Feasibility Analysis of Recurrent DAG Tasks is PSPACE-hard

Vincenzo Bonifaci*[1] and Alberto Marchetti-Spaccamela†[2]

[1]Dipartimento di Matematica e Fisica, Università Roma Tre
[2]Dipartimento di Ingegneria Informatica, Automatica e Gestionale "Antonio Ruberti", Sapienza Università di Roma

Abstract

We study a popular task model for scheduling parallel real-time tasks, where the internal parallelism of each task is modeled by a directed acyclic graph (DAG). We show that deciding the feasibility of a set of sporadically recurrent DAG tasks is hard for the complexity class PSPACE, thus ruling out approaches to this problem that rely on Integer Linear Programming or Satisfiability solvers (assuming NP $\neq$ PSPACE).

Keywords: real-time scheduling, computational complexity, precedence constraints, sporadic task system, feasibility testing, parallel computing

*vincenzo.bonifaci@uniroma3.it
†alberto@diag.uniroma1.it

1 Introduction

Directed Acyclic Graphs (DAGs) are widely used in parallel computing to naturally model intra-parallelism and precedence constraints between tasks or processes. In particular, many real-time systems may be modeled as a finite number of independent recurrent *DAG tasks*, where a DAG task is specified in terms of a directed acyclic graph, a minimum inter-arrival time, and a relative deadline. In such a model, the time instant when the next invocation of a task will be released after its minimal inter-arrival time has elapsed is unknown. Thus, there are infinitely many job sequences that conform to the system's specification, making the analysis of such systems challenging.

In this work, we study the *(online) feasibility problem for sets of sporadic DAG tasks*: the problem of determining whether a given real-time workload which is specified by a set of sporadic DAG tasks can be scheduled by some online algorithm to always meet all deadlines upon a given platform of identical processors.

The main contribution of the paper is to show that the feasibility problem for sets of sporadic DAG tasks is highly intractable. Namely, it is proven that determining whether a given set of sporadic DAG tasks can always be feasibly scheduled online on a specified multiprocessor platform is hard for the complexity class PSPACE. The construction proves hardness already in the case of systems consisting of two DAG tasks, the nodes of which have unitary execution time.

The result of the paper has implications for parallel programming and real-time systems design. In particular, under the complexity-theoretic conjecture $\mathsf{NP} \neq \mathsf{PSPACE}$, the result rules out the possibility of effectively using approaches such as Integer Linear Programming (ILP) or Satisfiability (SAT) solvers to address the feasibility problem for sporadic DAG tasks in its general form.

Related work In the precedence-constrained scheduling problem the goal is to schedule a set of jobs with precedence constraints: jobs are represented as nodes of a DAG and there is a directed edge from the vertex J_i to the vertex J_j when the job J_i is an immediate predecessor of the job J_k. It is known that even deciding the feasibility of a collection of unit-length precedence-constrained non-recurrent jobs is *i)* NP-complete in the strong sense when the number of machines is part of the input, *ii)* a long-standing open question if the number of machines is a constant m, $m \geq 3$ [20, 33].

The *(3-parameter) sporadic task model* [4, 28] is a well-established model in real-time systems to model recurrent jobs. In this model a task is represented by a triple (J, D, T), where J is a job, D a positive integer representing the relative deadline of the task, and T a positive integer representing the minimum separation time between two successive jobs of the task.

The sporadic task model allows to naturally model real-time system applications; for this reason, it is at the center of the research efforts of the real-time systems community, see for example Davis and Burns [13] and references therein.

Therefore, we limit ourselves to citing contributions that are most directly connected to the results in this paper.

For sequential sporadic tasks, the exact problem of testing the online feasibility of a task-set on a single processor is known to be NP-hard [14, 15] and to be exactly solvable in exponential time [8] in the general case.

The DAG Task model, originally proposed by Baruah et al. [3, 9], extends the sporadic task model assuming that each task is represented by a DAG. The model has attracted significant attention in the real time community (see [35] and references therein). Significant effort has therefore been devoted to the development of inexact approaches involving some form of resource augmentation, or to the study of special cases of the problem. We briefly review some important special cases of the feasibility problem in Section 2.3. Among the main approaches to resource augmentation are *speedup bounds* [9, 26, 32] and *utilization bounds* or *capacity augmentation bounds* [11, 25, 26].

From a complexity theory perspective, although there is a plethora of scheduling problems that are known to be NP-hard [20, 23], natural scheduling problems that are PSPACE-hard appear to be a rather rare breed [29, 30]. Namely, in [6] the feasibility analysis problem for Conditional Directed Acyclic Graph model [2, 27], that extends the DAG model allowing the execution of conditional (e.g. if-then-else) constructs is proven to be PSPACE-complete.

In [21] the authors studied the complexity of checking whether a sporadic task system is schedulable under a given scheduler; they show that the problem is PSPACE-hard with a reduction from the universality problem for (finite state labeled) automata, that, given a labeled automaton A asks whether A accepts all strings. Namely, given a labeled automaton A, [21] defines a set of sporadic tasks $\mathcal{T}$ *and* an algorithm R and proves that $\mathcal{T}$ is schedulable using algorithm R if and only if automaton A verifies the universality property. Observe that the definition of the scheduler *depends on the input automaton*: for two different labeled automata A_1 and A_2 the reduction defines two different task sets S_1, S_2 and two different algorithms R_1 and R_2. It follows that the result does not imply that the feasibility problem of scheduling a sporadic task system is PSPACE-hard.

We believe DAG task feasibility is a natural problem representing an interesting addition to the family of PSPACE-hard problems.

Overview of the sections In Section 2 we define the task model, informally review the relevant definitions from complexity theory, and discuss related results and special cases. In Sections 3 and 4 we state and prove our main result, the PSPACE-hardness of the feasibility analysis problem for DAG tasks. Finally, some open problems are suggested in Section 5.

2 Preliminaries

2.1 Problem definition

We consider a given set S of independent *DAG tasks* (or simply *tasks*) $\tau_1, \ldots, \tau_n$. Each task τ_i consists of a triple (G_i, D_i, T_i) where G_i is a node-weighted directed acyclic graph (DAG), and D_i and T_i are positive integers. Each DAG G_i represents a parallel task. Each node $v \in V(G_i)$ in the DAG represents a basic sequential operation or *job*, and arcs represent precedence relations between jobs. That is, if (u, v) is an arc of G_i then job v cannot be started before job u is completed. Each job u has an associated worst-case execution time $c(u)$; in this work, we assume $c(u) = 1$ (unit-size jobs).

Each DAG task can be activated recurrently; each activation of the task is called a *dag-job*. Parameters D_i and T_i represent, respectively, the *relative deadline* and *minimum inter-arrival time* (or *period*) of task τ_i. The meaning is that an instance (i.e., a dag-job) of τ_i released at time t shall be completed by time $t + D_i$, and no subsequent instance will be released before time $t + T_i$. We assume $D_i \leq T_i$ (*constrained deadlines*). Jobs are to be scheduled by a platform of m identical and parallel processors. Task preemptions and migrations (at integer times) are allowed.

If A is a scheduling algorithm, a taskset is called *A-schedulable* on m processors if every arrival sequence of dag-jobs that can be generated by S is correctly scheduled by algorithm A, that is, all dag-jobs are completed within their deadline. A taskset is *(online) feasible* if it is A-schedulable by some online algorithm A. By algorithm A being *online*, we mean that the scheduling decisions of A at any time t can only be based on information attached to jobs released up to time t, and therefore cannot involve information attached to jobs that are released after time t [10,16,34]. Algorithms that do not satisfy this requirement are called *offline*, or *clairvoyant*, and would be unrealistic in many, if not most, applications of real-time scheduling [1, Chapter 3].

