The Computational Complexity of Feasibility Analysis for Conditional DAG Tasks

SANJOY BARUAH, Washington University in Saint Louis
ALBERTO MARCHETTI-SPACCAMELA, Sapienza University of Rome

The Conditional DAG (CDAG) task model is used for modeling multiprocessor real-time systems containing conditional expressions for which outcomes are not known prior to their evaluation. Feasibility analysis for CDAG tasks upon multiprocessor platforms is shown to be complete for the complexity class PSPACE; assuming NP ≠ PSPACE, this result rules out the use of Integer Linear Programming solvers for solving this problem efficiently. It is further shown that there can be no pseudo-polynomial time algorithm that solves this problem unless P = PSPACE.

CCS Concepts: • **Computer systems organization** → **Embedded and cyber-physical systems**; • **Software and its engineering** → **Real-time schedulability**; *Scheduling;*

Additional Key Words and Phrases: Multiprocessor feasibility analysis, global scheduling, PSPACE complete

ACM Reference format:
Sanjoy Baruah and Alberto Marchetti-Spaccamela. 2023. The Computational Complexity of Feasibility Analysis for Conditional DAG Tasks. *ACM Trans. Parallel Comput.* 10, 3, Article 14 (September 2023), 22 pages.
https://doi.org/10.1145/3606342

1 INTRODUCTION

This article investigates the *feasibility analysis problem for* ***Conditional Directed Acyclic Graph (CDAG)*** *tasks*: the problem of determining whether a given real-time workload, which is specified in the CDAG model ([7]; briefly described in Section 2.2), can be scheduled to always complete by a specified deadline upon a specified number of identical processors. It follows from earlier results [16] that a simpler version of this problem is already NP-hard in the strong sense; hence, we should not expect to obtain algorithms with polynomial or pseudo-polynomial running times that solve this problem exactly. In the real-time literature, two kinds of algorithms are considered for solving such feasibility analysis problems (i.e., those that are provably NP-hard in the strong sense): (i) approximation algorithms that run in polynomial or pseudo-polynomial time; or (ii) exact algorithms that (necessarily, assuming P ≠ NP) run in exponential time. The latter approach (i.e., exact algorithms) is often based upon transforming the feasibility analysis problem in polynomial time into an **integer linear program (ILP)**, and then leveraging the tremendous recent improvements that have been obtained in the performance of ILP solvers to achieve running times that are acceptable in practice for reasonably large problem instances.

Authors' addresses: S. Baruah, Department of Computer Science & Engineering, Washington University in Saint Louis, 1 Brookings Drive, St. Louis, MO 63130-4899, USA; email: baruah@wustl.edu; A. Marchetti-Spaccamela, Department of Computer Control and Management Engineering, Sapienza University of Rome, via Ariosto 25, I-00185 Rome, Italy; email: alberto@diag.uniroma1.it.

2329-4949/2023/09-ART14 $15.00
https://doi.org/10.1145/3606342

The main technical result in this article (Theorem 1) establishes that *the CDAG feasibility analysis problem is* PSPACE *complete*—to the best of our knowledge, this is among the *first* "natural" scheduling problems shown to be PSPACE complete—see https://en.wikipedia.org/wiki/List_of_PSPACE-complete_problems and Reference [9, Appendix A8].[1] (In fact, Theorem 1 holds even when the values of the parameters characterizing the CDAG are polynomial in the size of the representation of the CDAG, there is no nesting of conditional constructs in the CDAG, and a total ordering among them can be defined.) While at first glance our PSPACE hardness result may appear to be of theoretical significance only, we emphasize that it has implications to parallel programming and real-time systems design and implementation: under the widely believed assumption that NP $\subsetneq$ PSPACE, our result implies that an approach based on transformation to ILPs is not likely to be helpful for solving the CDAG feasibility-analysis problem. (To our knowledge, this is among the first feasibility-analysis problems for which such a negative result regarding the use of ILPs has been obtained in the real-time scheduling literature.) We also show, in Section 4.4, that the PSPACE hardness result holds even for CDAGs in which all numerical parameters—execution times and the deadline—are polynomially bounded in the size of the representation of the CDAG; hence, assuming that P $\subsetneq$ PSPACE as is widely believed, it also follows that pseudo-polynomial time algorithms cannot be obtained for solving the CDAG feasibility analysis problem.

Organization. The remainder of this article is organized as follows. In Section 2, we describe the CDAG workload model and discuss what was previously known regarding CDAG feasibility analysis; we also briefly summarize (in Section 2.1) some basic facts concerning the computational complexity classes that are relevant to this article. In Section 3, we state the main result of this article—that the CDAG feasibility analysis problem is PSPACE complete—and establish membership of this problem in PSPACE. The **Quantified Boolean Formula (QBF)** problem is a canonical PSPACE-complete problem [15, 17]; in Section 4, we define a reduction from any quantified Boolean formula F to an instance G_F of the CDAG feasibility analysis problem, and in Section 5, we prove that the quantified Boolean formula F is true if and only if G_F is feasible. We close in Section 6 with a brief listing of some open questions (of particular interest: whether our PSPACE hardness result can be extended to the CDAG feasibility analysis problem when the number of available processors is a constant).

2 BACKGROUND AND MODEL

We start out in Section 2.1 with a brief summary of some basic facts concerning the computational complexity classes that are relevant to this article. Following that, in Section 2.2, we describe the CDAG workload model, formally define the CDAG feasibility analysis problem, and in Section 2.3, we summarize the current state of knowledge regarding this problem.

2.1 Some Relevant Complexity Classes

We will make reference to the following three complexity classes in this article:

- P is the set of problems that can be *solved* by algorithms with running time polynomial in the size of their inputs.
- NP is the set of problems that can be *verified* by algorithms with running time polynomial in the size of their inputs.
- PSPACE is the set of problems that can be *solved* by algorithms using an amount of *space* (memory) that is polynomial in the size of their inputs. (Since this complexity class has not

[1] A real-time scheduling problem was shown to be PSPACE-hard by Geeraerts et al. [10]; however, it is debatable whether the problem in Reference [10] can be considered to be a naturally occurring one.

previously been widely used in real-time scheduling theory, we discuss it a bit more below, and provide some intuition of its relationship to CDAG feasibility analysis.)

It is well known that the **satisfiability problem (SAT)** is a paradigmatic NP-complete problem. For the class PSPACE the paradigmatic problem is the **Quantified Boolean Formula (QBF)** Problem, which has previously [15, 17] been shown to be PSPACE complete.

Definition 1 (The QBF). INSTANCE. A Boolean formula in prenex normal form:

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

where each x_i and each y_i is a Boolean variable, and each $\ell_{j,k}$ is one of the x_i or y_i Boolean variables or its negation.
QUESTION. Does this formula evaluate to TRUE? □

(Observe that all the variables of a quantified Boolean formula in prenex normal form are quantified: It follows that the formula is either true or false.)

Two players games provide an intuitive way to understand the class PSPACE. In fact, PSPACE can be thought of as representing the existence of a winning strategy for a particular player in bounded-length perfect-information games that can be played in polynomial time. Namely, consider a two-player game where players alternate making moves for a total of n moves. Given moves $m_1, \ldots, m_n$ by the players, let $M(m_1, \ldots, m_n) = 1$ if and only if player 1 has won the game. Then player 1 has a winning strategy in the game if and only if there exists a move m_1 that player 1 can make such that for every possible response m_2 of player 2 there is a move m_3 for player 1, such that for every possible response m_4 of player 2 ...and so on, that yields $M(m_1, \ldots, m_n) = 1$. Formalizations of many popular two-player games, including checkers, generalized geography, and Sokoban, have been proven to be PSPACE-complete [11].

If we consider the QBF problem, then, given a QBF formula F, the first player has to decide values of variables $x_1, x_2, \ldots$ and the second player has to decide values of variables $y_1, y_2, \ldots$. The two players alternate their decision. Namely, the first player has to decide the value of x_1 and then the second player has to decide the value of y_1; then the first player has to decide x_i after values of x_1, x_2, x_{i-1} and of y_1, y_2, y_{i-1} are fixed but before knowing the value of y_i, that is set by the second player after the first one has decided the value of x_i. The goal of the first player is to obtain a satisfying assignment when all truth variables have been decided by both players; therefore, the QBF F evaluates to true if and only if the first player has a winning strategy.

2.2 The Conditional DAG (CDAG) Model

Task models based upon **Directed Acyclic Graphs (DAGs)** have been proposed for the purposes of exposing parallelism in real-time workloads: the *sporadic DAG model* [3] is an early example. A task in this model is specified as a 3-tuple (G, D, T), where G is a DAG, D a positive integer representing the relative deadline of the task, and T a positive integer representing the period parameter of the task. The task repeatedly releases *dag-jobs*, each of which is a collection of sequential jobs. Successive dag-jobs are released a duration of at least T time units apart. The DAG G is specified as $G = (V, E)$, where V is a set of vertices and E a set of directed edges between these vertices. Each $v \in V$ represents a job, which corresponds to the execution of a sequential piece of code and is characterized by a **worst-case execution time (WCET)**. These jobs are to be executed upon a given multiprocessor platform comprising a specified number of identical processors. We assume *global scheduling* –a job may execute upon any processor. We will consider both *preemptive* scheduling (where preempting an executing job and resuming its execution upon any processor at a

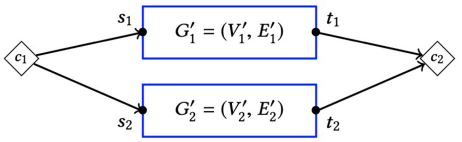


Fig. 1. A canonical conditional construct.

later point in time incurs no penalty) and *non-preemptive* scheduling (where such preemption is forbidden). The edges represent precedence constraints between pairs of jobs: if $(v_1, v_2) \in E$, then job v_1 must complete execution before job v_2 can begin execution. A release of a dag-job of the task at time-instant t means that all $|V|$ jobs $v \in V$ are released at t. If a dag-job is released at time t, then all $|V|$ jobs that were released at t must complete execution by time $t + D$.

