RZ 3884 (# ZUR1410-050) 10/17/2014 (Revised May 2015)
Computer Sciences/Mathematics 18 pages

Research Report

The Complexity of Deadline Analysis for Workflow Graphs with a Single Resource (Revised Version)

Mirela Botezatu[1,2], Hagen Völzer[1] and Lothar Thiele[2]

[1]IBM Research – Zurich
8803 Rüschlikon
Switzerland

[2]ETH Zurich
Switzerland

The Complexity of Deadline Analysis for Workflow Graphs with a Single Resource

Mirela Botezatu[1,2], Hagen Völzer[1], and Lothar Thiele[2]

[1] IBM Research – Zurich, Switzerland
[2] ETH – Zurich, Switzerland

Abstract. Workflow graphs (WFGs) are control-flow graphs extended by parallel fork and join. They are used to represent the main control-flow of e.g. business process models modeled in languages such as BPMN or UML activity diagrams. A WFG is said to be *sound* if it is free of deadlocks and exhibits no lack of synchronization. We study the question whether the executions of a time-annotated sound WFG meet a given deadline. We present polynomial-time algorithms and NP-hardness results for different cases. In particular, we show that it can be decided in polynomial time whether some executions of a sound WFG meet the deadline. Furthermore we show that for general probabilistic WFGs, it is NP-hard to determine whether the probability of an execution meeting the deadline is higher than a given threshold, whereas the expected duration of an execution can be computed in polynomial time.

1 Introduction

Workflow graphs can capture the main control flow of processes modeled in languages such as BPMN, UML-AD, and EPC [15]. That is, the core routing constructs of these languages can be mapped to the routing constructs of WFGs, which are alternative choice and merge, and concurrent fork and join. Fig. 1 shows an example of a WFG modeling a data analysis workflow. After a task to read the data, there is an alternative choice $s1$ whether the read data is of type 1 (DT1) or type 2 (DT2). After a preprocessing task for each type, there is a concurrent fork $f1$, and different tasks (Feature Selection, Train SVM, Train RF) are executed in part concurrently to each other and separately for each of the two data types. Finally, two tasks are executed concurrently that are independent of the data type (Predict SVM and Predict RF) after which the concurrent threads are synchronized through a concurrent join $j1$ and two tasks are executed after the synchronization to return the results.

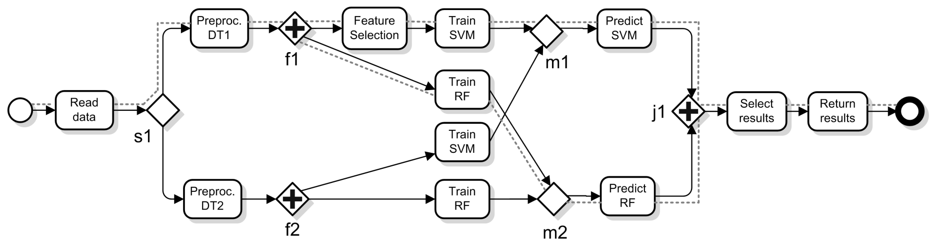

Fig. 1: An example of a WFG and one of its executions (dotted)

A workflow graph is equivalent to a two-terminal Free-Choice Petri net i.e., a connected net with a unique source and sink, which is also called a *free-choice workflow net* [7]. A workflow graph can be seen as a compact representation of the corresponding free-choice net. Therefore, the theory of free-choice Petri nets directly applies to workflow graphs.

A workflow graph may contain a deadlock or *lack of synchronization*. The latter corresponds to unsafeness in Petri nets. The absence of deadlock and lack of synchronization has been termed *soundness*, which can be decided in cubic time by help of the rank theorem for free-choice Petri nets [4].

In this paper, we study real-time analysis for WFGs, in particular, whether the executions of a sound WFG meet a given deadline, where tasks, or, equivalently, edges are annotated with execution times. We are not aware of any similar work for this model class. We restrict here to the case where all tasks are executed by a single resource, i.e., the time that is needed to execute two concurrent tasks is the sum of the times needed for each task. The case where a WFG is executed by multiple resources is left as future work.

Table 1 shows the results, where our contributions are written in bold. The first two columns refer to the question whether all executions or some execution of the WFG finish before the deadline, respectively. In the former case, cycles in the graph are constrained by a termination order. For the third and fourth columns of Table 1, alternative choice is resolved by a coin flip and we ask whether the probability to terminate within the given deadline is above a given threshold (third column) or what the expected duration is (fourth column).