We are concerned with the computational complexity of the following *feasibility problem.*

Online feasibility of a set of DAG tasks
Given: a DAG taskset $S = \{(G_i, D_i, T_i)_{i=1}^{n}\}$ and a number of processors $m \geq 1$.
Find: Whether there exists an online algorithm A such that S is A-schedulable on m processors.

We remark that there exists a (sequential) taskset $S^\star$ for which every arrival sequence of jobs can be correctly scheduled, but for which no single online scheduler can produce a correct schedule for every job arrival sequence [17]; therefore, $S^\star$ is infeasible under our definition, as it is not A-schedulable for any online algorithm A (while at the same time being A-schedulable for some clairvoyant A). Therefore, if we were instead concerned with the existence of *any* scheduling algorithm, whether online or clairvoyant, for a given taskset S, the resulting feasibility problem would be distinct than the one defined above, but less interesting from an application perspective, due to the aforementioned application requirement. We refer the reader to [8, Section 3.2] and [17] for a deeper discussion of this point.

Additional useful concepts A few additional concepts are useful when discussing the feasibility of a taskset. The *depth* of a job $v \in V(G_i)$ is the length of a longest path from a source node of G_i to v. The *critical path length* of task τ_i is $\delta_i \overset{\text{def}}{=} \max_P \sum_{u\in P} c(u)$ where P ranges over all directed paths in G_i. The *synchronous arrival sequence* of taskset S is the arrival sequence of dag-jobs where the k-th dag-job of task τ_i is released at time $(k-1)T_i$.

2.2 Complexity classes: P, NP, PSPACE

We recall (informally) the complexity classes referred to in this paper; we refer to standard texts such as Papadimitriou [31] for the formal definitions.

- P is the class of problems that can be solved by algorithms with running time polynomial in the size of their inputs.
- NP is the class of decision problems the solutions of which can be verified by algorithms with running time polynomial in the size of their inputs, i.e., the class of problems admitting polynomial-length certificates and a polynomial-time verifier.
- PSPACE is the class of problems that can be solved by algorithms using an amount of space (memory) that is polynomial in the size of their inputs.

A problem X is *hard* for a computational complexity class $\mathcal{C}$, or *$\mathcal{C}$-hard*, if every problem $Y \in \mathcal{C}$ can be efficiently reduced to X. In our context, a reduction is efficient if it has a polynomial running time. A problem X is *$\mathcal{C}$-complete* if it belongs to $\mathcal{C}$ and it is $\mathcal{C}$-hard.

The *Quantified Boolean Formula* (QBF) problem is the canonical PSPACE-complete problem (like the Satisfiability problem is the canonical NP-complete problem) [31].

Quantified Boolean Formula (QBF)
Given: a fully quantified Boolean formula

$$F \overset{\text{def}}{=} \forall y_1\ \exists x_1\ \forall y_2\ \exists x_2 \ldots,\ \forall y_n\ \exists x_n \bigwedge_{k=1}^{m} \left(\ell_{k,1} \vee \ell_{k,2} \vee \ell_{k,3}\right) \tag{1}$$

where the $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_n$ are Boolean variables and each $\ell_{k,j}$ is one of the x_i or y_i Boolean variables or its negation.
Find: Whether F is true.

The terms $\ell_{k,j}$ are called *literals* and the expression $\left(\ell_{k,1} \vee \ell_{k,2} \vee \ell_{k,3}\right)$ is called a *clause*. Observe that all the variables of a quantified Boolean formula in the above definition are bound by quantifiers; this implies that the formula is either true or false. Note also that we adopt the convention of starting F with a universal quantifier, as opposed to the more usual existential quantifier; this doesn't affect the complexity of QBF.

Two-player games PSPACE-hardness results can often be presented by viewing the problem under study as a game. Consider a game that involves two players performing actions according to some specified rules. The two players alternate their decisions: the first player starts, then the second player follows, followed by the second action of the first player and so on, until one of the players achieves a goal and wins the game. Well-known examples of these sequential games include Tic-Tac-Toe, Go, Chess, Checkers, etc.

A player has a *winning strategy* in the game if, roughly speaking, for every choice of actions of the opponent, the player has some actions leading to a winning condition. Determining if a player has a winning strategy has been shown to be PSPACE-hard for several classic games such as Checkers, Go or Hex [31, Chapter 19].

Indeed, determining if a player has a winning strategy in an n-rounds game is analogous to determining whether a quantified Boolean formula is true. If a_i are Player 1's actions and b_i are Player 2's actions, then Player 2 has a winning strategy in a game if

$$\forall a_1 \ \exists b_1 \ \forall a_2 \ \exists b_2 \ldots, \ \forall a_n \ \exists b_n \ W(a_1, b_1, a_2, b_2, ...) \tag{2}$$

is true, where W is Player 2's winning condition of the game depending on the players' actions. (Here a_i, b_i do not range over Boolean values but are taken from an appropriate domain determined by the specific game considered.)

The online feasibility problem for DAG tasks can be seen as a game where the first player is the environment, that decides the activation times of the tasks, and the second player is the scheduling algorithm, that decides in which order and on which processor each job is scheduled; this idea will be further discussed in Section 3.2.

2.3 Special cases of the feasibility problem

We review some special cases of the feasibility problem for sporadic DAG tasks that are known or direct consequence of known results.

One DAG task, multiple processors In this special case, the problem is polynomial-time solvable for 2 processors [12, 18, 19], open for a fixed number of at least 3 processors [22], and NP-complete when the number of processors is part of the input [24, 33].

Observation 1. *With a single DAG task with constrained deadline, the feasibility problem is equivalent to (non-recurrent) precedence-constrained scheduling of unit-size jobs with a common deadline.*

Proof. Since $D_1 \leq T_1$, at any time there can be only one pending dag-job (as long as no deadline is missed). Hence a feasible schedule exists if and only if a feasible schedule exists for the corresponding instance of non-recurrent precedence constrained scheduling of unit-size jobs with deadline D_1. We remark that this special case is an offline problem: the only information that is discovered online is the release date of each dag-job, which is irrelevant as there is only a single DAG task. □

One processor, multiple DAG tasks In this case, the problem is coNP-complete (i.e., the *in*feasibility problem is NP-complete) [7,15]. The taskset is feasible if and only if its synchronous arrival sequence is schedulable [4,7].

Observation 2. *With a single processor, the feasibility problem is equivalent to sequential sporadic task scheduling with polynomially bounded execution times.*

Proof. Given an instance I on DAG tasks $G_1, \ldots, G_n$, construct an instance I' on sequential tasks $\tau_1, \ldots, \tau_n$, where the deadline and periods of each task is the same as in I, and the execution time is the number of nodes of the original DAG task. A feasible schedule for I' can be turned into a feasible schedule for I by transforming the processing of the j-th unit of a task τ_i of I' into the processing of the j-th node of an arbitrary (but fixed) topological ordering of G_i. Similarly, any feasible schedule for I can be transformed into a feasible schedule for I'. Note that the total length of tasks is polynomially bounded in the size of I due to the assumption of unit-size jobs.

The opposite reduction is trivial: just associate to each task τ_i a chain G_i of the same length, with the same deadline and period (in fact, in place of a chain we could use any arbitrary DAG with the same number of nodes as the execution time of τ_i).

Finally we remark that this special case too is equivalent to an offline problem, due to the property that the taskset is feasible if and only if its synchronous arrival sequence is schedulable [4,7], and the fact that all information about the synchronous arrival sequence is available before scheduling time. □

3 A polynomial-time reduction from Quantified Boolean Formula

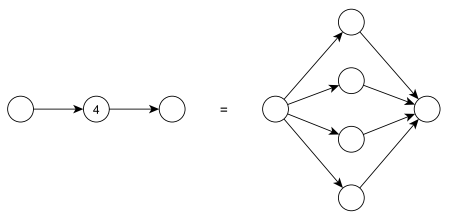
The following theorem is the main result of this paper.