Conditional DAG tasks. The CDAG task model was introduced [2, 14] to model the execution of conditional (e.g., if-then-else) constructs in parallel real-time code. A CDAG task, too, is specified as a 3-tuple (G, D, T), where $G = (V, E)$ is a DAG, and D and T are positive integers denoting the relative deadline and period parameters of the task. They differ from regular sporadic DAGs in that certain vertices $\in V$ are designated as *conditional vertices* that are defined in matched pairs, each such pair defining a *conditional construct*. Let (c_1, c_2) be such a pair in the DAG $G = (V, E)$—see Figure 1. Informally speaking, vertex c_1 represents a point in the code where a conditional expression is evaluated and, depending upon the outcome of this evaluation, control will subsequently flow along one of two different possible branches. It is required that these two different branches meet again at a common point in the code, represented by the vertex c_2. More formally,

(1) There are two outgoing edges from c_1 in E (say, to the vertices s_1 and s_2), and two incoming edges to c_2 (say, from the vertices t_1 and t_2), in E—see Figure 1.
(2) For each $\ell \in \{1, 2\}$, let $V'_\ell \subseteq V$ and $E'_\ell \subseteq E$ denote all the vertices and edges on paths reachable from s_ℓ that do not include vertex c_2. Vertex s_ℓ must be the sole source vertex of the DAG $G'_\ell \stackrel{\text{def}}{=} (V'_\ell, E'_\ell)$, and vertex t_ℓ must be the sole sink vertex of G'_ℓ.
(3) It must hold that $V'_1 \bigcap V'_2 = \emptyset$. Additionally for each $\ell \in \{1, 2\}$, (i) with the exception of (c_1, s_ℓ) there should be no edges in E into vertices in V'_ℓ from vertices that are not in V'_ℓ; and (ii) with the exception of (t_ℓ, c_2) there should be no edges in E from vertices in V'_ℓ to vertices that are not in V'_ℓ.

Edges (v_1, v_2) between pairs of vertices neither of which are conditional nodes represent precedence constraints exactly as in traditional sporadic DAGs, while edges involving conditional nodes represent conditional execution of code. More specifically, let (c_1, c_2) denote a defined pair of conditional vertices that together define a conditional construct. The semantics of conditional DAG execution mandate that

- After the job c_1 completes execution, exactly one of its two successor jobs becomes eligible to execute; it is not known beforehand which successor job this may be.
- Job c_2 begins to execute upon the completion of exactly one of its two predecessor jobs.

In the remainder of this article, without loss of generality, we make the *simplifying assumption that each of the conditional vertices c_1 and c_2 demarcating a conditional construct has zero execution time.*

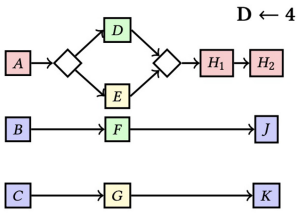
We are now ready to formally define the problem that is the focus of this paper.

Definition 2 (The CDAG Feasibility Analysis Problem). Given a CDAG G, a number $p \in \mathbb{N}$ of processors upon which G is to execute, and a relative deadline parameter D, *determine* whether

it is *feasible* to schedule G on the p processors such that it always completes execution within an interval of duration D, regardless of which conditional constructs in G evaluate to true and which evaluate to false.

(We reiterate that our results for this problem apply to both variants of this problem where preemption is permitted and where it is forbidden.)

Why this is a difficult problem. It has been widely recognized [2, 8, 14, 18] that *combinatorial explosion* is a major reason why CDAG feasibility analysis is such a difficult problem: exponentially many different combinations of outcomes are possible of the evaluation of the conditional constructs in a single task, each of which may require a very different collection of jobs to be scheduled for execution. There is, however, an additional aspect to the difficulty of this problem that has received somewhat less attention: its inherently *on line* nature. Consider the following simple illustrative example for a *restricted/ typed* CDAG [12] (i.e., where vertices are pre-assigned to individual processors):



Each vertex has WCET equal to one (except the conditional vertices – recall they have WCET zero). Processor assignments are color-coded: A, H_1, & H_2 share a processor, as do B, C, J, & K; D & F; and E & G.
If the conditional construct executes D, then C should execute during [0, 1] *—otherwise the "blue" processor will idle over* [2, 3]. *Else (i.e., the conditional construct executes E), B should execute during* [0, 1].

There are two possible outcomes of the execution of the sole conditional construct, and it may be verified that upon either outcome the set of vertices that must be executed is individually schedulable. However, which of vertices B or C, both assigned to the same processor, should execute over time-interval $[0, 1]$ necessarily differs in these two schedules and hence depends upon the outcome of the conditional construct's evaluation. But the conditional construct is only executed *after* time-instant 1, and hence this information is revealed too late. Thus, this CDAG is *in*feasible despite the sets of vertices needing to be executed upon either outcome being feasible.

CDAG feasibility as a two-player game. We can cast CDAG feasibility in the two-player game framework discussed in Section 2.1. Given a CDAG and a deadline D, the first move of player 1 (the scheduler) is to decide the set of jobs to be scheduled until the first branch is executed; then player 2 (the environment) decides the outcome of the branch. The game continues until the scheduling is completed, and the scheduler wins the game if and only if its strategy is able to complete the schedule in D time units for all outcomes of branches (i.e., all decisions of the environment).

2.3 Summarizing Prior Complexity Results

Ullman showed [16] that it is NP-complete in the strong sense to determine whether a given DAG can be scheduled to meet a specified deadline under global or partitioned scheduling upon an identical multiprocessor platform, regardless of whether preemption is permitted or forbidden. Jansen subsequently showed [12] that feasibility analysis of DAGs is NP-hard in the strong sense for restricted/ typed DAGs (where each vertex is pre-assigned to a particular processor), again under both preemptive and non-preemptive scheduling. Since these basic problems are already NP-hard in the strong sense, so are the corresponding problems for the more general CDAG model. It is also known that all these problems are also in NP for (regular) DAGs.

As for the complexity of the feasibility problem for CDAGs, it has been shown [1] that scheduling conditional DAGs is co-NP^{NP} hard; it is also known [13] that computing the worst case makespan

for a conditional DAG under list scheduling with an arbitrary but fixed ordering is co-NP complete. Finally, the problem was shown to be PSPACE complete [4] for restricted/ typed CDAGs; in this article, we generalize the techniques used in Reference [4] to show that this is also true for global scheduling.

In Reference [10] the authors studied the complexity of checking whether a sporadic task system is schedulable under a given scheduler on an identical multiprocessor platform. 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, Reference [10] 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. Note 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. Therefore, the result does not imply that the feasibility problem of scheduling a sporadic task system is PSPACE-hard.

3 THE COMPLEXITY OF CDAG FEASIBILITY ANALYSIS

The main technical result of this article is a proof of the following theorem.

THEOREM 1. *The CDAG feasibility analysis problem is* PSPACE *complete.*

In this section, we will establish that this problem is in PSPACE; PSPACE hardness is shown in the subsequent two sections. As discussed in Section 2.2 above (and illustrated on an example), the difficulty in scheduling CDAGs appears to arise from two primary sources: combinatorial explosion, and the on-line nature of the problem. While combinatorial explosion is not really an issue when designing PSPACE algorithms, dealing with the on-line nature of CDAG scheduling merits some careful handling. Consider any algorithm for testing feasibility of CDAGs. The schedule starts execution with no knowledge of the outcomes of the execution of conditional vertices; hence, at the beginning it chooses for execution a set of vertices that execute prior to knowing the outcome of the execution of any conditional vertex. This observation motivates the definition of *sets of initial vertices*—Definition 3 below. Let us assume for simplicity that each (non-conditional) vertex has worst-case execution time equal to one (later in this section, we will generalize the definition in a straightforward manner to the case where WCETs may exceed one).

*Definition 3 (Set of **Initial** Vertices).* A set S of the vertices in CDAG G that is to be scheduled upon p processors is an *initial set of vertices* if

- S contains at least one, and no more than p, conditional vertices that begin conditional constructs;
- all predecessor vertices of all vertices in S are also in S; and
- the vertices in S can be scheduled starting at time 0 such that the conditional vertices in S all execute at the last time instant in the schedule.

Algorithm 1 determines the smallest possible makespan that can be guaranteed for a CDAG by any non-clairvoyant scheduler. For each set of initial vertices S it computes the optimal schedule in which the conditional vertices are processed all together at the end. The execution of these vertices provides the information of the outcomes of all conditional executions in S. Let G' denote the subgraph of G obtained by removing the vertices in S (since they have already executed) and vertices that need not execute as a consequence of the outcomes of the conditional executions in S. Algorithm 1 proceeds recursively on G', repeatedly calling itself recursively until all conditional vertices have been processed; at this point the algorithm optimally schedules the remaining (unconditional) DAG.

ALGORITHM 1: Algorithm Compute(G)

Input: CDAG G
Output: The smallest f for which G is guaranteed to be schedulable with makespan $\leq f$ by an optimal non-clairvoyant algorithm, for all combinations of outcomes of the execution of the conditional vertices

```
1  f = ∞
2  if G contains conditional vertices then
3      for each set S of initial vertices in G do (See Definition 3)
4          (initial vertices of S are executed such that conditional vertices are executed at the end)
5          d= minimum duration of any schedule for S in which the conditional vertices are scheduled concurrently at
           the end
6          let G' be the subgraph obtained from G by removing the vertices in S
7          let C_S be the set of conditional vertices in S
8          e = 0 (e will denote the largest possible makespan of G')
9          for each truth assignment T of conditions in C_S do
10             let G'_T be the subgraph obtained from G' assuming T
11             e = max(e, Compute(G'_T))
12         f = min(f, d + e)
13 else
14     f= the minimum duration of a schedule of G upon p processors
15 return f
```

To establish that Algorithm 1 requires space that is polynomial in the size of the input CDAG G, we observe that the enumeration of the set of initial vertices of G and of outcomes of conditional executions for the considered set of initial vertices can be done using space $O(nc)$, where c denotes the number of conditional vertices. This is therefore the space requirement of the procedure excluding the space requirements of recursive calls. Note that recursive calls of Algorithm Compute(G) take as input sub-graphs G' of G that have at least one conditional vertex fewer than G does; it follows that the depth of the recursive calls is bounded by c. It follows that the space requirement of Algorithm 1 is $O(nc^2)$.

Algorithm 1 is easily extended to the case when WCETs are arbitrary positive integers and preemption is allowed. Namely, the definition of initial set of vertices (Definition 3) should be modified to include the integer WCET of each vertex. Therefore, if p_{max} denotes the maximum WCET, then the memory requirement to encode an initial set is $O(n \log p_{max})$ and the procedure can be implemented with space $O(cn \log p_{max})$. It follows that the space requirement of the procedure is $O(nc^2 \log p_{max})$.

