1. *For all $t \in \mathbb{N}$ it holds that*

$$\frac{1}{1+\epsilon}\text{work}(t) \leq \hat{\text{w}}(t) \leq \text{work}(t).$$

Proof. Consider a task τ_i. As in Lemma 6.13, we can show that $\text{work}_i(t) \geq \hat{\text{w}}_i(t)$, since for all $t > T_i/\epsilon + (1+1/\epsilon)D_i^{\max}$, by Equation (10),

$$\begin{aligned} \frac{\text{work}_i(t)}{\text{vol}(\mathcal{G}_i)} &\geq \left\lfloor \frac{t + T_i - D_i^{\max}}{T_i} \right\rfloor \geq \frac{t + T_i - D_i^{\max}}{T_i} - 1 \\ &= \frac{t - D_i^{\max}}{T_i} = \frac{\hat{\text{w}}_i(t)}{\text{vol}(\mathcal{G}_i)}. \end{aligned}$$

Moreover, again for $t > T_i/\epsilon + (1+1/\epsilon)D_i^{\max}$, we have, using Equation (11),

$$\begin{aligned} \frac{\text{work}_i(t)}{\hat{\text{w}}_i(t)} &\leq \frac{\lceil t/T_i \rceil}{\frac{t-D_i^{\max}}{T_i}} \leq \frac{t/T_i + 1}{t/T_i - D_i^{\max}/T_i} \\ &= \frac{t + T_i}{t - D_i^{\max}} \leq \frac{(D_i^{\max} + T_i)/\epsilon + D_i^{\max} + T_i}{(D_i^{\max} + T_i)/\epsilon + D_i^{\max} - D_i^{\max}} = 1 + \epsilon. \end{aligned}$$

This proves that $\frac{1}{1+\epsilon}\text{work}_i(t) \leq \hat{\text{w}}_i(t) \leq \text{work}_i(t)$ for all t. Using $\text{work}(t) = \sum_{i=1}^n \text{work}_i(t)$ and $\hat{\text{w}}(t) = \sum_{i=1}^n \hat{\text{w}}_i(t)$, the claim of the lemma follows. □

For our pseudopolynomial time test observe that $\hat{\text{w}}(t)$ can change its slope only on integral time points. Thus, it is piecewise linear with at most $T_i/\epsilon + (1+1/\epsilon)D_i^{\max} + 1$ many linear pieces. Using Lemma 6.12, we compute the value $\hat{\lambda}_{\mathcal{T}} := \sup_{t\in\mathbb{N}} \hat{\text{w}}(t)/t$ in time $O(\frac{1}{\epsilon}(T_i + D_i^{\max}))$. Then, $\hat{\lambda}_{\mathcal{T}}$ is a $(1+\epsilon)$-approximation to $\lambda_{\mathcal{T}}$ that we computed in pseudopolynomial time. Lemma 6.5 also holds in the context of intra-task deadlines, and thus we obtain the following theorem.

Theorem 9.2. *Let $\epsilon > 0$. There is a pseudopolynomial time EDF-schedulability test with speedup $2 - 1/m + \epsilon$, and a pseudopolynomial time DM-schedulability test with speedup $3 - 1/m + \epsilon$ for sets of unconditional dag-tasks with intra-task deadlines.*

10 CONCLUSIONS AND FUTURE WORKS

In this article, we have closed the gap between feasibility analysis for the sequential sporadic task model and that of its parallel generalization, in which each sporadic task is modeled as a DAG of precedence constraints and/or conditional structures. We have shown that, even for dag-tasks, global EDF has a tight speedup bound of $2 - 1/m$, where m is the number of processors, while DM has a speedup bound of at most $3 - 1/m$. We have also presented polynomial and pseudopolynomial time tests for determining whether a set of sporadic dag-tasks can be scheduled by EDF or DM to meet all deadlines on a specified number of processors. It is remarkable that the speedup bound of the pseudopolynomial time test matches that of the best EDF- and DM-schedulability tests known for ordinary (sequential) sporadic task sets, see Baruah et al. (2010) and Bonifaci et al. (2012), for implicit-deadline and constrained-deadline tasks.[4] This suggests that better speedup bounds can only be achieved by algorithms with a higher degree of sophistication than global EDF.

To handle conditional tasks, we exhibited a general transformation to unconditional tasks that preserves the relevant quantities for our tests, thus allowing to carry over the schedulability analysis. We have also provided faster sufficient polynomial time schedulability tests and an extension of the model to tasks with intra-task deadlines.

An interesting direction for future work is to provide speedup bounds for sufficient schedulability tests based on simpler conditions, such as the polynomial time schedulability tests that we proposed in Section 8. While very efficient schedulability tests for implicit-deadline dag-tasks have been analyzed in the literature, such as the linear tests by Li et al. (2013, 2014, 2015) and Saifullah et al. (2014), no linear-time or even truly polynomial time tests with bounded speedup are known for constrained or arbitrary-deadline tasks.

ACKNOWLEDGMENT

The authors thank Enrico Bini for very stimulating discussions, and the anonymous reviewers for many useful suggestions.

REFERENCES

Björn Andersson and Dionisio de Niz. 2012. Analyzing global-EDF for multiprocessor scheduling of parallel tasks. In *Proceedings of the 16th International Conference on Principles of Distributed Systems*. Springer, 16–30.