Theorem 1. *The DAG task feasibility problem is PSPACE-hard.*

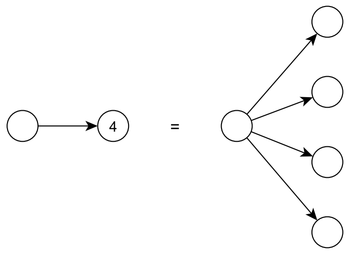
In this section we detail a reduction from the Quantified Boolean Formula (QBF) problem and we provide a high level intuition of the proof that casts DAG task feasibility in the two-player game framework discussed in Section 2.2. In the next section, we will prove the correctness of the reduction.

In the sequel, for any integer $a \geq 2$, the symbol $\textcircled{a}$ in the figures denotes a parallel nodes, each of unit execution time, with identical predecessors and successors (Figure 1a).

Given a formula F with $2n$ variables (of which n are universally quantified and n are existentially quantified) and m clauses, we define a task set $I(F)$ of two sporadic DAG tasks, H and G_F, to be executed on a set of $m' \overset{\text{def}}{=} 3+7m+3q$



(a)



(b)

Figure 1: (a) Representation of sub-dag $\textcircled{a}$ when $a = 4$; (b) Illustration of DAG task H when $q = 4$

identical processors, where $q \overset{\text{def}}{=} 10m$. Without loss of generality, we assume that $n, m \geq 2$.

All nodes of H and G_F have unit execution time. Task G_F encodes the formula F, while H does not depend on F (but it is used to reduce the scheduling options for G_F).

Namely, task H consists of a source node having q successors (Figure 1b); the relative deadline of H is 2, and its period is 8. Since the depth of H is equal to the relative deadline of H, it follows that all nodes of H must be immediately scheduled as soon as they are ready for execution by any correct scheduler; this fact will influence the scheduling decisions for G_F. We will see that certain release patterns of jobs of task H can be used to model the truth assignment of universally quantified variables of F, and will constrain the scheduling decisions for G_F.

3.1 Encoding the formula

Task G_F encodes the formula F and consists of three interconnected sub-dags A, B, C: Task G_F depends on the formula F; it has deadline $D = 10n + 1$, period $T = D$ and consists of three interconnected sub-dags A, B, C.

A high-level view of task G_F is shown in Figure 2. Before entering into the details of the construction, let us highlight the intuitive role of the sub-dags A, B, C:

- Sub-dag A encodes the variables of F. It consist of n sequentially concatenated blocks $A_1, A_2, \ldots, A_n$, one for each pair of variables x_i, y_i of the QBF formula; the last block is followed by a final sink node V.
- Sub-dag B encodes the clauses of F. For each clause of F, B contains seven chains of length $D - 1$; each chain will have three incoming arcs from nodes of sub-dag A, among the arcs represented in red in Fig. 3. The presence of these arcs (precedence constraints) might delay execution of the chains of B. We will show that in a feasible schedule of F there

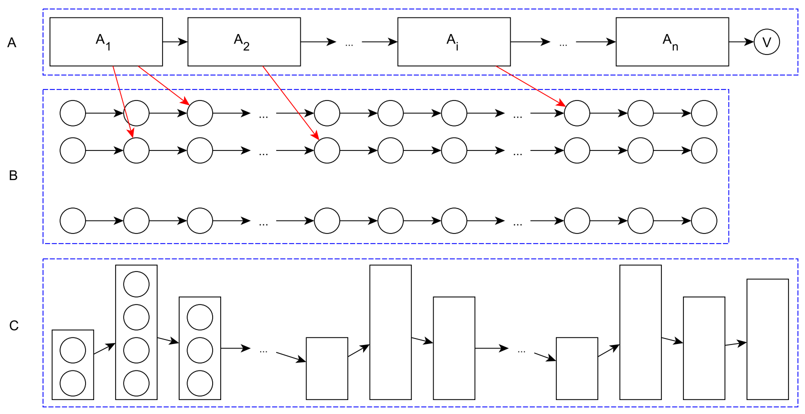


Figure 2: High-level view of the task G_F and its sub-dags A, B, C

are exactly m chains that are not delayed, one per clause, and that these chains represent a satisfying assignment for F.

- Sub-dag C will be defined to globally control the number of available processors, constraining the scheduling decisions for the rest of the system.

Representing the variables Sub-dag A encodes the variables of F and is formed by a chain of blocks A_i, $i = 1, 2, \ldots, n$; each block has a source and a sink and there is a directed arc from the sink of A_i to the source of A_{i+1}; the sink of A_n is connected to a special node V that is the sink of A. It follows that the schedule of block A_i must start after the schedule of block A_{i-1} is completed. Block A_i, shown in Fig. 3, consists of two parts: $A_{i,y}$, that encodes the truth values of variable y_i, and $A_{i,x}$ that encodes the truth value of x_i. Since the critical path of each block A_i is 10, it follows that the critical path of A is $10n + 1 = D$. Figure 3 shows four distinguished nodes in block A_i: $X_i, \neg X_i, Y_i, \neg Y_i$; these nodes have directed arcs (shown in red) to nodes of sub-dag B that we detail below, in the paragraph "Representing the clauses".

Intuition Before completing the presentation of G_F we motivate the definition of blocks A_i, $i = 1, 2, \ldots, n$. Informally speaking the intended interpretation is that assigning True (False respectively) to the boolean variable x_i corresponds to having completed the scheduling of vertex X_i ($\neg X_i$ respectively) by a certain point in time; a similar correspondence applies to the boolean variables Y_i and $\neg Y_i$.

Since G_F is released at time 0 and the critical path of A is equal to the deadline, it follows that in any correct scheduling of G_F block A_i must be executed over the interval $[10(i-1), 10i)$ for each i, $i = 1, 2, \ldots, n$. Moreover, both nodes Y_i and $\neg Y_i$ must complete in the interval $[10(i-1)+4, 10(i-1)+5)$

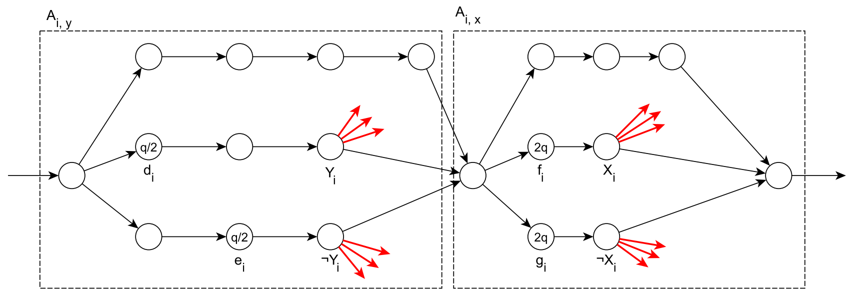


Figure 3: Block A_i; the red arcs connect to nodes of sub-dag B

and both nodes X_i and $\neg X_i$ must complete in the interval $[10(i-1)+8, 10(i-1)+9)$.

We will ensure (see Section 4 for a formal proof) that in any feasible schedule:

1. Only one node between Y_i and $\neg Y_i$ may complete by time $10(i-1)+4$ and the other one completes by time $10(i-1)+5$. Which of the two completes earlier may be forced by the release date of a job of Dag task H: if a job of H is released at time $10(i-1)$ (resp., $10(i-1)+1$) then in order to correctly schedule this job, only $\neg Y_i$ (resp., Y_i) can complete by time $10(i-1)+4$. See Figures 4 and 5.

2. Only one node between X_i and $\neg X_i$ may complete by time $10(i-1)+8$ and the other completes by time $10(i-1)+9$. Which of the two completes earlier is a decision of the scheduler.