4 A POLYNOMIAL-TIME REDUCTION

In Section 3, we showed that the CDAG feasibility analysis problem is in PSPACE; this section and the next one are devoted to proving that this problem is also PSPACE-hard. In this section, we will define a polynomial time reduction from QBF (see Definition 1) to the CDAG feasibility analysis problem: given a quantified Boolean formula F, we will describe how to construct a CDAG G_F and compute a deadline D, and provide some intuition of the motivation for this construction. Then, in Section 5, we will rigorously prove that G_F can be feasibly scheduled under both preemptive and non-preemptive scheduling for all outcomes to the conditional branches if and only if F is true, thereby completing the proof of Theorem 1.

Rather than working directly with the version of QBF defined in Definition 1, we find it convenient to reduce from a variant in which the outermost quantifier is universally quantified. (The two are easily shown to be equivalent, since either can be converted to the other by simply adding a dummy Boolean variable.) Hence, let $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_n$ denote Boolean variables;

we will define a reduction from a quantified Boolean formula,

$$F \stackrel{\text{def}}{=} \forall y_1\ \exists x_1\ \forall y_2\ \exists x_2\ \forall y_3\ \ldots \exists x_i\ \forall y_i \ldots\ \forall\, y_n\ \exists x_n\ \bigwedge_{k=1}^{m} \Big(\ell_{k,1} \vee \ell_{k,2} \vee \ell_{k,3}\Big), \tag{2}$$

where each $\ell_{k,j}$ is one of the x_i or y_i Boolean variables or its negation (i.e., $\ell_{k,j} \in \bigcup_{i=1}^{n}\{x_i, \neg x_i, y_i, \neg y_i\}$ for each $(k,j) \in [1,\ldots,m] \times [1,2,3]$), to an instance of the CDAG feasibility analysis problem.

Specifically, we will define how a CDAG G_F is constructed that is feasible with a deadline

$$D = 6n + 1 \tag{3}$$

upon p processors, where p is as defined below—see Equation (7)—if and only if F as defined in Equation (2) is TRUE.

Let α, β, and γ denote any integers satisfying the following constraints[2]:

$$\alpha = 60m, \tag{4}$$

$$\beta = 6\alpha, \tag{5}$$

$$\gamma = 6\beta. \tag{6}$$

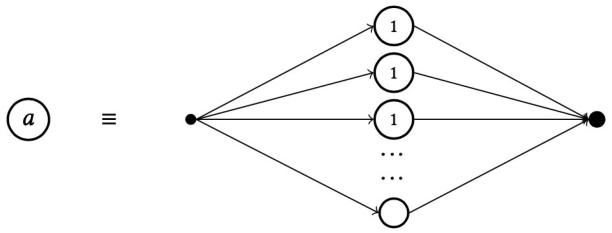
With these values defined, we choose the number of processors as follows:

$$p = (6m\gamma + 2). \tag{7}$$

In the remainder of this section, we describe in detail how G_F may be constructed from F, and provide some intuitive justification for the manner of construction; a formal proof that G_F is always schedulable to complete with makespan D upon p processors if any only if F is TRUE is provided in Section 5.

We now introduce some notation and terminology. In our diagrams of the graph we will construct, we will continue to represent conditional vertices as diamonds, and continue with our simplifying assumption that these vertices have zero execution duration.[3] We additionally introduce "join nodes" that we represent as small black filled circles; these exist solely for notational convenience, and we hence assume that they, too, have execution duration equal to zero.

A final notational convenience: for any integer $a \geq 1$, in our graph diagrams @ denotes a parallel nodes each of WCET 1, all preceded and succeeded by a single join node:



In the remainder of this article, we may refer to scheduling of such subgraphs as scheduling of a single node; namely, scheduling of @ at time t means that all a vertices of the subgraph are scheduled at time t.

[2]The choice of the constants **6, 60** in these definitions is somewhat arbitrary; informally, the intent is to ensure that (a constant $\times\ m$) $\ll \alpha \ll \beta \ll \gamma$.

[3]However, we will assume that the result of the test performed at time t cannot be used at time t by the scheduler.

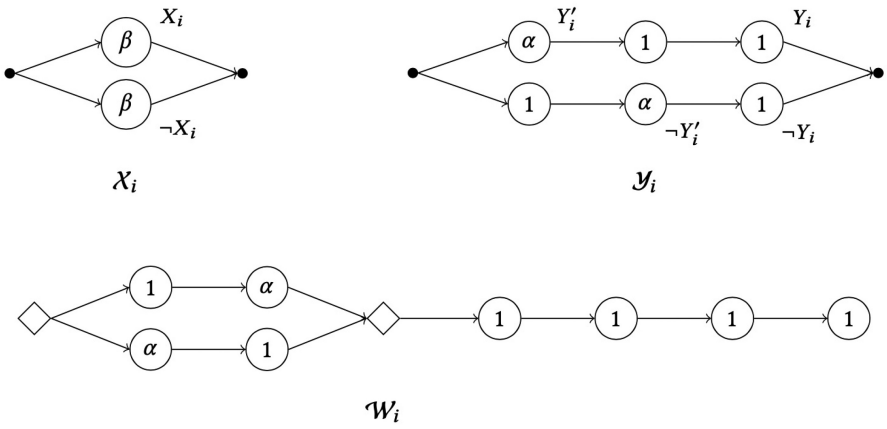


Fig. 2. Subgraphs $\mathcal{X}_i$, $\mathcal{Y}_i$, and $\mathcal{W}_i$.

4.1 Representing the Variables

For each i, $1 \leq i \leq n$, we define two subgraphs $\mathcal{X}_i$ and $\mathcal{Y}_i$. There is a pair of vertices in subgraph $\mathcal{X}_i$ ($\mathcal{Y}_i$, respectively) labeled X_i and $\neg X_i$ (Y_i and $\neg Y_i$, respectively)—see the top row of Figure 2. Informally speaking, the intended interpretation is that assigning TRUE (FALSE, respectively) to the Boolean variable x_i in the quantified Boolean formula F of Equation (2) "corresponds" to having completed the scheduling of the vertex labeled X_i ($\neg X_i$, respectively) by a particular point in time that we will specify later in this section; a similar correspondence is intended between the assignment of a truth value to y_i and the scheduling of the vertices labeled Y_i and $\neg Y_i$ in subgraph $\mathcal{Y}_i$.

Notice that the subgraphs X_i and Y_i do not contain conditional vertices; these are instead to be found in the subgraphs $\mathcal{W}_i$, also defined for each i, $1 \leq i \leq n$, that are as depicted in the bottom row in Figure 2. For these $\mathcal{W}_i$ subgraphs let us assume without loss of generality that the upper branch is taken if the conditional expression evaluates to TRUE, and the bottom branch if it evaluates to FALSE.

We now describe the manner in which the $3n$ subgraphs constructed as above are connected with each other (see Figure 3):

(1) For each i, $1 \leq i < n$,
   - There is an edge from the last vertex in $\mathcal{W}_i$ to the first vertices of $\mathcal{W}_{i+1}$, $\mathcal{X}_{i+1}$, and $\mathcal{Y}_{i+1}$.
   - There is an edge from the last vertex in $\mathcal{X}_i$ to the first vertex of $\mathcal{W}_{i+1}$.
   - There is an edge from the last vertex in $\mathcal{Y}_i$ to the first vertex of $\mathcal{W}_{i+1}$.

(2) There is an edge from the last vertex in $\mathcal{W}_n$, $\mathcal{X}_n$, and $\mathcal{Y}_n$ to a single vertex V that has WCET 1.

Intuition. We now provide some insights into our motivation for constructing the $\mathcal{X}_i$, $\mathcal{Y}_i$, and $\mathcal{W}_i$ subgraphs in the manner described above.

Since

- each $\mathcal{W}_i$ has a makespan of 6,
- there is an edge from the last vertex of $\mathcal{W}_i$ to the first vertex of $\mathcal{W}_{i+1}$ for each $i < n$,
- there is an edge from the last vertex of $\mathcal{W}_n$ to a single vertex V of WCET 1, and
- the deadline is $6n + 1$ (see Equation (3)),

it immediately follows that in any correct schedule the entire subgraph $\mathcal{W}_i$, $i = 1, 2, \ldots, n$ must execute over the interval $[6(i-1), 6i]$ for each i, $1 \leq i \leq n$; moreover at each time step all nodes

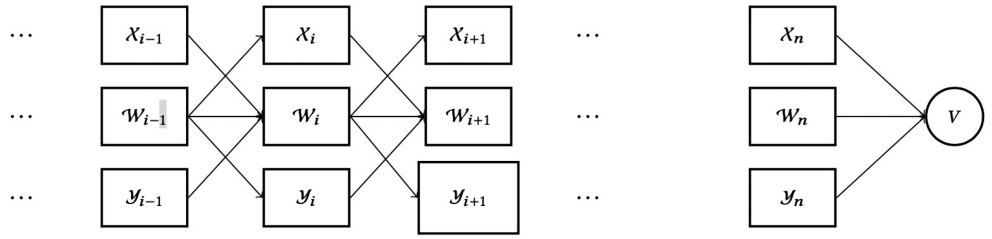


Fig. 3. Connecting the subgraphs: the connections between the subgraphs indexed by $(i-1)$, i, and $(i+1)$ are depicted, as well as edges from subgraphs $calX_n, \mathcal{W}_n, \mathcal{Y}_n$ to a single vertex V with WCET 1.

of $\mathcal{W}_i$ that are ready for execution must be scheduled to execute. Additionally, since for each i $(1 \leq i < n)$ there is an edge from the last vertex of $\mathcal{W}_i$ to the first vertex of $\mathcal{X}_{i+1}$ and the first vertex of $\mathcal{Y}_{i+1}$ and an edge from the last vertex of $\mathcal{X}_i$ and the last vertex of $\mathcal{Y}_i$ to the first vertex of $\mathcal{W}_{i+1}$, in any correct schedule the entire subgraphs $\mathcal{X}_i$ and $\mathcal{Y}_i$ must also execute over the interval $[6(i-1), 6i]$ for each i, $1 \leq i \leq n$.

We will ensure (see Section 4.2 below) that the availability of processors is such that the $\mathcal{W}_i, \mathcal{X}_i$, and $\mathcal{Y}_i$ subgraphs can be scheduled in such a manner that

(1) If the conditional expression in subgraph $\mathcal{W}_i$ evaluates to TRUE (thereby taking the upper branch), then vertex Y_i of the subgraph $\mathcal{Y}_i$ may complete by time-instant $6(i-1)+3$; both vertices $\{Y_i, \neg Y_i\}$ complete by time-instant $6i$.
(2) If, however, the conditional expression in subgraph $\mathcal{W}_i$ evaluates to FALSE and takes the lower branch, then it is the vertex $\neg Y_i$ that may complete by time-instant $6(i-1)+3$; both vertices $\{Y_i, \neg Y_i\}$ complete by time-instant $6i$.
(3) At most one of the vertices $\{X_i, \neg X_i\}$ of the subgraph $\mathcal{X}_i$ may complete by time-instant $6(i-1)+3$; both complete by time-instant $6i$.