Theodore Baker and Sanjoy Baruah. 2007. Schedulability analysis of multiprocessor sporadic task systems. In *Handbook of Real-Time and Embedded Systems*, Sang H. Son, Insup Lee, and Joseph Y.-T Leung (Eds.). Chapman Hall/CRC Press.

Sanjoy Baruah. 1998. A general model for recurring real-time tasks. In *Proceedings of the Real-Time Systems Symposium*. IEEE Computer Society Press, 114–122.

Sanjoy K. Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Sebastian Stiller. 2010. Improved multiprocessor global schedulability analysis. *Real-Time Syst.* 46, 1 (2010), 3–24.

Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Sebastian Stiller. 2012. A constant-approximate feasibility test for multiprocessor real-time scheduling. *Algorithmica* 62, 3–4 (2012), 1034–1049.

Michael L. Dertouzos. 1974. Control robotics: The procedural control of physical processes. In *Proceedings of the International Federation for Information Processing Congress*. North-Holland, Amsterdam, 807–813.

Friedrich Eisenbrand and Thomas Rothvoss. 2010. EDF-schedulability of synchronous periodic task systems is coNP-hard. In *Proceedings of the 21st Symposium on Discrete Algorithms*, Moses Charikar (Ed.). SIAM, Philadelphia, PA, 1029–1034.

[4] An analogous conclusion cannot be drawn for arbitrary-deadline tasks since in the sequential arbitrary-deadline task model it is typically required that a job complete execution before the subsequent job of the task may begin to execute; in contrast, in the sporadic DAG model we do not require that all jobs of a dag-job complete execution before jobs of the next dag-job of the same task may begin to execute.

José Carlos Fonseca, Vincent Nélis, Gurulingesh Raravi, and Luís Miguel Pinho. 2015. A multi-DAG model for real-time parallel applications with conditional execution. In *Proceedings of the ACM/ SIGAPP Symposium on Applied Computing (SAC'15)*. ACM, 1925–1932.

Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York.

Ronald L. Graham, Eugene L. Lawler, Jan K. Lenstra, and Alexander H. G. Rinnooy Kan. 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discr. Math.* 5 (1979), 287–326.

Karthik Lakshmanan, Shinpei Kato, and Raj Rajkumar. 2010. Scheduling parallel real-time tasks on multi-core processors. In *Proceedings of the IEEE Real-Time Systems Symposium*. IEEE, Los Alamitos, CA, 259–268.

Jan K. Lenstra and Alexander H. G. Rinnooy Kan. 1978. Complexity of scheduling under precedence constraints. *Operat. Res.* 26, 1 (1978), 22–35.

Joseph Y.-T. Leung and M. L. Merrill. 1980. A note on preemptive scheduling of periodic, real-time tasks. *Inf. Process. Lett.* 11, 3 (1980), 115–118.

Joseph Y.-T. Leung and Jennifer Whitehead. 1982. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Perform. Eval.* 2, 4 (1982), 237–250.

Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher D. Gill. 2013. Analysis of global EDF for parallel tasks. In *Proceedings of the Euromicro Conference on Real-Time Systems*. IEEE, 3–13.

Jing Li, Jian-Jia Chen, Kunal Agrawal, Chenyang Lu, Christopher D. Gill, and Abusayeed Saifullah. 2014. Analysis of federated and global scheduling for parallel real-time tasks. In *Proceedings of the Euromicro Conference on Real-Time Systems*. IEEE, 85–96.

Jing Li, Zheng Luo, David Ferry, Kunal Agrawal, Christopher D. Gill, and Chenyang Lu. 2015. Global EDF scheduling for parallel real-time tasks. *Real-Time Syst.* 51, 4 (2015), 395–439.

Chung L. Liu. 1969a. Scheduling algorithms for hard real-time programming of a single processor. *JPL Space Progr. Sum.* 37–60, II (1969), 31–37.

Chung L. Liu. 1969b. Scheduling algorithms for multiprocessors in a hard real-time environment. *JPL Space Progr. Sum.* 37–60, II (1969), 28–31.

Chung L. Liu and James W. Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM* 20, 1 (1973), 46–61.

Aloysius K. Mok. 1983. *Fundamental Fesign Problems of Distributed Systems for the Hard Real-time Environment*. Ph.D. Dissertation. Laboratory for Computer Science, Massachusetts Institute of Technology. Available as Technical Report No. MIT/LCS/TR-297.

Geoffrey Nelissen, Vandy Berten, Joël Goossens, and Dragomir Milojevic. 2012. Techniques optimizing the number of processors to schedule multi-threaded tasks. In *Proceedings of the Euromicro Conference on Real-Time Systems*. IEEE, 321–330.

Cynthia A. Phillips, Clifford Stein, Eric Torng, and Joel Wein. 2002. Optimal time-critical scheduling via resource augmentation. *Algorithmica* 32, 2 (2002), 163–200.

Abusayeed Saifullah, David Ferry, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher D. Gill. 2014. Parallel real-time scheduling of DAGs. *IEEE Trans. Parallel Distrib. Syst.* 25, 12 (2014), 3242–3252.

Abusayeed Saifullah, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher D. Gill. 2013. Multi-core real-time scheduling for generalized parallel task models. *Real-Time Syst.* 49, 4 (2013), 404–435.

Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. 2011. The digraph real-time task model. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, Los Alamitos, CA, 71–80.

Jeffrey D. Ullman. 1975. NP-complete scheduling problems. *J. Comput. Syst. Sci.* 10, 3 (1975), 384–393.

Received February 2017; revised May 2018; accepted December 2018