Representing the clauses Sub-dag B encodes the clauses of F. The definition of B exploits an observation by Ullman [33] in his proof to show that the (non-recurrent) DAG scheduling problem is NP-complete even when all nodes have unit execution time. A clause with three literals $(\ell_{k,1} \vee \ell_{k,2} \vee \ell_{k,3})$ can be satisfied by seven out of the eight possible truth assignments of the three variables involved. Ullman [33] observed that this is equivalent to asserting that in any satisfying assignment, one and only one of the following seven conjuncts evaluates to true: (i) $\big(\ell_{k,1} \wedge \neg\ell_{k,2} \wedge \neg\ell_{k,3}\big)$, (ii) $\big(\neg\ell_{k,1} \wedge \ell_{k,2} \wedge \neg\ell_{k,3}\big)$, (iii) $\big(\neg\ell_{k,1} \wedge \neg\ell_{k,2} \wedge \ell_{k,3}\big)$, (iv) $\big(\ell_{k,1} \wedge \ell_{k,2} \wedge \neg\ell_{k,3}\big)$, (v) $\big(\ell_{k,1} \wedge \neg\ell_{k,2} \wedge \ell_{k,3}\big)$, (vi) $\big(\neg\ell_{k,1} \wedge \ell_{k,2} \wedge \ell_{k,3}\big)$, and (vii) $\big(\ell_{k,1} \wedge \ell_{k,2} \wedge \ell_{k,3}\big)$. Therefore, a CNF formula with m clauses is satisfiable if and only if it admits a truth assignment that satisfies exactly m conjuncts (one per clause).

For each clause C_j in formula F, we define seven chains, $B_{j,k}$, $k = 1, 2, \ldots, 7$; each chain has $D-1$ nodes. Similarly to [5], each chain has three incoming arcs from sub-dag A in correspondence to the three nodes representing the literals occurring in k-th conjunct of clause j. Namely,

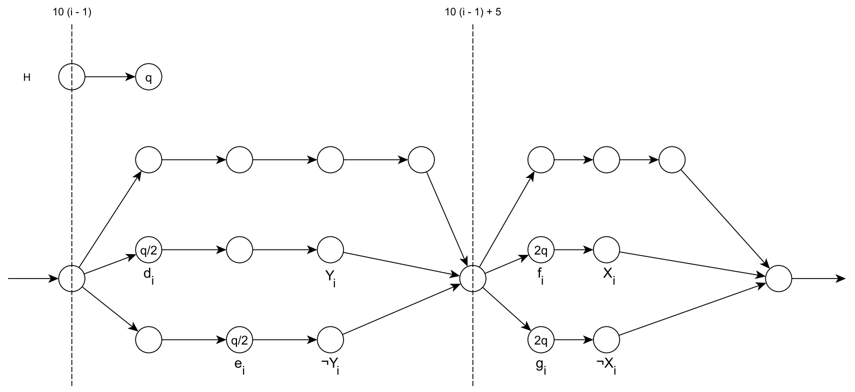


Figure 4: Case in which task H is released at time $10(i-1)$

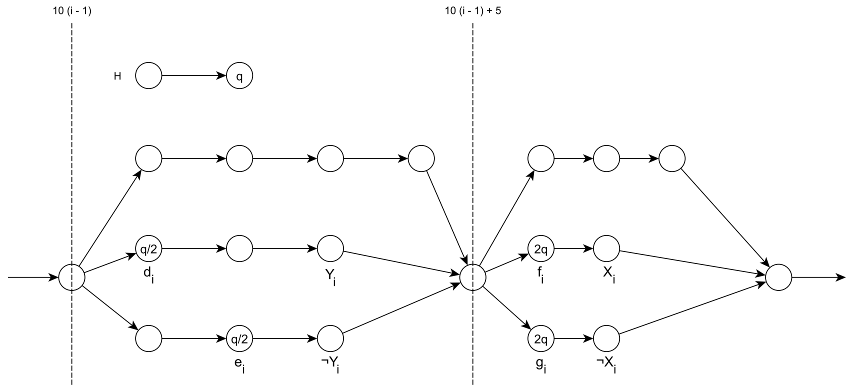


Figure 5: Case in which task H is released at time $10(i-1)+1$

- if literal x_i (respectively, $\neg x_i$) occurs in $B_{j,k}$ then there is an arc from node X_i (resp., $\neg X_i$) of A_i to the $10(i-1)+9$ node of $B_{j,k}$;
- if literal y_i (resp., $\neg y_i$) occurs in $B_{j,k}$ then there is an arc from node Y_i (resp., $\neg Y_i$) of A_i to the $10(i-1)+5$ node of $B_{j,k}$.

It follows that the number of outgoing arcs from node Y_i (resp., $\neg Y_i, X_i, \neg X_i$) is equal to the number of occurrences of literal y_i (resp., $\neg y_i, x_i, \neg x_i$) in F.

In the next section we will show how these arcs influence scheduling of B; namely, scheduling of nodes X_i, $\neg X_i$, Y_i and $\neg Y_i$ might delay execution of the chains of B. We will show that in a feasible schedule of F there are exactly m chains that are not delayed, and that these chains represent a satisfying assignment for F.

Controlling processor availability In order to control the properties of feasible schedules over each interval $[10(i-1), 10i)$, we will restrict the number of available processors for executing subdag A_i. We achieve this goal by defining a sub-dag C that reduces processor availability at each time step in the interval $[0, 10n+1)$.

Subgraph C is a layered graph, with each layer fully connected to the next; the number of layers of C is $D = 10n+1$. Namely, nodes of C can be partitioned into disjoint subsets $C_1, C_2, \ldots, C_D$ such that there is an edge form any node in C_k to any node of C_{k+1}, for each k, $k = 1, 2, \ldots D$. Since D is by definition the relative deadline of G_F, the layers of C will have to be executed in lock-step if the deadline of G_F has to be met.

Let $r = 2 + q + 7m$; the number of nodes in each layer of C is defined so that the number of remaining available processors after processing all nodes of C that are ready for execution is, at time $10(i-1)+t$, $i = 1, 2, \ldots, n$, after a release of G_F as per the following table:

t	0	1	2	3	4	5	6	7	8	9
available processors	$r-1$	r	r	r	r	r	$r+2q$	$r+2q$	r	$r-1$

See Fig. 6 for an illustration of how these available processors compare with each block A_i. These remaining processors can be used for the execution of nodes of sub-dags A, B and of task H. Additionally, the first layer of C includes the source of block A_1.

The last layer of C is defined so that the number of remaining processors in $[D-1, D)$ is $1+q+6m$; we will see that, in a feasible schedule, these processors will execute node V (the sink of sub-dag A), q nodes of task H (if necessary), and the final nodes of $6m$ (out of $7m$) of the chains $B_{j,k}$.

3.2 Scheduling recurrent DAG task sets as a two-player game

Before formally presenting the proof, we present a high level view of the reduction as a two player game between the environment and the scheduler. A

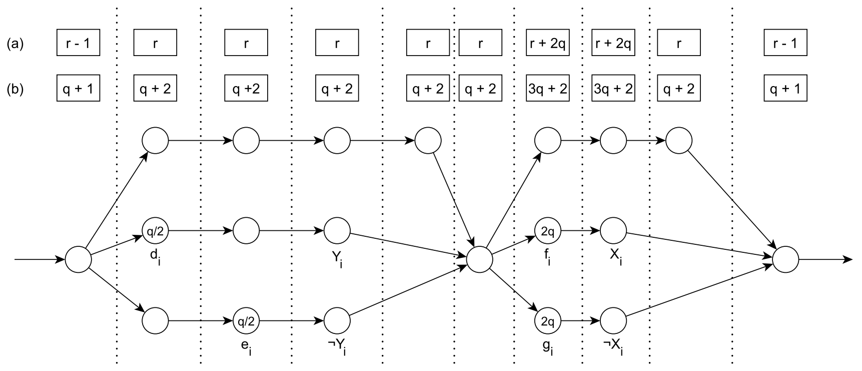


Figure 6: Count of the available processors while each block A_i is pending: (a) available processors after sub-dag C is accounted; (b) remaining processors after sub-dag B and C are accounted

dag-job of G_F is released (without loss of generality) at time 0: the scheduler wins if it is able to schedule G_F (along with all instances of H) within the deadline, while the goal of the environment is to release dag-jobs of task H in such a way as to make it impossible to meet all deadlines. Decisions on release dates of H are taken online by the environment as the schedule of G_F proceeds.