4.2 Controlling Processor Availability

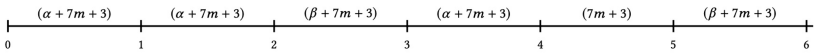
Recall (Equation (7)) that we had chosen the number of processors p in our CDAG feasibility problem instance to be $(6m\gamma + 2)$. To achieve the intended schedule discussed in Section 4.1 above over the interval $[6(i-1), 6i]$ for each i, $1 \leq i \leq n$, we must restrict the number of these processors that are available upon which to execute these sub-graphs. We do so by constructing an additional sub-graph, $\mathcal{U}$, that is a complete D-partite graph, where $D = 6n + 1$ (see Equation (3)) denotes the deadline of our CDAG feasibility problem instance. That is, the vertices of subgraph $\mathcal{U}$ may be partitioned into D disjoint subsets $U_1, U_2, \ldots, U_D$ such that there is an edge from each vertex in U_t to each vertex in U_{t+1} for each $t, 1 \leq t < D$. Each vertex has WCET 1, and the number of vertices in each U_t is chosen to ensure the processor availability depicted in Figure 4. This is achieved by choosing $|U_t|$, the number of vertices in U_t, to be equal to p minus the number of processors that we intend to have available during the time-interval $[t-1, t]$. (Hence, for example, U_D would contain a single vertex, since $(p - (6m\gamma + 1)) = (6m\gamma + 2 - 6m\gamma - 1) = \mathbf{1}$.)

The number of processors left unconsumed by $\mathcal{U}$ in any correct schedule for the instance is depicted visually in Figure 4.

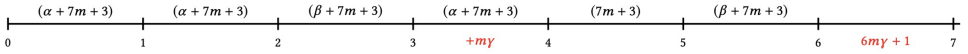
4.3 Representing the Clauses

Note that a clause $(\ell_{k,1} \vee \ell_{k,2} \vee \ell_{k,3})$ may be satisfied by having either one, two, or all three of the literals $\{\ell_{k,1}, \ell_{k,2}, \ell_{k,3}\}$ evaluate to true. Ullman [16] observed that this is equivalent to asserting

a. Processor availability over $[6 \times (i-1), 6 \times i]$, for all $i < n$:



b. Processor availability over $[6 \times (n-1), 6n+1]$:



c. Diagrammatic representation of processor availability (y axis not to scale). The gray polygon depicts processor availability over the interval $[6(i-1), 6i]$ for each i, $1 \le i < n$. The red rectangles denote additional processor availability over the interval $[6(n-1), D]$.

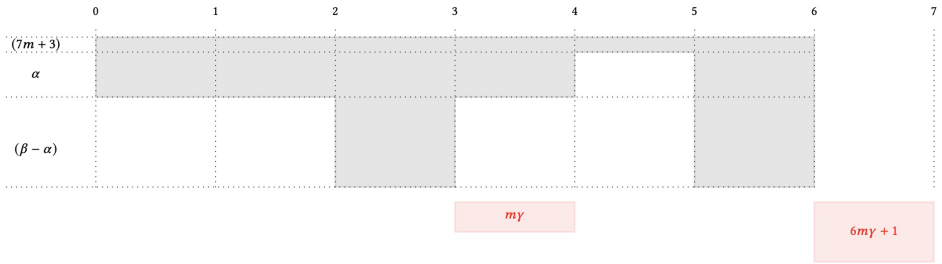


Fig. 4. Controlling the availability of processors—see Section 4.2.

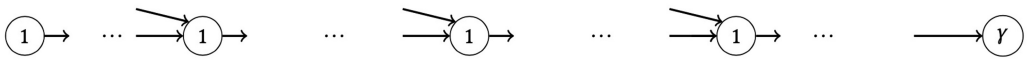


Fig. 5. The subgraph $\mathcal{Z}_{k,j}$. It comprises a chain of $D-3$ vertices, the last labeled γ and the rest, 1. Three of the vertices in the chain each have an additional incoming edge.

that in any satisfying assignment exactly one of the following seven conjuncts evaluates to true: (i) $(\ell_{k,1} \wedge \neg\ell_{k,2} \wedge \neg\ell_{k,3})$, (ii) $(\neg\ell_{k,1} \wedge \ell_{k,2} \wedge \neg\ell_{k,3})$, (iii) $(\neg\ell_{k,1} \wedge \neg\ell_{k,2} \wedge \ell_{k,3})$, (iv) $(\ell_{k,1} \wedge \ell_{k,2} \wedge \neg\ell_{k,3})$, (v) $(\ell_{k,1} \wedge \neg\ell_{k,2} \wedge \ell_{k,3})$, (vi) $(\neg\ell_{k,1} \wedge \ell_{k,2} \wedge \ell_{k,3})$, and (vii) $(\ell_{k,1} \wedge \ell_{k,2} \wedge \ell_{k,3})$. (Here, the negation operation is interpreted in the usual manner: for any Boolean variable v, we have $\neg\neg v = v$.) This observation motivates the definition for each clause C_j of seven subgraphs, $\mathcal{Z}_{j,1}, \mathcal{Z}_{j,2}, \ldots, \mathcal{Z}_{j,7}$, one to "represent" each of the seven conjuncts. These seven subgraphs can each be depicted as a chain of $(D-3)$ vertices, the first $(D-4)$ of which have WCET 1 and the last is labeled γ thereby denoting that it represents γ parallel vertices each with WCET 1 (here γ is as defined in Equation (6)), with *three additional incoming edges*—see Figure 5. For each of the seven chains, these three additional incoming edges are from the vertices corresponding to[4] the literals in the conjunct that the chain represents. Specifically,

- if the literal is x_i ($\neg x_i$, respectively) then there is an edge from the vertex X_i (the vertex $\neg X_i$, respectively) of subdag $\mathcal{X}_i$ to the $(6(i-1)+4)$th vertex in the chain
- if the literal is y_i ($\neg y_i$, respectively) then there is an edge from the vertex Y_i (the vertex $\neg Y_i$, respectively) of subdag $\mathcal{Y}_i$ to the $(6(i-1)+4)$th vertex in the chain.

We illustrate via an example.

[4]This notion of correspondence is described in Section 4.1: literal x_i ($\neg x_i$, respectively) corresponds to vertex X_i ($\neg X_i$, respectively) of subgraph $\mathcal{X}_i$, and literal y_i ($\neg y_i$, respectively) corresponds to vertex Y_i ($\neg Y_i$, respectively) of subgraph $\mathcal{Y}_i$.

Example 1. Suppose the kth clause in the QBF F is $(x_5 \vee y_3 \vee \neg y_7)$. By Ullman's observation [16], in any truth assignment for which this clause evaluates to TRUE, exactly one of the seven conjuncts *(i)* $(x_5 \wedge \neg y_3 \wedge y_7)$, *(ii)* $(\neg x_5 \wedge y_3 \wedge y_7)$, *(iii)* $(\neg x_5 \wedge \neg y_3 \wedge \neg y_7)$, *(iv)* $(x_5 \wedge y_3 \wedge y_7)$, *(v)* $(x_5 \wedge \neg y_3 \wedge \neg y_7)$, *(vi)* $(\neg x_5 \wedge y_3 \wedge \neg y_7)$, *(vii)* $(x_5 \wedge y_3 \wedge \neg y_7)$ evaluates to true. Let us consider the sixth of these conjuncts: $(\neg x_5 \wedge y_3 \wedge \neg y_7)$. In our construction of the CDAG G_F, this clause is represented by the subdag $\mathcal{Z}_{k,6}$. The three additional incoming edges into the chain $\mathcal{Z}_{k,6}$ are therefore as follows:

(1) From the vertex labeled $\neg X_5$ in the sub-DAG $\mathcal{X}_5$ into the $(6 \times 4 + 4) = \mathbf{28}$th vertex in the chain;
(2) From the vertex labeled Y_3 in the sub-DAG $\mathcal{Y}_3$ to the $(6 \times 2 + 4) = \mathbf{16}$th vertex in the chain; and
(3) From the vertex labeled $\neg Y_7$ in the sub-DAG $\mathcal{Y}_7$ to the $(6 \times 6 + 4) = \mathbf{40}$th vertex in the chain.

We point out that this chain has makespan $(D - 3)$; hence, if it is to complete execution by time-instant $(D-3)$, then it must execute without interruption. For this to happen it is necessary that the vertices in $\mathcal{X}_5$, $\mathcal{Y}_3$, and $\mathcal{Y}_7$ that are labeled $\neg X_5$, Y_3, and $\neg Y_7$, respectively, each complete execution by time-instants 27, 15, and 39, respectively. □

Intuition. We now provide some insights into our motivation for constructing the $\mathcal{Z}_{k,j}$ subgraphs in the manner described above. In Section 5, we will formally show that the availability of processors (discussed in Section 4.2 above) is such that in any correct schedule, at least one of the seven subdags $\mathcal{Z}_{j,1}, \mathcal{Z}_{j,2}, \ldots, \mathcal{Z}_{j,7}$ must complete execution by time-instant $(D - 3)$. Since the makespan of each is $D - 3$, this immediately implies that at least one of these seven subgraphs must execute without interruption over the interval $[0, (D - 3)]$. We will show, in Section 5, a correspondence between this happening and a truth assignment to the Boolean variables that causes the kth clause to be satisfied—i.e., evaluate to TRUE.

4.4 Putting the Pieces Together

The graph G_F constructed from QBF F in the manner described above comprises two (weakly) connected components: the $3n$ $\mathcal{W}_i, \mathcal{X}_i$, and $\mathcal{Y}_i$ subgraphs $(1 \leq i \leq n)$, vertex V, and the $7m$ subgraphs $\mathcal{Z}_{k,1}, \mathcal{Z}_{k,2}, \ldots, \mathcal{Z}_{k,7}$ $(1 \leq k \leq m)$ form one component; the subgraph $\mathcal{U}$ forms the other. Since the values of α, β, and γ (as defined in Equations (4)–(6)) are all polynomial in m, it follows that both the size of this DAG G_F that we construct, and the value of the deadline parameter D, are both polynomial in the size of the QBF F.