Sequential graphs refers to the subclass of classical control-flow graphs without concurrency. We can use Dijkstra's algorithm [6] for computing the shortest path of a sequential graph and therefore determine the minimum duration (Cell B.2). If the sequential graph is acyclic, then the shortest and the longest path can be computed in linear time through a combination of topological sort and dynamic programming [20] (Cells A.1 and A.2). To define the maximum duration of a WFG, we need to constrain the number of times a loop can be traversed. We propose a general model of loop constraints for cyclic WFG in this paper. For this general model, we adapt the known result that computing the longest simple path in a sequential graph is NP-hard to show that it is NP hard to compute whether all admissible executions of a sequential WFG meet a given deadline (Cells B.2 and E.2).

The expected duration of a probabilistic sequential graph, i.e., Markov chain, can be computed in polynomial time [3] (Cell B.4), and again there is a linear time solution for the acyclic case [3] (Cell A.4). *Regular graphs* refers to the subclass where the graph can be generated by a regular expression, i.e., every split corresponds to a join of the same logic (alternative or concurrent), see Fig. 3 for an example. For regular workflow graphs, solutions are simple recursive algorithms which we briefly mention in the paper and which run in linear time (Cells C.1, C.2, and C.4).

General WFGs can of course be analyzed for timing behavior in terms of their reachability graph, and there are various techniques and tools that support this [9, 11, 19]. However, since the construction of the reachability graph incurs an exponential blowup, these techniques do not run in polynomial time. In this paper, we show that some deadline analysis problems for WFGs can nevertheless be solved in polynomial time. Specifically, we show that Cells E.2, E.4 can be solved in polynomial time, whereas Cells D.1, D.2 and D.4 can be solved in linear time. Furthermore, we show that computing the probability of transgressing a deadline is NP-hard even for sequential regular graphs, while it is known that there is a pseudo-polynomial solution for sequential graphs.

	1. All executions	2. Some execution	3. Probability of transgression	4. Expected duration
A. Acyclic Sequential WFG	$O(\|V\| + \|E\|)$	$O(\|V\| + \|E\|)$	Weakly **NP-hard**	$O(\|V\| + \|E\|)$
B. Sequential WFG	NP-hard	$O(\|E\|+\|V\| \cdot log\|V\|)$	**As in A.3**	$O(\|E\|^3)$
C. Regular WFG	$O(\|V\| + \|E\|)$	$O(\|V\| + \|E\|)$	**As in A.3**	$O(\|V\| + \|E\|)$
D. Acyclic Sound WFG	$\boldsymbol{O(\|V\| + \|E\|)}$	$\boldsymbol{O(\|V\| + \|E\|)}$	**NP-hard**	$\boldsymbol{O(\|V\| + \|E\|)}$
E. Sound WFG	As in B.1	$\boldsymbol{O(\|V\|\|E\|)}$	**As in D.3**	$\boldsymbol{O(\|E\|^3)}$

Table 1: Overview of results; new contributions in bold.

The paper is structured as follows. After the preliminaries, we present a new algorithm for computing the minimum duration for a given WFG in polynomial time, and we present the NP-hardness proof for computing the maximum duration execution in Sect. 3. In Sect. 4, we present the hardness result for assessing the probability of deadline transgression and the polynomial time algorithm for computing the expected duration for probabilistic cyclic WFG.

2 Preliminaries

In this section, we introduce workflow graphs, their semantics and their various subclasses.

A *weighted, directed multi-graph* $G = (V, E, c, w)$ consists of a set of nodes V, a set of edges E, a mapping $c : E \rightarrow V \times V$ that maps each edge to an ordered pair of nodes and a mapping $w : E \rightarrow \mathbb{N}$ that maps each edge to a nonnegative integer, called its *weight* or *duration*. For each edge e with $c(e) = (v, z)$, we assume $v \neq z$ for simplicity throughout the paper.