The first move of Player 1 (the environment) is to decide the release date of the first instance of H relative to the release of G_F. We will show that only two possibilities are relevant for the environment player, in the sense that all others can easily be accommodated by the scheduler. Namely, the environment can decide which node between Y_1 and $\neg Y_1$ may complete by time 4, depending on whether a job of H is released at time 0 or at time 1. These two possibilities fix the schedule of block $A_{1,y}$ in two different ways, and correspond to setting the universally quantified variable y_1 of F either `true` or `false`.

After the environment has constrained the schedule of block $A_{1,y}$, Player 2 (the scheduler) decides how to schedule block $A_{1,x}$. Namely, the scheduler has to decide which node between X_1 and $\neg X_1$ may complete by time 8. This decision of the scheduler cannot be influenced by the environment; these two possibilities fix the schedule of block $A_{1,x}$ in two different ways and correspond to setting the truth value of the variable x_1 one way or the other.

The game continues in a similar fashion until sub-dag A is completed: the environment decides the release date of a job in $[10(i-1), 10i)$, $i = 1, \ldots, n$, thus constraining the schedule of $A_{i,y}$, and the scheduler decides how to schedule block $A_{i,x}$ (as in the case $i = 0$, any release outside one of these intervals can be easily accomodated by the scheduler).

In the proof we will show that the schedule of A influences the schedule of sub-dag B. Namely, the scheduling of $A_{i,y}$ and $A_{i,x}$ will possibly delay executions of chains of sub-dag B. Intuitively, the goal of the scheduler (of the environment) is to limit (to increase) the number of delayed chains of sub-dag B.

We will see that at time $10n = D - 1$ there are two possibilities:

1. exactly m chains of B are not delayed by the schedule of sub-dag A. We will show that in this case F is true and G_F can be feasibly scheduled within the deadline independently of release dates of the jobs of task H;

2. at least $m + 1$ chains of B are delayed. In this case we will show that if a job of H is released at time $D - 1$ then F is not true and G_F misses the deadline.

The scheduler has a winning strategy if and only if it can complete the schedule of G_F (along with all instances of H) irrespectively of the online decisions of the environment.

4 PSPACE-hardness of sporadic DAG tasks

Correctness of the reduction – and therefore, Theorem 1 – follows from two implications: (i) if formula F is true, then the sporadic taskset $I(F)$ is feasible, and (ii) vice versa, if the sporadic task set is feasible, then the formula is true. We now detail each of these.

4.1 If F is true then the sporadic task set $I(F)$ is feasible

Lemma 2. *If F is* `true`*, then every arrival sequence from $I(F)$ is schedulable.*

Proof. Assume F is true; we define a run-time scheduling algorithm that completes all jobs within their deadline for any compliant arrival sequence of dag-jobs. We first discuss priorities of the scheduling policy among jobs that are eligible for execution.

We first observe that dag-jobs of task H have both depth and deadline equal to 2; it follows that nodes of these jobs must be executed without interruption as soon as they are eligible for execution. Therefore, we let nodes of H have higher priority with respect nodes of G_F.

We now observe that G_F (the only other task) has period (equal to the deadline) that is greater than the period of dag task H. Therefore, it is sufficient to prove that, if the formula F is `true`, then each individual dag-job J of G_F can be feasibly scheduled for all possible arrival patterns of jobs of H interfering with J.

In the sequel, wlog we assume that dag-job J of task G_F is released at time 0; it follows that dag-jobs of task H that might interfere with job J might be released in the interval $[-1, D]$; we will see (Claim 5) that a dag-job of H can cause delay of other dag-jobs only when it is released either at time $10(i - 1)$ or at time $10(i - 1) + 1$ for some integer $i = 1, 2, \ldots, n$.

Since the relative deadline of G_F is $10n+1$, it follows that nodes of sub-dag C must be executed without interruption, thus directly limiting the number of processors available for the other sub-dags of G_F. We therefore let the nodes of C (together with nodes of H) have highest priority. Finally, our algorithm will give higher priorities to nodes of sub-dag B than to nodes of A.

Therefore, we will use the following priority order: first nodes of C, followed by nodes of H, B and A in this order.

Processor availability and the observation that there are no arcs from nodes of sub-dags A and B to nodes of C immediately implies the following claim.

Claim 3. *Since dag-jobs of H and sub-dags C have higher priority, they always complete execution by their deadline.*

Proof. After accounting for sub-dag C, by construction there are always at least $1+q+6m$ processors available at any time step (in fact, at least $1+q+7m$ except at the very last time step $[D-1, D)$) to schedule dag-jobs of H, sub-dag A. The maximum parallelism of H is q, hence dag-jobs of H will be processed without delays and complete execution by their deadline. $\square$

As far as the execution of sub-dag B we observe that, after nodes of H and C have been scheduled in $[0, D-1]$, there are at least $1+q+7m$ available processors. Since nodes of B have higher priority than nodes of A, there are always enough processors to execute all nodes of $B_{j,k}$, $j = 1, 2, \ldots, m$, $k = 1, 2, \ldots, 7$ in $[0, D-1]$; therefore, at time D all nodes of sub-dag B that are eligible for execution are processed.

We will see in the sequel that

- the execution of nodes $Y_i, \neg Y_i, X_i, \neg X_i$ of A_i $(i = 1, 2, \ldots, n)$ might be delayed by the release of jobs of H and by the scheduler decisions
- the above delay might delay the execution of nodes of B.

However we will prove that if F is true then exactly m chains of B are not delayed and remaining $6m$ chains are delayed by one time unit. Therefore, all chains of B complete by D and there exists a schedule that completes G_F within the deadline.

We now analyse the relationships between the schedule of A_i and the release times of dag-jobs of task H.

Claim 4. *The number of processors available at each time step and the higher priorities given to nodes of H and of sub-dags C and B with respect to priorities of nodes of A_i, i, $i = 1, 2, \ldots, n$, imply the following:*

1. *if a dag-job of task H is released at time $10(i-1)$ then node $\neg Y_i$ of A_i completes execution at time $10(i-1)+4$ and Y_i cannot complete execution either at time $10(i-1)+4$ but can complete execution at time $10(i-1)+5$;*
2. *if a dag-job of task H is released at time $10(i-1)+1$ then node Y_i of A_i completes execution at time $10(i-1)+4$ and $\neg Y_i$ cannot complete execution at time $10(i-1)+4$ but can complete execution at time $10(i-1)+5$;*

3. *otherwise (i.e. if a job of task H is not released at time $10(i-1)+1$ or $10(i-1)$), the release of the job does not delay the completion of any other node of A_i. Therefore, both $\neg Y_i$ and Y_i complete execution at time $10(i-1)+4$.*

Proof. 1) After taking into account the resources dedicated to C and B, the remaining available processors for H and A_i during time $10(i-1)+t$ are as per the following table:

t	0	1	2	3	4	5	6	7	8	9
proc.	$q+1$	$q+2$	$q+2$	$q+2$	$q+2$	$q+2$	$3q+2$	$3q+2$	$q+2$	$q+1$

See Figure 6 for reference. If a dag-job of H is released at time $10(i-1)$, since $q+q/2 > q+2$ at time $10(i-1)+1$ there are not enough processors to complete block d_i, hence Y_i cannot complete by time $10(i-1)+4$. On the other hand, by delaying only block d_i by one time unit there are enough resources to complete $\neg Y_i$ by time $10(i-1)+4$ and Y_i by time $10(i-1)+5$, since $q+2+2(q/2)+1 = 2q+3 < 2(q+2)$. (See also Figure 4).