We also point out that there are n conditional expressions—one per $\mathcal{W}_i$—in G_F, that there is no nesting of conditional expressions, and that there is a total ordering among them.

Based on these observations above, it directly follows from Theorem 1 (which we will formally prove in the next section) that

COROLLARY 1. *The CDAG feasibility analysis problem is* PSPACE *hard even when*

(1) the values of the parameters (WCETs, D) characterizing the CDAG are polynomial in the size of the representation of the CDAG; and
(2) there is no nesting of conditional constructs in the CDAG, and a total ordering among them can be defined (such that the exact order in which they will execute is a priori known).

That is, the CDAG feasibility analysis problem cannot be solved by a pseudo-polynomial time algorithm (assuming P ≠ PSPACE), and remains computationally highly intractable even when the structural relationship among the conditional constructs is very simple—a linear chain.

5 PROOF OF PSPACE HARDNESS

In this section, we rigorously establish the correctness of Theorem 1 by showing that the CDAG G_F, constructed from the given quantified Boolean formula F of Equation (2) as detailed in the previous section, is always schedulable to complete with makespan D (Equation (3)) upon p processors (Equation (7)) if and only if F is TRUE, regardless of whether preemption is permitted or forbidden. The presentation is divided in two parts—Lemma 1 (Section 5.1) establishes that if F is TRUE then G_F is always scheduled correctly, while Lemma 2 (Section 5.2) shows that the schedulability of G_F implies that F is TRUE.

5.1 If F is True, then G_F is Schedulable

The CDAG G_F that we constructed in Section 4 above has n conditional constructs, where n is the number of x_i variables (and also the number of y_i variables) in the quantified Boolean formula F defined in Equation (2). On any particular complete execution of the task, the outcome of the evaluation of each of the n conditionals will be either TRUE or FALSE. We will now establish that if the quantified Boolean formula F as defined in Equation (2) is TRUE then G_F can be scheduled upon p processors to complete within its specified deadline D for all 2^n possible combinations of these outcomes.

LEMMA 1. *If the quantified Boolean formula F of Equation (2) is* TRUE*, then the CDAG G_F constructed as described in Section 4 can be scheduled to always complete within a deadline $D = (6n+1)$ upon $p = (6m\gamma + 1)$ processors, where γ is as defined in Equation (6).*

Assume F is TRUE; we define below a run-time scheduling algorithm for G_F upon p processors that completes within the deadline D for any combination of outcomes of its n conditional expressions. Although at any time this algorithm only makes scheduling decisions regarding the jobs corresponding to all the vertices in G_F that are currently eligible to execute, we will, for ease of presentation, first describe (and prove properties of) the manner in which it schedules the subgraph $\mathcal{U}$; next, how it schedules the $\mathcal{X}_i, \mathcal{Y}_i$, and $\mathcal{W}_i$ subgraphs; and finally, how it schedules the $\mathcal{Z}_{k,j}$ subgraphs.

The subgraph $\mathcal{U}$. Our run-time scheduling algorithm assigns greatest priority to the vertices of the subgraph $\mathcal{U}$, thereby ensuring that the number of processors available for executing the remainder of G_F is as specified in Section 4.2. The following claim is easily seen to be correct by construction: our choice, as articulated in Section 4.2, of the number of vertices to have in each partition U_i of the vertices of $\mathcal{U}$, ensures this.

CLAIM 1.1. *All vertices in subgraph $\mathcal{U}$ complete execution by the deadline D.*

The $\mathcal{X}_i, \mathcal{Y}_i$, and $\mathcal{W}_i$ subgraphs. We now address the manner in which our run-time algorithm schedules the vertices in the subgraphs $\mathcal{X}_i, \mathcal{Y}_i$, and $\mathcal{W}_i$ for each $i, 1 \leq i \leq n$. Observe that the conditional expression (the diamond-shaped vertex) in $\mathcal{W}_1$ is eligible to execute at time-instant 0; our run-time scheduler will ensure (see Claim 1.2 below) that for each $i > 1$ the conditional expression in $\mathcal{W}_i$ is eligible to execute at time-instant $6(i-1)$. It does so in the following manner. Suppose the conditional expression in $\mathcal{W}_i$ is executed at time-instant t. We will see below that the remaining processor availability (after scheduling vertices of subdag $\mathcal{U}$) that is depicted in Figure 4 enables our run-time algorithm to construct the following schedules over $[t, t+6]$.

- Suppose that the conditional expression evaluates to TRUE; intuitively (as discussed in Section 4.1), we associate this with the Boolean variable y_i being assigned value TRUE. Since F is assumed to be TRUE, it must be the case that x_i can subsequently be assigned some truth value such that F evaluates to TRUE when y_i is TRUE and x_i gets this value.

Recall from Section 4.1 that we have assumed that the upper branch of the conditional construct needs to execute when the conditional expression evaluates to TRUE. In so doing, our run-time algorithm executes each of the subgraphs $\mathcal{W}_i$, $\mathcal{Y}_i$, and $\mathcal{X}_i$ upon the following number of processors over the interval $[t, t+6)$:

	Time Intervals					
Subgraph	$[t, t+1)$	$[t+1, t+2)$	$[t+2, t+3)$	$[t+3, t+4)$	$[t+4, t+5)$	$[t+5, t+6)$
$\mathcal{W}_i$	1	α	1	1	1	1
$\mathcal{Y}_i$	α	1	2	α	1	0
$\mathcal{X}_i$	0	0	β	0	0	β

in a manner that completes execution of (i) vertex Y_i at time-instant $(t+3)$; (ii) vertex $\neg Y_i$ at time-instant $(t+5)$; (iii) X_i ($\neg X_i$, respectively) by time-instant $(t+3)$ if the value TRUE (FALSE, respectively) is assigned to x_i to have F evaluate to TRUE when y_i is TRUE; and (iv) the remaining vertex from among $\{X_i, \neg X_i\}$ by time-instant $t+6$. (We will prove in Claim 1.2 that this is possible.)

- Suppose that the conditional expression evaluates to FALSE; we informally associate this with the Boolean variable y_i being assigned value FALSE. Since F is assumed to be TRUE, x_i can subsequently be assigned some value in {TRUE, FALSE} such that F evaluates to TRUE when y_i is FALSE and x_i gets this value.
  Recall that the lower branch of the conditional construct needs to execute when the conditional expression evaluates to FALSE. In this case, our run-time algorithm executes the subgraphs $\mathcal{W}_i$, $\mathcal{Y}_i$, and $\mathcal{X}_i$ upon the following number of processors each:

	Time Intervals					
Subgraph	$[t, t+1)$	$[t+1, t+2)$	$[t+2, t+3)$	$[t+3, t+4)$	$[t+4, t+5)$	$[t+5, t+6)$
$\mathcal{W}_i$	α	1	1	1	1	1
$\mathcal{Y}_i$	1	α	1	α	1	1
$\mathcal{X}_i$	0	0	β	0	0	β

  in a manner that completes execution of (i) vertex $\neg Y_i$ at time-instant $(t+3)$; (ii) vertex Y_i at time-instant $(t+5)$; (iii) X_i ($\neg X_i$, respectively) by time-instant $(t+3)$ if the value TRUE (FALSE, respectively) is assigned to x_i to have F evaluate to TRUE when y_i is FALSE; and (iv) the remaining vertex from among $\{X_i, \neg X_i\}$ by time-instant $t+6$. (We will again prove in Claim 1.2 that this is possible.)

Note that the above scheduling decisions are only based on the truth values of variables x_i and y_i and, therefore, they are compliant with the definition of DAG G_F. The following establishes that the schedule generated by the run-time scheduling algorithm described above does indeed map the truth values of the Boolean variables to the scheduling of the X_i, $\neg X_i$, Y_i, and $\neg Y_i$ vertices.

Claim 1.2. *For all i, $i = 1, 2, \ldots, n$, the following hold*

(1) The graphs $\mathcal{W}_i, \mathcal{X}_i, \mathcal{Y}_i$, execute entirely within the interval $[6(i-1), 6i)$.
(2) If the truth value of variable y_i is TRUE *(*FALSE*, respectively), then the vertex of $\mathcal{Y}_i$ that is labeled Y_i ($\neg Y_i$, respectively) will complete by time $6(i-1)+3$; otherwise it will complete by time $6i$.*
(3) If the truth value of variable x_i is TRUE *(*FALSE*, respectively), then all β jobs of $\mathcal{X}_i$ corresponding to the vertex labeled X_i ($\neg X_i$, respectively) will complete by time $6(i-1)+3$; otherwise, they will complete by time $6i$.*

Proof. To prove this claim, we have to show that the processor availability depicted in Figure 4 allows the run-time algorithm to construct the schedules discussed above for both the cases when

the conditional expression in the ith conditional construct evaluates to TRUE and when it evaluates to FALSE. This is proved by induction on i.

Base case ($i = 0$). The conditional expression in $\mathcal{W}_1$ has no predecessors and so executes at time-instant $6(i-1) = 6 \times 0 = \mathbf{0}$. From Figure 4, it is evident that the processor availability over the interval $[0, 6]$:

Time Intervals					
$[0, 1)$	$[1, 2)$	$[2, 3)$	$[3, 4)$	$[4, 5)$	$[5, 6)$
$(\alpha + 7m + 3)$	$(\alpha + 7m + 3)$	$(\beta + 7m + 3)$	$(\alpha + 7m + 3)$	$(7m + 3)$	$(\beta + 7m + 3)$

is (more than) adequate to both provide the required number of processors when the conditional expression evaluates to TRUE (FALSE, respectively), and to ensure that the vertices labeled X_1 and $\neg X_1$ complete as stated in the claim. To see that the vertices labeled Y_1 and $\neg Y_1$ also complete as stated in the claim, observe that we associate executing the upper (lower, respectively) branch of $\mathcal{W}_1$ with the variable y_1 having truth value TRUE (FALSE, respectively); such execution of the upper (lower, respectively) branch of $\mathcal{W}_1$ consumes the α processors available during the time-interval $[0, 1]$ ($[1, 2]$, respectively), but leaves the α processors available during the time-interval $[1, 2]$ ($[0, 1]$, respectively) for the predecessor vertex of vertex Y_1 ($\neg Y_1$, respectively) to execute and thereby enables vertex Y_1 ($\neg Y_1$, respectively) to complete by time-instant 3. Furthermore, it is evident that all four vertices X_1, $\neg X_1$, Y_1, and $\neg Y_1$ complete by time-instant 6.