A *workflow graph* $\Gamma = (V, E, c, l, w)$, is a weighted multi-graph $G = (V, E, c, w)$ with distinct and unique source and sink nodes, denoted v_{source} and v_{sink}, respectively, equipped with an additional mapping $l : V \setminus \{v_{source}, v_{sink}\} \rightarrow \{\text{XOR}, \text{AND}\}$ that associates a *branching logic* with every node, except for the source and the sink. Furthermore, we assume that every node is on a path from the source to the sink, that the source has a unique outgoing edge, called the *source edge* (e_{source}), and that the sink has a unique incoming edge, called the *sink edge* (e_{sink}). For each node v, we define the *pre-set* of v, ${}^\bullet v = \{e \in E \mid \exists w \in V : c(e) = (v, z)\}$ and the post-set of v, $v^\bullet = \{e \in E \mid \exists\, z \in V : c(e) = (v, z)\}$. A node v where $|{}^\bullet v| > 1$ or $|v^\bullet| > 1$ is called a *gateway*, in the former case, v is called a *join* and in the latter case, a *split*.

Fig. 1 shows a WFG in BPMN notation: An XOR gateway is depicted as a diamond, an AND gateway as a diamond decorated with a plus sign. Source and sink are depicted as circles. A node that is neither a join, split, nor source or sink is usually called a *task*. A task is shown as a rounded rectangle in Fig. 1. It is natural to assign durations to tasks. However, we will henceforth omit tasks for simplicity and annotate each edge with a duration $w(e)$ as formalized above.

A *marking* $m : E \to \mathbb{N}$ of a WFG maps each edge to a non-negative integer. If $m(e) = i$, we say that there are *i tokens* on edge e. The marking with exactly one token on the source edge and no token elsewhere is called the *initial marking*. The marking with exactly one token on the sink edge and no token elsewhere is called the *final marking* of the WFG.

The *semantics* of workflow graphs is defined as a token game as it is in Petri Nets. A comprehensive analysis of the relationship between workflow graphs and free-choice workflow nets (a subclass of Petri nets) can be found in [7]. The execution of a node with an AND logic removes one token from each of its incoming edges and adds one token to each of the outgoing edges. The execution of a node with a XOR logic removes non-deterministically a token from one of its incoming edges that has a token, then non-deterministically adds one token to one of the outgoing edges. Although we omit tasks, we allow nodes with just one incoming and one outgoing edge for technical reasons. For such nodes, XOR and AND logic behaves the same. In figures, we depict such a node as an XOR node.

More formally, we define the relation $m \overset{v}{\to} m'$, pronounced v is *enabled* in m and execution of v in m results in m', for a pair m, m' of markings and a node v as follows: $l(v)$=AND and

$$m'(e) = \begin{cases} m(e) - 1, & \text{if } e \in {}^{\bullet}v \text{ and } m(e) > 0 \\ m(e) + 1, & \text{if } e \in v^{\bullet} \\ m(e), & \text{otherwise} \end{cases}$$

$l(v)$=XOR and there exists an $e' \in {}^{\bullet}v$ and an $e'' \in v^{\bullet}$ such that:

$$m'(e) = \begin{cases} m(e) - 1, & \text{if } e = e' \text{ and } m(e) > 0 \\ m(e) + 1, & \text{if } e = e'' \\ m(e), & \text{otherwise} \end{cases}$$

We write $m \to m'$ if $m \overset{v}{\to} m'$ for some v and $\overset{*}{\to}$ for the transitive and reflexive closure of $\to$. We say m' is *reachable* from m if $m \overset{*}{\to} m'$.

An *execution* σ of a WFG Γ is a sequence of markings of Γ, $\sigma = m_0, m_1, m_2, \ldots$ such that m_0 is the initial marking of Γ and for each $i \geq 0$, $m_i \to m_{i+1}$. If there is no marking m such that $m_n \to m$, then we call the execution $\sigma = m_0, ..., m_n$ finite.

A reachable marking m is a *deadlock* if m has a token on an incoming edge $e \in E$ of an AND join such that each marking reachable from m also contains a token on e. A reachable marking m is *unsafe* or has *lack of synchronization* if one edge has more than one token in m. A workflow graph is said to be *sound* if it has no *deadlock* and no unsafe reachable marking. Soundness guarantees that every finite execution terminates in the final marking of Γ. Soundness has various equivalent characterizations and can be decided in polynomial time [2, 4].