2) The proof is similar to point 1), except that now block d_i can be processed with no delay, while block e_i has to be delayed by one time unit. (See also Figure 5).

3) The only times at which H could be released and cause interference with A_i are $10(i-1)$ and $10(i-1)+1$ (which we already covered in points 1) and 2) above), or possibly $10(i-1)+5$. For the latter case, note that of the two blocks f_i and g_i only one can be completed without delay anyway, since $2q+2q > 3q+2$. Hence, since $2q+q+1 < 3q+2$, it is possible to process without delay a dag-job of H released at time $10(i-1)+5$ without delaying any additional node of A_i. □

Claim 5. *Processor availability implies that, independently of the release of a dag-job of task H, only one node between X_i and $\neg X_i$ can complete by time $10(i-1)+8$, and both can complete at time $10(i-1)+9$. Furthermore, which node between X_i and $\neg X_i$ completes first is a decision of the scheduler.*

Proof. Only one node between X_i and $\neg X_i$ can complete without delay by time $10(i-1)+8$ because only one between f_i and g_i can complete without delay as we already argued. On the other hand, there are enough resources to complete both X_i and $\neg X_i$ by time $10(i-1)+9$ because $1+q+1+2q+2q+3 \le 2(3q+2)$. Due to the symmetry of X_i and $\neg X_i$ in block A_i, which node to complete first is a decision of the scheduler (note however that this decision may impact the completion of other parts of G_F, namely of sub-dag B). □

Claims 4 and 5 show that release of jobs of task H can delay the execution of nodes $Y_i, \neg Y_i, X_i, \neg X_i$ by at most one time unit. Since these nodes are not on a critical path, the following claim immediately follows.

Claim 6. *1. For all i, $i = 1, 2, \ldots, n$, block A_i starts execution at time $10(i-1)$ and completes execution at time $10i$.*

2. *If at most $6m$ processors are used for sub-dag B during $[D-1, D)$, then node V (and hence the entire sub-dag A) completes execution by the deadline.*

Claims 3 and 6 imply that DAG jobs of H and subdags C and A can be feasibly scheduled. It remains to show that if F is true then one can fix the remaining scheduling decisions so that also all nodes of sub-dag B and the final node V of A complete within the deadline for all possible release dates of dag-jobs of task H. We also observe that Claims 4 and 5 imply that it is sufficient to prove that G_F can be feasibly scheduled when dag-jobs of H cause the maximum interference with the job of G_F, i.e. when a dag-job of H is released either at time $10(i-1)$ or at time $10(i-1)+1$, $i = 1, 2, \dots, n$.

To fix the remaining scheduling decisions, we proceed as follows: we use release times of dag-jobs of task H to define a truth assignment S_y of universally quantified variables, use the truth of F to derive corresponding existentially quantified variables, and use the existentially quantified variables to complete the schedule (in particular, to fix the priorities between X_i and $\neg X_i$).

We define an assignment of universally quantified variables y_i, $i = 1, 2, \dots, n$, of F based on the arrival time of dag-jobs of H. Namely,

1. if a dag-job of H is released at time $10(i-1)$, we take $y_i =$ `true`;
2. if a dag-job of H is released at time $10(i-1)+1$, we take $y_i =$ `false`;
3. if no dag-job of H is released at time $10(i-1)$ or at time $10(i-1)+1$, we choose an arbitrary truth value for variable y_i .

Let S_y be the obtained truth assignment of universally quantified variables of F. Since F is true, there must exist a truth assignment S_x of existentially quantified variables $x_1, \dots, x_n$ when universally quantified variables have values as defined in S_y that satisfies all clauses. We use these values to determine the scheduler decisions on which node between X_i and $\neg X_i$ completes first. Namely,

- if the truth value of x_i is `true`, then execution of block f_i and node X_i has higher priority than execution of block g_i and node $\neg X_i$;
- if the truth value of x_i is `false`, then execution of block g_i and node $\neg X_i$ has higher priority than execution of block f_i and node X_i;

It follows that, if the truth value of x_i is `true` (`false`), then node X_i ($\neg X_i$) completes at time $10(i-1)+8$ and $\neg X_i$ (X_i) completes at time $10(i-1)+9$. Note that the decisions of the scheduler are compliant with the order of quantifiers of F, that is, x_i can be set based only on $y_1, \dots, y_i$ and $x_1, \dots, x_{i-1}$.

We complete the proof of the lemma by showing that the above defined scheduling decisions allow to complete execution of all nodes of sub-dag B and of the final node V within the deadline. Namely, we show that the scheduling of nodes of $B_{j,k}$, $j = 1, 2, \dots, m$, $k = 1, 2, \dots, 7$, only depends on the eligibility of nodes for processing that is determined by the truth values of variables of F.

Observe that a chain of B can complete at time $D-1$ if and only if its execution is never delayed. We will show that exactly m chains of B are never delayed (and therefore complete by time $D-1$), while the remaining $6m$ are delayed one time unit (and therefore complete by time D and are not so many as to interfere with V). Note that this is sufficient to ensure feasibility, since $1+q+6m$ processors are available during $[D-1, D)$, and node V and nodes from H (if there are any pending) require together $1+q$ processors.

Observe that if literal y_i ($\neg y_i$, respectively) belongs to chain $B_{k,j}$ for some k and j, then there is an edge from the node labeled Y_i ($\neg Y_i$, resp.) of A_i to the $10(i-1)+5$ node of $B_{k,j}$. Claim 4.1 asserts that if this node Y_i ($\neg Y_i$, resp.) does not complete by time-instant $10(i-1)+4$ it will complete at time $10(i-1)+5$, thus delaying execution of $B_{k,j}$ for one time unit. Analogously if literal x_i ($\neg x_i$, resp.) belongs to $B_{k,j}$ for some k and j, then there is an edge from the node labeled X_i ($\neg X_i$, resp.) of A_i to the $10(i-1)+9$ node of $B_{k,j}$. Claim 5 asserts that if node X_i ($\neg X_i$, respectively) does not complete by time-instant $10(i-1)+8$ it will complete by time-instant $10(i-1)+9$, thus delaying execution of $B_{k,j}$ for one time unit.

Consider clause C_j; since the assignments S_y and S_x satisfy F, there must exist $\hat{k}$ such that conjunct associated to chain $B_{j,\hat{k}}$ is true. Hence the above defined scheduling claims 4, 5 and 6 imply that chain $B_{j,\hat{k}}$ is never delayed in $[0, 10n]$ and, therefore, $B_{j,\hat{k}}$ is eligible for completion by time $D-1$. Since at most one conjunct per clause can be true and all clauses of F are satisfied it follows that exactly m chains complete by time $D-1$. Claims 4, 5 imply that any other chain is delayed by one time unit; therefore, $6m$ chains $B_{j,k}$ can complete by time D. This completes the proof of the lemma. □

4.2 If the sporadic task set $I(F)$ is feasible then F is true

We complete the proof by showing that if G_F is schedulable for all release sequences then F is satisfiable for all possible assignments of universally quantified variables. Namely, assume that a dag-job of task G_F is released at time 0 and consider S_y, one of the 2^n possible truth assignments of universally quantified variables.

We define an instance $I(S_y)$ of the scheduling problem by defining a sequence of n time arrivals of dag-jobs of task H that we put in one-to-one correspondence with truth values of S_y. We then use the feasible schedule of $I(S_y)$ to define a truth assignment S_x of existentially quantified variables.

We will first show that scheduling decisions for $I(S_y)$ are compliant with the order of existentially quantified variables of F. We complete the proof by showing that the existence of a feasible schedule of $I(S_y)$ implies that F is true when the truth value of variables is specified as in S_y and S_x.