Induction Step. Given, as an induction hypothesis, that $\mathcal{X}_{i-1}$, $\mathcal{Y}_{i-1}$, and $\mathcal{W}_{i-1}$ all complete by time-instant $6(i-1)$, an argument virtually identical to the one in the base case above may be used: the processor availability over $[6(i-1), 6i]$ is given by

Time Intervals					
$[6(i-1), 6(i-1)+1)$	$[6(i-1)+1, 6(i-1)+2)$	$[6(i-1)+2, 6(i-1)+3)$	$[6(i-1)+3, 6(i-1)+4)$	$[6(i-1)+4, 6(i-1)+5)$	$[6(i-1)+5, 6i)$
$\alpha + 7m + 3$	$\alpha + 7m + 3$	$\beta + 7m + 3$	$\alpha + 7m + 3$	$7m + 3$	$\beta + 7m + 3$

which is identical to the availability in the base case, and hence analogous conclusions can be drawn.

And finally, note from Figure 4 that the processor availability for the case $i = n$ is in fact even greater—there are additional $m\gamma$ processors available in the interval $[6(n-1)+3, 6(n-1)+4]$—and hence the run-time algorithm is once again able to schedule $\mathcal{X}_n$, $\mathcal{Y}_n$, and $\mathcal{W}_n$ to possess the properties expressed in the claim above. □

The following claim immediately follows by observing that all vertices of $\mathcal{X}_n$, $\mathcal{Y}_n$, and $\mathcal{W}_n$ complete by time $6n$ and and considering the processor availability in $[D, D-1]$.

Claim 1.3. *Vertex V completes execution by time D.*

The $Z_{k,j}$ subgraphs. We have seen above how our run-time algorithm schedules the subgraph $\mathcal{U}$ as well as the $\mathcal{X}_i$, $\mathcal{Y}_i$, and $\mathcal{W}_i$ subgraphs $1 \leq i \leq n$. It remains to discuss how the $Z_{k,j}$ subgraphs are to be scheduled. The run-time algorithm executes these upon the processors that are left over after the scheduling of the subgraph $\mathcal{U}$ and the $\mathcal{X}_i$, $\mathcal{Y}_i$, and $\mathcal{W}_i$ subgraphs as previously described above, by prioritizing the scheduling of the first $(D-4)$ vertices of any of these $Z_{k,j}$ subgraphs over the scheduling of the last vertex of any $Z_{k,j}$ subgraph. We now establish that all these subgraphs are successfully scheduled in this manner.

Claim 1.4. *Exactly m subgraphs $\mathcal{Z}_{k,j}$, $1 \leq k \leq m, 1 \leq j \leq 7$, complete execution by $6(n-1)+4$.*

Proof. It is sufficient to show that, for each k, $1 \leq k \leq m$, the last γ parallel jobs of exactly one subgraph $\mathcal{Z}_{j,k}$ are ready for execution at time $6(n-1)+3$. Therefore, the $m\gamma$ processors that are

available over $[6(n-1)+3, 6(n-1)+4)$—see Figure 4—can be used to complete the execution of these jobs by time $6(n-1)+4$.

Recall from Section 4.3 that the seven subgraphs $\mathcal{Z}_{k,j}$, $j = 1, 2, \ldots, 7$, "represent" the seven possible satisfying assignments of the kth clause and that at most one of these assignments can be true. We now observe that if literal x_i ($\neg x_i$, respectively) belongs to subgraph $\mathcal{Z}_{k,j}$ for some k and j, then there is an edge from the vertex labeled X_i ($\neg X_i$, respectively) of $\mathcal{X}_i$ to the $6(i-1)+4$th vertex of the chain $\mathcal{Z}_{k,j}$. Claim 1.2.(3) asserts that if this vertex X_i ($\neg X_i$, respectively) does not complete by time-instant $6(i-1)+3$ it will complete at time $6i$, thus delaying execution of $\mathcal{Z}_{k,j}$ for up to three time units. Analogously, if literal y_i ($\neg y_i$, respectively) belongs to subgraph $\mathcal{Z}_{k,j}$ for some k and j, then there is an edge from the vertex labeled Y_i ($\neg Y_i$, respectively) of $\mathcal{Y}_i$ to the $6(i-1)+4$th vertex of the chain $\mathcal{Z}_{k,j}$. Claim 1.2.(2) asserts that if vertex Y_i ($\neg Y_i$, respectively) does not complete by time-instant $6(i-1)+3$ it will complete by time-instant $6i$, thus delaying execution of $\mathcal{Z}_{k,j}$ for at at most three time units.

We have shown that, for each k, at most one assignment corresponding to the seven subgraphs $\mathcal{Z}_{k,j}$, $j = 1, 2, \ldots, 7$, can be true; since the considered assignment satisfies the formula it follows that for each k, $1 \leq k \leq m$, the last γ parallel jobs of exactly 1 subgraph $\mathcal{Z}_{k,j}$, $j = 1, 2, \ldots, 7$, is ready for execution at time $6(n-1)+3$. □

Proof of Lemma 1. We now complete the proof of Lemma 1. We first observe that the above defined scheduling decisions are compliant with the order and values of the truth assignment of F; namely, scheduling of

(1) $\mathcal{U}$ and V is independent on the truth assignment of F;
(2) $\mathcal{W}_i$ and $\mathcal{Y}_i$ only depends on the truth value of variable y_i;
(3) $\mathcal{X}_i$ only depends on the truth value of variable x_i;
(4) $\mathcal{Z}_{j,k}$ only depends on the eligibility of nodes of the subgraph.

We have shown

(1) subgraphs $\mathcal{U}$ and vertex V complete by time D (Claims 1.1 and 1.3);
(2) subgraphs $\mathcal{W}_n, \mathcal{X}_n, \mathcal{Y}_n$ complete by time $D-1$ (Claim 1.2);
(3) exactly m subgraphs $\mathcal{Z}_{j,k}$ complete by time $D-3$ (Claim 1.4).

Note that subgraphs $\mathcal{Z}_{j,k}$ that do not complete by time $D-3$ are delayed by at most three time units; therefore, the last γ parallel jobs of all these $6m$ remaining subgraphs $\mathcal{Z}_{j,k}$ are ready for execution at time-instant $D-1$. By considering vertices U_D, V and processor availability, we can see that there are $6m\gamma$ available processors in $[D-1, D]$; these processors are sufficient to complete execution of all remaining subgraphs $\mathcal{Z}_{j,k}$. Therefore, the schedule completes by the deadline D thus completing the proof of Lemma 1. □

5.2 If G_F is Schedulable, then F is True

The CDAG G_F constructed in Section 4 has n conditional constructs, where n is the number of x_i variables (and also the number of y_i variables) in the quantified Boolean formula F defined in Equation (2). On any particular complete execution of the task, the outcome of the evaluation of each of the n conditionals will be either TRUE or FALSE. We will now establish that if G_F can be scheduled upon p processors to complete within its specified deadline D for all 2^n possible combinations of these outcomes, then the quantified Boolean formula F as defined in Equation (2) is TRUE. Consider any particular combination of outcomes of the n conditional constructs. Based upon the construction of G_F from F as described in Section 4, the correspondence of this combination of outcomes to an assignment of truth values to the universally quantified variables (the y_i's)

is quite straightforward: Variable y_i is assigned value TRUE (FALSE, respectively) if the conditional construct in subgraph $\mathcal{Y}_i$ evaluates to TRUE (FALSE, respectively).

We start out establishing some properties that any such correct schedule must possess.

CLAIM 1.5. *In any correct schedule of G_F*

(1) The sub-graph $\mathcal{U}$ executes such that all vertices in U_i (i.e., those in the ith-level partition of $\mathcal{U}$'s vertex set) execute over the time-interval $[t-1, t)$.
(2) For each $i, 1 \leq i \leq n$, sub-graph $\mathcal{W}_i$ begins execution at time-instant $6(i-1)$ and completes at time-instant $6i$.
(3) For each $i, 1 \leq i \leq n$, sub-graphs $\mathcal{X}_i$ and $\mathcal{Y}_i$ begin execution at or after time-instant $6(i-1)$ and complete execution by time-instant $6i$.

PROOF. Immediately follows from the manner in which the subgraphs were constructed and connected to each other (see Figures 2 and 3). □

The proof of Lemma 1 above may additionally suggest that the existentially quantified variables (the x_i's) be assigned truth values based on the completion times of the vertices labeled X_i and $\neg X_i$ in subgraph $\mathcal{X}_i$: x_i should be assigned the value TRUE (FALSE, respectively) if all β WCET-1 jobs represented by the vertex labeled X_i ($\neg X_i$, respectively) in subgraph $\mathcal{X}_i$ complete execution by time-instant $6(i-1)+3$.

However this idea does not quite work: it is possible for neither vertex X_i nor $\neg X_i$ to have completed execution by time-instant $6(i-1)+3$ in a correct schedule for G_F. Consider, for instance, the following quantified Boolean formula:

$$\forall y_1\ \exists x_1\ \forall y_2\ \exists x_2\ (\neg y_1 \vee x_1 \vee x_2) \wedge (y_1 \vee x_1 \vee y_2).$$

Observe that if x_1 is assigned the value TRUE, then this QBF evaluates to TRUE regardless of the value assigned to x_2 (or indeed, the values of the y_i variables); this in turn would imply that if we were to construct a CDAG for the above quantified Boolean formula as described in Section 4, then correct schedules exist in which neither vertex in $\{X_2, \neg X_2\}$ completes by time-instant $6 \times (2-1) + 3 = \mathbf{9}$. Similarly the same claim holds for vertices $\{Y_2, \neg Y_2\}$ associated to variable y_2.

This subtlety makes proving the following lemma somewhat less straightforward than may appear at first sight. We circumvent it in the following manner: If neither X_i nor $\neg X_i$ (Y_i nor $\neg Y_i$) complete by time $6(i-1)+3$, then we define (Definition 4) a suitable truth value to variable x_i (y_i), and we show that this truth assignment makes F true.

LEMMA 2. *If the CDAG G_F constructed as described in Section 4 can always (i.e., for all 2^n combinations of outcomes of the evaluations of its n conditional constructs) be scheduled to complete within a deadline $D = 6n + 1$ upon $p = (6m\gamma + 2)$ processors, where γ is as defined in Equation (6), then the quantified Boolean formula F of Equation (2) is* TRUE.