Let Γ be a WFG; Γ is *sequential* if it contains no AND-split or -join, it is *acyclic* if the underlying graph has no cycles. A *regular* WFG is a WFG that can be generated from a regular expression as follows. Let ϵ be a constant symbolizing an edge and X, Y variables for WFGs. Then a regular WFG expression is the smallest set such that ϵ is a regular WFG, and if X and Y are regular WFG, then X ; Y, X AND Y, X XOR Y, and X LOOP Y are also regular WFG. From each regular WFG expression, we can generate a WFG, where each expression type corresponds to one of the graph fragment patterns shown in Fig. 2 and composition is done by replacing an edge labeled with a variable by another pattern. For example, the expression $(\epsilon;\epsilon)$ AND $(\epsilon$ $LOOP$ $\epsilon)$ generates

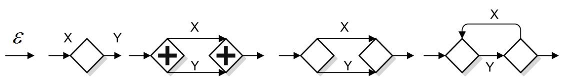

Fig. 2: Regular patterns

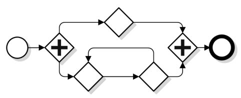

Fig. 3: Regular graph

the graph shown in Fig. 3. Note that the loop construct has two loop bodies. It can be viewed as a combination of a while and a repeat loop, one loop body before the loop condition one after it. It can be decided in linear time whether a WFG is a regular WFG using graph parsing techniques [16].

3 Workflow graphs with nondeterministic choice

In this section, we study deadline analysis where choices in the WFG are assumed to be nondeterministic. We distinguish two cases. First, we assume that choices are made by the process internally, e.g., based on data-based decisions that we abstract from. In this case, we are interested in whether the process always terminates within the deadline, i.e., whether all its executions meet the deadline. Secondly, we consider that the choices are made by a superimposed scheduler that has the goal to make choices in order to meet the deadline, i.e., we ask whether there exists an execution that meets the deadline. It is clear that for the first case, it is sufficient to compute the maximum duration execution and for the second case, the minimum duration execution.

In the following, we present the polynomial time algorithm to determine the minimum duration of an execution of a WFG, Sect. 3.1. In Sect. 3.2, we define a system model to rule out infinite executions of a WFG and adapt the known NP-hardness result of the longest path problem to that system model. Finally, we show in Sect 3.3 that if we restrict to acyclic WFGs the minimum or maximum duration of an execution can be computed in linear time.

3.1 The minimum duration of a WFG

In the following, we present an algorithm to compute the minimum duration execution of a WFG. WE will start by presenting several preliminaries that are needed for the algorithm.

Let Γ be a sound workflow graph.

Since we assume a single resource, the *duration of an execution* σ, of a WFG, which we denote as $c(\sigma)$, can be defined by:

$$c(\sigma) = \sum_{e \in E} w(e) \cdot \sigma(e). \tag{1}$$

where

$$\sigma(e) = \begin{cases} |\{i \mid m_i, m_{i+1} \text{ consecutive markings of } \sigma, m_i(e) < m_{i+1}(e)\}|, & \text{if } e \in E \setminus \{e_{source}\} \\ 1 & \text{if } e = e_{source} \end{cases} \tag{2}$$

is the number of times edge e is marked in an execution σ, which is either 0 or 1 if the WFG is acyclic, but can be any nonnegative integer or ∞, otherwise.

We will re-write the cost of an execution (1) in terms of recursive equations, that represent the *accumulated cost* for reaching the sink, for a chosen execution.

As an example, consider the WFG in Fig. 4. In Fig. 4, edges are labeled (e.g. $e8$; 2) with an edge name ($e8$) and a duration (2). Fig. 5 represents the WFG restricted to the elements that are contained in the execution with minimum duration, i.e., it is a representation of the minimum duration execution. Each edge in Fig. 5 is labeled with the accumulated cost for reaching the sink.

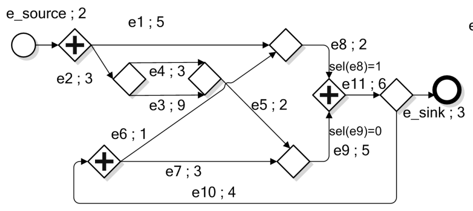

Fig. 4: WFG with edge weights

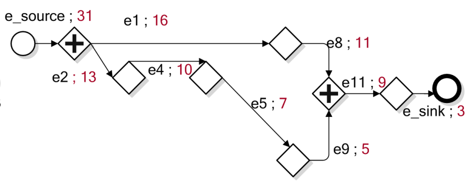

Fig. 5: Minimum duration execution and the accumulated costs

For e_{11}, the accumulated cost to reach the sink is: $w(e_{11})$ to which we add the cost of e_{sink} therefore, $6 + 3 = 9$. Since in any execution in which edge e_8 is marked, edge e_9 is also marked we accumulate the cost to reach the sink on only one of the edges (otherwise the cost is over-counted). The cost associated to e_9 stays $w(e_9)$ and the cost associated to e_8 becomes $w(e_8)$ plus the cost of e_{11}, and we obtain 2+9=11, etc. The duration of the minimum duration execution (Fig. 5) equals the cost accumulated to e_{source} which is 31.