Namely, given a truth assignment S_y for the y-variables of F, we define an instance $I(S_y)$ of the DAG task scheduling problem as follows: a dag-job of G_F is released at time 0. The release time of dag-jobs of H depends on the truth values of universally quantified variables; for all i, $i = 1, 2, \ldots, n$ we have

1. if $y_i = \texttt{true}$ then a dag-job of task H is released at time $10(i-1)+1$,
2. if $y_i = \texttt{false}$ then a dag-job of task H is released at time $10(i-1)$.

Finally, a dag-job of task H is released at time $D-2$, irrespectively of the truth assignment S_y. Note that the resulting arrival sequence is legal, since H's interarrival time is 8.

Lemma 7. *If every arrival sequence from $I(F)$ is schedulable, then F is* `true`.

Proof. By assumption, every arrival sequence from $I(F)$ is schedulable, so in particular the arrival sequence defined by $I(S_y)$ (with S_y defined above) admits a feasible schedule. We first define properties of this feasible schedule.

Claim 8. *In any feasible schedule for $I(S_y)$, at most one node between Y_i and $\neg Y_i$, $i = 1, 2, \ldots, n$, may complete by time $10(i-1)+4$. More precisely: if a job of dag H arrives at time $10(i-1)+1$ (resp., $10(i-1)$) then only node Y_i (resp., $\neg Y_i$) can complete by time $10(i-1)+4$.*

Proof. Assume that a dag-job of H is released at time $10(i-1)+1$; clearly this dag-job must complete by time $10(i-1)+3$. Note that during interval $[10(i-1)+2, 10(i-1)+3)$, after nodes of sub-dag C are accounted for, there are $r = 2+q+7m$ available processors. Some of these processors, namely $q+7m$, must be used to process q nodes of the released dag-job of task H (otherwise the dag-job of H misses its deadline) and $7m$ nodes of chains $B_{j,k}$ (otherwise G_F misses its deadline due to some chain of B); after taking that into account, there are two remaining processors that have to be used to schedule two nodes of A_i: c_i and the predecessor of Y_i. Note in particular that it is not possible to schedule all $q/2$ nodes of block e_i (refer to Fig. 3) during that time step. The proof in case a dag-job of task H is released at time $10(i-1)$ is similar and is omitted. □

Claim 9. *In any feasible schedule for $I(S_y)$ and for all i, $i = 1, 2, \ldots, n$, at most one node between $X_i, \neg X_i$ can complete by time $10(i-1)+8$ (this is under the decision of the scheduler).*

Proof. During interval $[10(i-1)+6, 10(i-1)+8)$, after taking into account sub-dag C, there are $r + 2q = 2 + 3q + 7m$ available processors for each time unit. Since $q = 10m$, there are not enough processors to complete execution of both nodes f_i and g_i of A_i by time $10(i-1)+7$, and hence at most one between $X_i, \neg X_i$ can complete by time $10(i-1)+8$. □

We now define a truth assignment of existentially quantified variables as follows:

1. if node X_i of A_i completes by time $10(i-1)+8$, then set $x_i = \texttt{true}$;
2. if node $\neg X_i$ of A_i completes by time $10(i-1)+8$, then set $x_i = \texttt{false}$;

3. if neither node $\neg X_i$ nor node X_i of A_i complete by time $10(i-1)+8$, then arbitrarily assign a value to variable x_i.

Note that the truth assignment is well-defined due to Claim 9. We now observe that the order in which truth assignments S_y and S_x are constructed is compliant with the order of quantifiers of F, and that all clauses of the quantified Boolean formula F are satisfied.

Claim 10. *S_y and S_x define a truth assignment to variables of F that is compliant with the order of quantifiers of F.*

Proof. Scheduling decisions concerning which node between X_i and $\neg X_i$ completes by time $10(i-1)+4$ are based only on release times of dag-jobs of task H until time $10(i-1)+1$, and on previous scheduling decisions. Note that these release times of H are in one-to-one relationship with the truth values of universally quantified variables y_j, $j = 1, 2, \dots, i$. The relevant previous scheduling decisions are in one-to-one relationship with x_j, $j = 1, 2, \dots, i-1$. The claim follows. □

The following claims show that the truth assignment S_y together with the derived truth assignment of existentially quantified variables S_x make F true.

Claim 11. *Let $j = 1, 2, \dots m$. If, in the feasible schedule for sequence $I(S_y)$, there exists $k^* = 1, \dots, 7$ such that chain B_{j,k^*} completes by time $D-1$, then clause C_j is satisfied by truth assignment (S_y, S_x).*

Proof. For chain B_{j,k^*} to complete by time $D-1$, since the length of the chain is also $D-1$, all of its nodes must be executed in lock-step, without delay. However, due to the incoming arcs from sub-dag A, this requires that nodes corresponding to any literal occurring in B_{j,k^*} should also be executed without delay. By Claims 8 and 9 and by definition of the truth assignments, this implies that the literal $\ell_{j,\cdot}$ corresponding to B_{j,k^*} is `true`. Hence, one of the seven conjuncts (namely, the k^*-th one) associated to clause C_j is true, and clause C_j is satisfied. □

Claim 12. *In the feasible schedule for sequence $I(S_y)$,*

1. *exactly m chains $B_{j,k}$ complete by time $D-1$;*

2. *for each j, $j = 1, 2, \dots m$, there exists k^* s.t. chain B_{j,k^*} completes by time $D-1$.*

Proof. For point (i), if less than m chains of B complete by time $D-1$, then at time $D-1$ there would be $2+q+6m$ pending nodes: $6m+1$ or more nodes from B still requiring processing, together with the additional node V (the sink of A, which cannot have been scheduled before $D-1$ due to its depth) and with q nodes of the dag-job of H released at time $D-2$, against only $1+q+6m$ available processors; this would contradict the feasibility of the schedule. On the other hand, the chains of B that are completed by time $D-1$ cannot be more then m, since for each clause exactly one of the 7 conjuncts is true and thus only one of the 7 chains can be completed by time $D-1$ due to the proof of Claim 11. The latter fact also implies point (ii), i.e., since exactly one conjunct for each clause C_j is true, for each j there is k^* s.t. chain B_{j,k^*} completes by time $D-1$. $\square$

By Claim 12, the hypothesis of Claim 11 is satisfied for all $j = 1, 2, \ldots, m$, hence all clauses are satisfied by truth assignment (S_y, S_x). This completes the proof of the lemma. $\square$

Theorem 1 now follows from Lemma 2, Lemma 7, and the PSPACE-hardness of QBF.

5 Open problems

We have shown that deciding the online feasibility of a set of sporadic parallel tasks on a multiprocessor platform is a PSPACE-hard problem. Beyond the obvious question of pinpointing the exact computational complexity of the online feasibility problem for DAG tasks, our results suggest a couple of questions:

1. Characterize the complexity when the number of processors is constant.
2. Show whether the feasibility problem belongs to the class PSPACE.
3. Show positive complexity results when the scheduling algorithm is fixed.

Finally, we remark that even for sequential sporadic tasksets, the exact complexity of the online feasibility problem on multiprocessor platforms is still open.

Acknowledgment We acknowledge the reviewers for their valuable suggestions and feedback. Research partly supported by Project ECS 0000024 Rome Technopole - CUP B83C22002820006, NRP Mission 4 Component 2 Investment 1.5, Funded by the European Union - NextGenerationEU. In particular, manuscript editing was funded by Rome Technopole. Alberto Marchetti-Spaccamela was supported by the ERC Advanced Grant 788893 AMDROMA “Algorithmic and Mechanism Design Research in Online Markets”.

References

[1] Sanjoy K. Baruah, Marko Bertogna, and Giorgio C. Buttazzo. *Multiprocessor Scheduling for Real-Time Systems.* Embedded Systems. Springer, 2015. doi:10.1007/978-3-319-08696-5.