Given a correct schedule for any combination of outcomes for the conditional constructs in G_F, the following definition determines a truth assignment $A_S(F)$ to the Boolean variables

Definition 4 (Truth Assignment $A_S(F)$). Given a correct schedule S of the CDAG G_F, we define a truth assignment $A_S(F)$ to the x_i and y_i Boolean variables of the quantified Boolean formula F of Equation (2) in the following mannner:

(1) x_i is assigned truth value TRUE (FALSE) if the vertex X_i ($\neg X_i$) completes by time $6(i-1)+3$;
(2) Any unassigned x_i is assigned the value TRUE or FALSE arbitrarily;
(3) y_i is assigned truth value TRUE (FALSE) according to the outcome of the ith branch condition of S.

To prove Lemma 2, we will show that Definition 4 determines a truth assignment $A_S(F)$ to the Boolean variables such that

(1) each Boolean variable is only assigned one value and, hence, $A_S(F)$ is indeed a well-defined truth assignment—Claim 2.1.
(2) the truth assignment to variable y_i, the ith universally quantified Boolean variable, is compliant with the outcome of the ith conditional construct; i.e., if the ith conditional construct is true (false), then y_i is true (false)—Claim 2.1.
(3) the truth values of x_i and y_i, the ith existentially and universally quantified Boolean variables, are compliant with the kind and the order of the quantifiers in F—Claim 2.2.
(4) all clauses of the quantified Boolean formula F are satisfied—Claims 2.3 and 2.4.

The following claim proves that $A_S(F)$ is a well-defined truth assignment; its proof builds upon the structural properties of correct schedules that were identified in Claim 1.5 above.

Claim 2.1. *In any correct schedule of G_F it is the case for each i, $1 \leq i \leq n$, that*

(1) At most one of the vertices $\{Y_i, \neg Y_i\}$ completes execution by time $6(i-1)+3$, and which one this may be is determined by the outcome of evaluating the conditional expression in subgraph $\mathcal{W}_i$. Specifically, only Y_i ($\neg Y_i$, respectively) may complete by time $6(i-1)+3$ if the conditional expression in $\mathcal{W}_i$ evaluates to TRUE (FALSE, *respectively). The other vertex in $\{Y_i, \neg Y_i\}$ completes by time $6i$ but not before time $6(i-1)+5$.*
(2) At most one of the vertices $\{X_i, \neg X_i\}$ completes execution by time $6(i-1)+3$, and both complete execution by time $6i$.

Proof. We first observe that Claim 1.5 implies that in any correct schedule for G_F (i) the initial vertices of $\mathcal{X}_i$ and $\mathcal{Y}_i$ cannot begin execution before time $6(i-1)$, and (ii) subgraphs $\mathcal{X}_i$ and $\mathcal{Y}_i$ must complete execution by time $6i$. It follows that nodes $X_i, \neg X_i, Y_i, \neg Y_i$ all complete by time $6i$.

We next observe (see Figure 4) that the processor availability left by subgraph $\mathcal{U}$ in the interval $[6(i-1), 6(i-1)+2]$ is $2 \times (\alpha + 7m + 3)$, and that $\mathcal{W}_i$ consumes $(\alpha+1)$ units of this availability regardless of whether it traverses its upper or its lower branch. This leaves $(\alpha + 14m + 5)$ units of processor capacity over this interval; since $\alpha > 14m+5$ (by Equation (4), $\alpha > 60m$), this implies that at most one of the two vertices labeled $\{Y_i', \neg Y_i'\}$ may have completed execution by time-instant $6(i-1)+2$.

Since $\mathcal{W}_i$ must be executed without any delay it follows that α processing units for processing nodes $\{Y_i', \neg Y_i'\}$ of $\mathcal{Y}_i$ are available either in $[6(i-1), 6(i-1)+1]$ (if the upper branch of the conditional construct is taken) or in $[6(i-1)+1, 6(i-1)+2]$ (if the lower branch of the conditional construct is taken). Note also that there are other (α) units in the interval $[6(i-1)+3, 6(i-1)+4]$ that can be used to complete execution of both $\{Y_i'$ and $\neg Y_i'\}$.

Namely, if sub-graph $\mathcal{W}_i$ executes

(1) the upper branch of its conditional construct then the available processor capacity of α units in $[6(i-1), 6(i-1)+1]$ can only be used to complete the execution of vertex Y_i', thus allowing to complete Y_i by time $6(i-1)+3$ and $\neg Y_i'$ by time $6(i-1)+4$;
(2) the lower branch of its conditional construct then the available capacity in $[6(i-1)+1, 6(i-1)+2]$ can be used to complete execution of either vertex Y_i' or vertex $\neg Y_i$ $\neg Y_i'$; however, the completion of Y_i' by time $6(i-1)+2$ does not allow to complete the execution of Y_i by time $6(i-1)+3$ and Y_i' by time $6(i-1)+4$.

It follows that the choice of which of the two vertices labeled $\{Y_i, \neg Y_i\}$ may have completed execution by time-instant $6(i-1)+3$ is dictated entirely by the outcome of the conditional construct. Consequently at most one of the two vertices labeled $\{Y_i, \neg Y_i\}$ may have completed execution by

time-instant $6(i-1)+3$, and which one this may be is determined by whether subgraph $\mathcal{W}_i$ executed the upper branch or the lower branch of its conditional construct, as claimed.

Note also that if vertex Y_i ($\neg Y_i$) is not completed by time $6(i-1)+3$ then vertex Y'_i ($\neg Y'_i$) is completed by time $6(i-1)+4$; this allows to complete Y_i ($\neg Y_i$) by time $6(i-1)+6$ and to complete the proof of Claim 2.1.(1).

Finally, we observe that the processor availability left by subgraph $\mathcal{U}$ in the interval $[6(i-1), 6(i-1)+3]$ is $2\times(\alpha+7m+3)+(\beta+7m+3)=\beta+2\alpha+21m+9$. But since $\beta > 2\alpha+21m+9$ (by Equation (5), $\beta > 60\alpha$ while by Equation (4), $\alpha > 60m$), at most one of the vertices of $\mathcal{X}_i$ labeled $\{X_i, \neg X_i\}$ may complete execution by time-instant $6(i-1)+3$, thus completing the proof of the Claim. □

The next claim establishes that the truth assignment $A_S(F)$ is consistent with the identity and ordering of quantifiers on the Boolean variables in quantified Boolean formula F of Equation (2).

Claim 2.2. *Given a quantified Boolean formula F and a correct schedule S for the CDAG G_F constructed from F as described in Section 4, the truth assignment $A_S(F)$ of Definition 4 to the x_i and y_i Boolean variables in F is done in a manner that is compliant with the kind and the order of the quantifiers in F.*

Proof. Observe first that Claim 2.1 establishes that the decision on which of the vertices $\{Y_i, \neg Y_i\}$ may complete by time $6(i-1)+3$ depends on the outcome of the conditional construct in $\mathcal{W}_i$ and it is not under the control of the scheduler, thus reflecting the universal quantifiers on the y_i variables.

Claim 2.1 also reflects the existential quantifiers on the x_i variables in that the scheduler chooses whether to complete X_i or $\neg X_i$ by time $6(i-1)+3$ before the environment (i.e., run-time conditions) determines which of Y_{i+1} and $\neg Y_{i+1}$ will complete $6i+3$.

Finally, observe that the order of quantifiers is maintained: the scheduler must decide to execute one of $\{X_i, \neg X_i\}$ before one of $\{Y_{i+1}, \neg Y_{i+1}\}$ is scheduled; furthermore, the scheduler has to decide to execute one of $\{X_i, \neg X_i\}$ after one of $\{Y_i, \neg Y_i\}$ is chosen for execution by the environment. □

The next two claims establish properties on the completion time of subgraphs $\mathcal{Z}_{k,j}$.

Claim 2.3. *If subgraph $\mathcal{Z}_{k,j}$ completes by time-instant $(D-3)$ in some correct schedule S of F, then the truth assignment $A_S(F)$ constructed from S as described in Definition 4 satisfies the kth clause of F.*

Proof. Consider any k, $1 \le k \le m$. Recall from Section 4.3 that each of the seven subgraphs $\mathcal{Z}_{k,1}, \mathcal{Z}_{k,2}, \ldots, \mathcal{Z}_{k,7}$ "represents" a different one of seven distinct conjuncts obtained by negating or not negating each of the three literals in the kth clause, such that exactly one of these seven conjuncts is true in any satisfying assignment. Assume that subgraph $\mathcal{Z}_{k,j}$ completes by time $(D-3)$ in the correct schedule S; since its makespan is equal to $(D-3)$, it must have been executed without interruption during the interval $[0, D-3]$ in S. Let us denote the three literals in the conjunct represented by $\mathcal{Z}_{k,j}$ as $\widehat{\ell}_1, \widehat{\ell}_2$, and $\widehat{\ell}_3$ (i.e., $\mathcal{Z}_{k,j}$ represents the conjunct $(\widehat{\ell}_1 \wedge \widehat{\ell}_2 \wedge \widehat{\ell}_3)$). Then (as described in. Section 4.3) subgraph $\mathcal{Z}_{k,j}$ is a chain with makespan $(D-3)$, with three additional incoming edges: for $\varrho = 1, 2, 3$,

- if literal $\widehat{\ell}_\varrho$ equals x_i ($\neg x_i$) then there is an edge from vertex X_i (vertex $\neg X_i$) to the $(6i+4)$th vertex of $\mathcal{Z}_{k,j}$;
- if literal $\widehat{\ell}_\varrho$ equals y_i ($\neg y_i$) then there is an edge from vertex Y_i (vertex $\neg Y_i$) to the $(6i+4)$th vertex of $\mathcal{Z}_{k,j}$.