As presented in the example, for each node v such that $l(v)$ =AND, and $|^\bullet v| > 1$, the cost associated to the outgoing edge has to be accumulated only once. For this, we define a mapping $sel : E \to \{0, 1\}$ that specifies which of the incoming edges of an AND node v, for which $|^\bullet v| > 1$, accumulates the cost of the outgoing edge. Note that for all the v of the type mentioned above, $\sum_{e \in ^\bullet v} sel(e) = 1$.

The recursive definition for the cost of an execution is technically simpler for a *loop-free execution*.

Definition 1 *An execution σ, of a WFG Γ, is a* loop-free execution *if no node is executed more than once in σ, and implicitly no edge is marked more than once in σ. Let $LF = \{\sigma \mid \sigma$ is a loop-free execution of $\Gamma\}$.*

It sufficies for us to restrict to loop-free executions due to the following lemma:

Lemma 1 *The execution with minimum duration of a WFG is a loop-free execution.*

The proof of the Lemma is presented in the Appendix.

We can finally write the recursive equations for defining the accumulated cost and the cost of an execution.

Let σ be a loop-free execution of a WFG. For each node v, and for each edge $e \in {}^\bullet v$, we define recursively $d_\sigma(e)$ - the *accumulated cost* for reaching the sink for e in σ. $d_\sigma(e_{sink}) = w(e_{sink})$ and for $e \in E \setminus \{e_{sink}\}$, if $\sigma(e) = 0$, then $d_\sigma(e) = 0$, otherwise, if $\sigma(e) = 1$ as follows:

$$d_\sigma(e) = \begin{cases} w(e) + d_\sigma(e') & \text{if } l(v) = \text{XOR and } |v^\bullet| > 1 \text{ and } e' \in v^\bullet \text{ s.t. } \sigma(e') = 1 \\ w(e) + d_\sigma(e') & \text{if } l(v) = \text{XOR and } |v^\bullet| = 1, \text{ and } \{e'\} = v^\bullet \\ w(e) + \sum_{e' \in v^\bullet} d_\sigma(e') & \text{if } l(v) = \text{AND and } |v^\bullet| > 1 \\ w(e) + d_\sigma(e') & \text{if } l(v) = \text{AND and } |v^\bullet| = 1, \{e'\} = v^\bullet \text{ and } sel(e') = 1 \\ w(e) & \text{if } l(v) = \text{AND and } |v^\bullet| = 1, \{e'\} = v^\bullet \text{ and } sel(e) = 0 \end{cases} \tag{3}$$

Due to the inductive definition of $d_\sigma(e)$, it holds that $d_\sigma(e_{source}) = c(\sigma)$, the duration of a loop-free execution σ (i). From Lemma 1, it holds that the minimum duration execution is loop-free (ii). From (i) and (ii) it holds that the duration of the minimum duration execution is $min\{d_\sigma(e_{source}) \mid \sigma \in LF\}$.

We define $d^*(e)$, *the minimum cost downstream from* e, as follows:

$$d^*(e) = \begin{cases} min\{d_\sigma(e) \mid \sigma(e) = 1 \land \sigma \in LF\}, & \text{if } \{\sigma \mid \sigma \in LF \land \sigma(e) = 1\} \neq \emptyset \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

Since $\sigma(e_{source}) = 1$ for any (loop-free) execution σ, $d^*(e_{source}) = min\{d_\sigma(e_{source}) \mid \sigma \in LF\}$ and therefore $d^*(e_{source})$ is the duration of the minimum duration execution.