[2] Sanjoy K. Baruah, Vincenzo Bonifaci, and Alberto Marchetti-Spaccamela. The global EDF scheduling of systems of conditional sporadic DAG tasks. In *27th Euromicro Conference on Real-Time Systems, ECRTS 2015, Lund, Sweden, July 8-10, 2015*, pages 222–231. IEEE Computer Society, 2015. doi:10.1109/ECRTS.2015.27.

[3] Sanjoy K. Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Leen Stougie, and Andreas Wiese. A generalized parallel task model for recurrent real-time processes. In *Proceedings of the 33rd IEEE Real-Time Systems Symposium, RTSS 2012, San Juan, PR, USA, December 4-7, 2012*, pages 63–72. IEEE Computer Society, 2012. `doi:10.1109/RTSS.2012.59`.

[4] Sanjoy K. Baruah, Rodney R. Howell, and Louis E. Rosier. Feasibility problems for recurring tasks on one processor. *Theoretical Computer Science*, 118(1):3–20, September 1993. `doi:10.1016/0304-3975(93)90360-6`.

[5] Sanjoy K. Baruah and Alberto Marchetti-Spaccamela. Feasibility analysis of conditional DAG tasks. In Björn B. Brandenburg, editor, *33rd Euromicro Conference on Real-Time Systems, ECRTS 2021, July 5-9, 2021, Virtual Conference*, volume 196 of *LIPIcs*, pages 12:1–12:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.ECRTS.2021.12`.

[6] Sanjoy K. Baruah and Alberto Marchetti-Spaccamela. The computational complexity of feasibility analysis for conditional DAG tasks. *ACM Trans. Parallel Comput.*, 10(3):14:1–14:22, 2023. `doi:10.1145/3606342`.

[7] S.K. Baruah, A.K. Mok, and L.E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings 11th Real-Time Systems Symposium*, pages 182–190. IEEE, December 1990. `doi:10.1109/REAL.1990.128746`.

[8] Vincenzo Bonifaci and Alberto Marchetti-Spaccamela. Feasibility analysis of sporadic real-time multiprocessor task systems. *Algorithmica*, 63(4):763–780, 2012. `doi:10.1007/s00453-011-9505-6`.

[9] Vincenzo Bonifaci, Andreas Wiese, Sanjoy K. Baruah, Alberto Marchetti-Spaccamela, Sebastian Stiller, and Leen Stougie. A generalized parallel task model for recurrent real-time processes. *ACM Trans. Parallel Comput.*, 6(1):3:1–3:40, 2019. `doi:10.1145/3322809`.

[10] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[11] Jian-Jia Chen and Kunal Agrawal. Capacity augmentation bounds for parallel DAG tasks under G-EDF and G-RM. Technical report, Technische Universität Dortmund, 2014.

[12] E. G. Coffman Jr. and R. L. Graham. Optimal scheduling for two-processor systems. *Acta Informatica*, 1(3):200–213, September 1972. `doi:10.1007/BF00288685`.

[13] Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35:1–35:44, 2011. `doi:10.1145/1978802.1978814`.

[14] Friedrich Eisenbrand and Thomas Rothvoss. EDF-schedulability of synchronous periodic task systems is coNP-hard. In Moses Charikar, editor, *Symp. on Discrete Algorithms, SODA, 2010*, pages 1029–1034, Philadelphia, PA, 2010.

[15] Pontus Ekberg and Wang Yi. Uniprocessor Feasibility of Sporadic Tasks with Constrained Deadlines Is Strongly coNP-Complete. In *27th Euromicro Conference on Real-Time Systems, ECRTS 2015, Lund, Sweden, July 8-10, 2015*, pages 281–286. IEEE Computer Society, 2015. `doi:10.1109/ECRTS.2015.32`.

[16] Amos Fiat and Gerhard J. Woeginger, editors. *Online Algorithms, The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*. Springer, 1998. `doi:10.1007/BFB0029561`.

[17] Nathan Fisher, Joël Goossens, and Sanjoy K. Baruah. Optimal online multiprocessor scheduling of sporadic real-time tasks is impossible. *Real Time Syst.*, 45(1-2):26–71, 2010. `doi:10.1007/s11241-010-9092-7`.

[18] M. Fujii, T. Kasami, and K. Ninomiya. Optimal Sequencing of Two Equivalent Processors. *SIAM Journal on Applied Mathematics*, July 1969. Publisher: Society for Industrial and Applied Mathematics. `doi:10.1137/0117070`.

[19] M. R. Garey and D. S. Johnson. Two-Processor Scheduling with Start-Times and Deadlines. *SIAM Journal on Computing*, 6(3):416–426, September 1977. Publisher: Society for Industrial and Applied Mathematics. `doi:10.1137/0206029`.

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

[21] Gilles Geeraerts, Joël Goossens, and Markus Lindström. Multiprocessor schedulability of arbitrary-deadline sporadic tasks: complexity and antichain algorithm. *Real Time Systems*, 49(2):171–218, 2013. `doi:10.1007/s11241-012-9172-y`.

[22] David S. Johnson. The NP-Completeness Column. *ACM Transactions on Algorithms*, 1(1):160–176, 2005. `doi:10.1145/1077464.1077476`.

[23] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of Machine Scheduling Problems. In P. L. Hammer, E. L. Johnson, B. H. Korte, and G. L. Nemhauser, editors, *Annals of Discrete Mathematics*, volume 1 of *Studies in Integer Programming*, pages 343–362. Elsevier, January 1977. `doi:10.1016/S0167-5060(08)70743-X`.

[24] Jan K. Lenstra and Alexander H. G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26(1):22–35, 1978.

[25] Jing Li, Jian-Jia Chen, Kunal Agrawal, Chenyang Lu, Christopher D. Gill, and Abusayeed Saifullah. Analysis of Federated and Global Scheduling for Parallel Real-Time Tasks. In *Proceedings of the Euromicro Conf. on Real-Time Systems*, pages 85–96. IEEE, 2014.

[26] Jing Li, Zheng Luo, David Ferry, Kunal Agrawal, Christopher D. Gill, and Chenyang Lu. Global EDF scheduling for parallel real-time tasks. *Real Time Syst.*, 51(4):395–439, 2015. `doi:10.1007/s11241-014-9213-9`.

[27] Alessandra Melani, Marko Bertogna, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Giorgio C. Buttazzo. Schedulability analysis of conditional parallel task graphs in multicore systems. *IEEE Trans. Computers*, 66(2):339–353, 2017. `doi:10.1109/TC.2016.2584064`.

[28] Aloysius K. Mok. *Fundamental design problems of distributed systems for the hard real-time environment.* PhD Thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983.

[29] James B. Orlin. The complexity of dynamic/periodic languages and optimization problems. Working paper 1679-85, Massachusetts Institute of Technology, 1985. URL: `https://dspace.mit.edu/handle/1721.1/2112`.

[30] Christos H. Papadimitriou. Games against nature. *Journal of Computer and System Sciences*, 31(2):288–301, October 1985. `doi:10.1016/0022-0000(85)90045-5`.

[31] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[32] Cynthia A. Phillips, Clifford Stein, Eric Torng, and Joel Wein. Optimal Time-Critical Scheduling via Resource Augmentation. *Algorithmica*, 32(2):163–200, 2002.

[33] Jeffrey D. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384–393, 1975.

[34] Rahul Vaze. *Online algorithms.* Cambridge University Press, 2023.

[35] Micaela Verucchi, Ignacio Sañudo Olmedo, and Marko Bertogna. A survey on real-time DAG scheduling, revisiting the Global-Partitioned Infinity War. *Real-Time Systems*, 59(3):479–530, September 2023. `doi:10.1007/s11241-023-09403-3`.