Since $\mathcal{Z}_{k,j}$ executes with no interruption in the schedule S, it follows that none of these three edges delay the execution of $\mathcal{Z}_{k,j}$. Hence, if any of these three edges is from one of the vertices

X_i, $\neg X_i$, Y_i, or $\neg Y_i$, then that vertex from which this edge is incoming has completed execution by time-instant $(6i+3)$ in S, and hence the corresponding literal takes on a truth value TRUE in $A_S(F)$. From this, we conclude that the truth assignment $A_S(F)$ assigns the truth value TRUE to each of the three literals $\widehat{\ell}_1$, $\widehat{\ell}_2$, and $\widehat{\ell}_3$, and therefore $A_S(F)$ is a satisfying assignment for the kth clause, as claimed. □

CLAIM 2.4. *For any k, $1 \leq k \leq m$, at most one of the seven subgraphs $\mathcal{Z}_{k,1}, \mathcal{Z}_{k,2}, \ldots, \mathcal{Z}_{k,7}$ may complete by time-instant $(D-3)$ in any correct schedule S of F.*

PROOF. Assume that subgraph $\mathcal{Z}_{k,j}$ completes by time $(D-3)$ in the correct schedule S, and consider any $j' \in \{1, 2, \ldots, 7\}, j' \neq j$. As in the proof of Claim 2.3 above, let us denote the conjunct represented by $\mathcal{Z}_{k,j}$ as $(\widehat{\ell}_1 \wedge \widehat{\ell}_2 \wedge \widehat{\ell}_3)$; furthermore, let us denote the conjunct represented by $\mathcal{Z}_{k,j'}$ as $(\widehat{\ell}'_1 \wedge \widehat{\ell}'_2 \wedge \widehat{\ell}'_3)$. Since exactly one of the seven conjuncts represented by the seven subgraphs $\mathcal{Z}_{k,1}, \mathcal{Z}_{k,2}, \ldots, \mathcal{Z}_{k,7}$ is TRUE in any truth assignment that causes the kth clause to evaluate to TRUE (as was described in Section 4.3) and none are TRUE in a truth assignment that causes the kth clause to evaluate to FALSE, it follows that both these conjuncts cannot be true under the same truth assignment. This in turn implies that one or more of the three literals $\widehat{\ell}_1$, $\widehat{\ell}_2$, and $\widehat{\ell}_3$ is the negation of one or more of the three literals $\widehat{\ell}'_1$, $\widehat{\ell}'_2$, and $\widehat{\ell}'_3$, i.e., there is some integer φ, $1 \leq \varphi \leq n$, such that x_φ is a literal in one of the two conjuncts and $\neg x_\varphi$ is a literal in the other, and/ or y_φ is a literal in one of the two conjuncts and $\neg y_\varphi$ is a literal in the other. We consider four possible cases:

(1) Let us first consider the case where x_φ is one of the literals in the conjunct $(\widehat{\ell}_1 \wedge \widehat{\ell}_2 \wedge \widehat{\ell}_3)$ and $\neg x_\varphi$ is one of the literals in conjunct $(\widehat{\ell}'_1 \wedge \widehat{\ell}'_2 \wedge \widehat{\ell}'_3)$. We saw in the proof of Claim 2.3 that for $\mathcal{Z}_{k,j}$ to complete by time-instant $(D-3)$ it is necessary that the vertex X_φ of subgraph $\mathcal{X}_\varphi$ complete by time-instant $6(\varphi - 1) + 3$. It must similarly be the case that for $\mathcal{Z}_{k,j'}$ to complete by time-instant $(D-3)$ it is necessary that the vertex $\neg X_\varphi$ complete by time-instant $6(\varphi-1)+3$. But we saw in Claim 2.1 that this is impossible—at most one of these two vertices may complete by time-instant $6(\varphi - 1) + 3$. Therefore, in this case, we cannot have both subgraphs $\mathcal{Z}_{k,j}$ and $\mathcal{Z}_{k,j'}$ complete execution by time-instant $(D-3)$.

The other three cases:

(2) $\neg x_\varphi$ a literal in conjunct $(\widehat{\ell}_1 \wedge \widehat{\ell}_2 \wedge \widehat{\ell}_3)$ and x_φ a literal in conjunct $(\widehat{\ell}'_1 \wedge \widehat{\ell}'_2 \wedge \widehat{\ell}'_3)$;
(3) y_φ a literal in conjunct $(\widehat{\ell}_1 \wedge \widehat{\ell}_2 \wedge \widehat{\ell}_3)$ and $\neg y_\varphi$ a literal in conjunct $(\widehat{\ell}'_1 \wedge \widehat{\ell}'_2 \wedge \widehat{\ell}'_3)$; and
(4) $\neg y_\varphi$ a literal in conjunct $(\widehat{\ell}_1 \wedge \widehat{\ell}_2 \wedge \widehat{\ell}_3)$ and y_φ a literal in conjunct $(\widehat{\ell}'_1 \wedge \widehat{\ell}'_2 \wedge \widehat{\ell}'_3)$

may be analyzed analogously to again yield the conclusion that both subgraphs $\mathcal{Z}_{k,j}$ and $\mathcal{Z}_{k,j'}$ cannot complete by time-instant $(D-3)$. □

We now complete the proof of Lemma 2.

Proof of Lemma 2. Consider one possible assignment B of truth values to the universally quantified variables of F; we need to show that there exists a truth assignment to existential variables that is compliant with the order of quantifies and satisfies F. Since the CDAG G_F can be scheduled to always complete by its deadline, for all possible outcomes of the n conditional constructs, we consider the correct schedule S for the combinations of outcomes such that the outcome of the ith conditional construct has the same truth value of the ith universally quantified variable in A.

Claim 2.1 ensures that we can define such a truth assignment $A_S(F)$ of variables of F such that values of y_i variables in A and B coincide. Claim 2.2 shows that such assignment is compliant with

the order of alternation of quantifiers. We now show that this assignment of truth values x_i and y_i variables is a satisfying assignment.

Observe (see Figure 4) that there are $6m\gamma$ available processors in $[D-1, D]$ and, therefore, in a correct schedule at most $6m$ of the $\mathcal{Z}_{k,j}$ subgraphs can complete at time D; processor availability over the interval $[D-3, D-1]$ and the fact that $\gamma > 6\beta > 360\alpha$ imply that exactly m of the $\mathcal{Z}_{k,j}$ subgraphs must complete by time $(D-3)$. By Claim 2.4, we know that for each k there is at most one subgraph $\mathcal{Z}_{k,j}$ that completes by time $(D-3)$. This, in conjunction with the above observation, implies that for each k, $1 \leq k \leq m$, exactly one subgraph $\mathcal{Z}_{k,j}$ completes at $(D-3)$; by applying Claim 2.3, we conclude that $A_S(F)$ is a satisfying assignment for the quantified Boolean formula F. □

6 SUMMARY AND OPEN PROBLEMS

We have shown that feasibility analysis of CDAG tasks, a problem that occurs naturally in the analysis of complex multiprocessor real-time systems, is computationally highly intractable—PSPACE complete, and not solvable by polynomial or pseudo-polynomial time algorithms under the assumption that P ≠ PSPACE.

Several interesting questions concerning feasibility analysis for CDAG tasks satisfying additional restrictions remain open. While it can be shown that feasibility analysis for CDAG tasks in which the number of conditional constructs is a priori bounded from above by a constant is NP-complete, we do not know the complexity of feasibility analysis if instead it is the number of processors that is bounded by a constant (notice that our reduction requires the use of polynomially many processors—see Equation (7)). In a similar vein, the complexity is unknown if the number of conditional constructs in any particular path in the CDAG is bounded by a constant—again, our reduction results in a CDAG with a path in which the number of conditional constructs equals the number of universally quantified Boolean variables.

An additional avenue for further research concerns the feasibility analysis of collections of sporadic tasks [3]. While an EXPTIME algorithm for solving this problem has been obtained [5, 6], to our knowledge no lower bound better than NP-hard is known. It would be interesting to determine whether the techniques we have introduced in this article may be adapted to apply to the feasibility analysis of collections of sporadic tasks as well, to prove a better (i.e., higher) lower bound of PSPACE-hardness.

REFERENCES

[1] Sanjoy Baruah. 2020. Feasibility analysis of conditional DAG tasks is co-NP^{NP}-hard (why this matters). In *Proceedings of the 29th International Conference on Real-Time and Network Systems (RTNS'21)*. ACM, 165–172.

[2] Sanjoy Baruah, Vincenzo Bonifaci, and Alberto Marchetti-Spaccamela. 2015. The global EDF scheduling of systems of conditional sporadic DAG tasks. In *Proceedings of the 26th Euromicro Conference on Real-Time Systems (ECRTS'15)*. IEEE Computer Society Press, 222–231.

[3] Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Leem Stougie, and Andreas Wiese. 2012. A generalized parallel task model for recurrent real-time processes. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS'12)*. IEEE Computer Society Press, 63–72.

[4] Sanjoy Baruah and Alberto Marchetti-Spaccamela. 2021. Feasibility analysis of conditional DAG tasks. In *Proceedings of the 33rd Euromicro Conference on Real-Time Systems (ECRTS'21) (Leibniz International Proceedings in Informatics (LIPIcs))*, Björn B. Brandenburg (Ed.), Vol. 196. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 12:1–12:17. https://doi.org/10.4230/LIPIcs.ECRTS.2021.12

[5] Vincenzo Bonifaci and Alberto Marchetti-Spaccamela. 2010. Feasibility analysis of sporadic real-time multiprocessor task systems. In *Proceedings of the 18th Annual European Conference on Algorithms: Part II (ESA'10)*. Springer-Verlag, Berlin, 230–241.

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

[7] 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

[8] Jose Fonseca, Vincent Nelis, Gurulingesh Raravi, and Luis 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 Press, 1925–1932.

[9] M. Garey and D. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, NY.

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

[11] Robert A. Hearn and Erik D. Demaine. 2009. *Games, Puzzles and Computation*. A K Peters.

[12] Klaus Jansen. 1994. Analysis of scheduling problems with typed task systems. *Discrete Appl. Math.* 52, 3 (1994), 223–232.

[13] Alberto Marchetti-Spaccamela, Nicole Megow, Jens Schlöter, Martin Skutella, and Leen Stougie. 2020. On the complexity of conditional DAG scheduling in multiprocessor systems. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS'20)*. 1061–1070.

[14] Alessandra Melani, Marko Bertogna, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Giorgio Buttazzo. 2015. Response-time analysis of conditional DAG tasks in multiprocessor systems. In *Proceedings of the 26th Euromicro Conference on Real-Time Systems (ECRTS'15)*. IEEE Computer Society Press, 222–231.

[15] L. Stockmeyer. 1976. The polynomial-time hierarchy. *Theoret. Comput. Sci.* 3 (1976), 1–22.

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

[17] C. Wrathall. 1976. Complete sets and the polynomial-time hierarchy. *Theoret. Comput. Sci.* 3 (1976), 23–33.

[18] Houssam-Eddine Zahaf, Nicola Capodieci, Roberto Cavicchioli, Marko Bertogna, and Giuseppe Lipari. 2020. The HPC-DAG task model for heterogeneous real-time systems. *IEEE Trans. Comput.* 70, 10 (2020), 1747–1761.

Received 19 August 2022; accepted 27 June 2023