The algorithm for computing the minimum duration of an execution of a WFG, works on a weighted WFG, and for each node v, for each edge $e \in {}^\bullet v$ it updates a value $\delta(e)$ that represents the currently known minimum duration of an execution to reach the sink from e. Upon termination of our algorithm, the value associated to e_{source}, $\delta(e_{source})$, represents the duration of the minimum duration execution. The idea is similar to the Bellman-Ford algorithm [1] for sequential graphs, but the edge relaxation procedure is different, to reflect the semantics of sound WFGs.

The algorithm that computes the duration of the minimum duration execution is Algorithm 1, which contains a call to the subroutine represented by Algorithm 3.

Next, we will show the correctness of the algorithm.

Algorithm 1 Minimum duration

```
function WFGMin( V, E)
    for $e \in E \setminus \{e_{sink}\}$ do
        $\delta(e) \leftarrow \infty$
    end for
    $\delta(e_{sink}) \leftarrow w(e_{sink})$
    for $i = 1 : |V|$ do
        for all $v \in V$ do
            for all $e \in {}^\bullet v$ do
                Relax(e,v)
            end for
        end for
    end for
end function
```

Algorithm 2 Min duration, acyclic

```
function AcyclicWFGMin(V, E)
    for $e \in E \setminus \{e_{sink}\}$ do
        $\delta(e) \leftarrow \infty$
    end for
    $\delta(e_{sink}) \leftarrow w(e_{sink})$
    TopologicalSort($\Gamma$)
    while $V \neq \emptyset$ do
        Select $v \in V$ s.t. $v$ is maximal with respect to the topological sort
        $V \leftarrow \{V \setminus v\}$
        for all $e \in {}^\bullet v$ do
            Relax(e,v)
        end for
    end while
end function
```

Algorithm 3 Relaxation of an edge $e \in {}^\bullet v$

```
function Relax(e,v)
    if $l(v) = \text{XOR}$, $|v^\bullet| = 1$, and $\{e'\} = v^\bullet$ then
        if $\delta(e) > w(e) + \delta(e')$ then
            $\delta(e) \leftarrow w(e) + \delta(e')$
        end if
    end if
    if $l(v) = \text{XOR}$, $|v^\bullet| > 1$, and $e' \in v^\bullet$ then
        if $\delta(e) > w(e) + min_{e'}(\delta(e'))$ then
            $\delta(e) \leftarrow w(e) + min_{e'}(\delta(e'))$
        end if
    end if
    if $l(v) = \text{AND}$, $|v^\bullet| = 1$, and $\{e'\} = v^\bullet$ then
        if $\delta(e) > w(e) + \delta(e')$ then
            if $sel(e) = 1$ then
                $\delta(e) \leftarrow w(e) + \delta(e')$
            else
                $\delta(e) \leftarrow w(e)$
            end if
        end if
    end if
    if $l(v) = \text{AND}$ and $|v^\bullet| > 1$ then
        if $\delta(e) > w(e) + \sum_{e' \in v^\bullet} \delta(e')$ then
            $\delta(e) \leftarrow w(e) + \sum_{e' \in v^\bullet} \delta(e')$
        end if
    end if
end function
```

Lemma 2 *After each call of Relax(e, v) it holds that $\delta(e) \geq d^*(e)$. Each relaxation of an edge e can only decrease the current value of $\delta(e)$.*

The proof of the Lemma is presented in the Appendix.

For a workflow graph Γ and one loop-free execution σ of Γ, we define Γ_σ as the workflow graph Γ restricted to σ such that it contains only the nodes of Γ that are executed in σ and the edges of Γ such that $\sigma(e) = 1$. Note that for a workflow graph Γ and σ - a loop-free execution of Γ it follows that Γ_σ is an acyclic workflow graph.

The elements of an acyclic workflow graph are in a partial order defined by the flow of the graph: Let $G = (V, E, c)$ be an acyclic multi-graph. If x_1, x_2 are two distinct elements in $V \cup E$ such that there is a path from x_1 to x_2, then we say that x_1 precedes x_2, denoted $x_1 \leq x_2$, and x_2 follows x_1.

Lemma 3 *Let e be an edge of a workflow graph Γ for which $\{\sigma \mid \sigma \in LF \wedge \sigma(e) = 1\} \neq \emptyset$. Let $S =< e_{i-1} \cdots, e_{sink} >$ be the edges that get marked after e gets marked, in an execution $\sigma \in LF$ for which $\sigma(e) = 1$ and $d_\sigma(e) = d^*(e)$. Each sequence of calls of $Relax(e, v)$ that has the property that edges $e_{sink}, \cdots, e_{i-1}, e$ have been relaxed in decreasing order with respect to $\preceq$ on Γ_σ, leads to $\delta(e) = d^*(e)$.*

The proof of the lemma is presented in the Appendix.

Lemma 4 *For a sound workflow graph, after running the Algorithm 1 for computing the minimum duration execution, it holds that for any e such that $\{\sigma \mid \sigma(e) = 1, \sigma \in LF\} \neq \emptyset$, $\delta(e) = d^*(e)$.*

Proof: Let e be an edge of the workflow graph such that $\{\sigma \mid \sigma(e) = 1, \sigma \in LF\} \neq \emptyset$. Let $\sigma^* = argmin_\sigma d_\sigma(e)$, $\sigma \in LF, \sigma(e) = 1$.

Since σ^* is loop-free, it means that at most $|V|$ nodes are executed in σ^*. In each complete relaxation step (one iteration of the loop on line 6 in Algorithm 1) we relax all the edges. Therefore, at the $|V|$-th iteration we have relaxed all the edges, in decreasing order with respect to the partial order on the edges of Γ_{σ^*}. It means that at the $|V|$-th iteration, we will have relaxed all the edges that get marked after e gets marked in σ^*. Therefore, from Lemma 3, $\delta(e) = d^*(e)$. □

Since $e_{source} \in \{e \mid \{\sigma \mid \sigma(e) = 1, \sigma \in LF\} \neq \emptyset\}$ it holds, given Lemma 4, that after running Algorithm 1, $\delta(e_{source}) = d^*(e_{source})$. We computed the duration of the minimum duration execution of the workflow graph, which is $d^*(e_{source})$.

For Algorithm 1, the initialization of the edge costs takes $O(|V|)$ time and each of the $|V|$ iterations over the edges of the WFG is performed in $O(|E|)$ time. The cost update is performed also in $O(|E|)$ time. Hence we have proven the following:

Theorem 1 *1 The minimum duration execution of a sound workflow graph can be computed in polynomial time $O(|V||E|)$.*

3.2 The maximum duration of a WFG

To define the maximum duration of a WFG, we need to constrain the number of times a loop can be traversed. We propose a general model of loop constraints for cyclic WFG in this section. For this general model, we adapt the known result that computing the longest simple path in a sequential graph is NP-hard to show that it is NP hard to compute whether all admissible executions of a sequential WFG meet a given deadline.

For graphs with *reducible loops* as produced by the structured constructs (while, repeat loops) where retreating edges are unique, (back edges), loop bounds represent the maximum iteration count of the loop body relative to the header. In a WFG however, there may be *irreducible loops* - loops with multiple entry points.

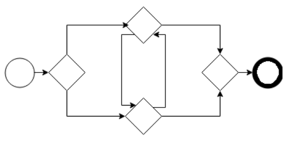
Fig. 6: A WFG with irreducible loops

For irreducible loops, the specification of loop bounds is more complex [10] due to the multiple loop entries. An example of a graph containing irreducible loops is given in

Fig. 6. As a generalization of the limitation of loop bounds in regular graphs, we define a system model that rules out infinite executions and show that computing the maximum duration execution for workflow graphs with cycles is NP-hard, as follows:

Assume a sound WFG Γ is given. We furthermore assume that the following two specifications are given with Γ:

1. a partitioning $V_0, V_1, \cdots, V_k$ of the nodes V of Γ, where each V_i is called a *termination layer* such that V_0 contains only the sink, and
2. a mapping $\phi : E \rightarrow \mathbb{N} \cup \{\infty\}$, called an *decision outcome traversal restriction*, which denotes the maximal number of times an edge may be traversed in an admissible execution such that

(i) each XOR-split v has an unrestricted *outcome edge*, i.e., an edge $e \in v^{\bullet}$, such that $\phi(e) = \infty$, and (ii) each unrestricted edge leads into a lower termination layer, i.e., $\phi(e) = \infty$ where $c(e) = (v, w)$ implies $v \in V_i$, $w \in V_j$ where $i > j$.

A workflow graph with these properties is called a workflow graph with loop constraints.

We then call an execution σ of Γ *admissible* if for each outcome edge o of Γ, $\sigma(o) \leq \phi(o)$, i.e., if the specified traversal restrictions are obeyed by σ. Condition (ii) above guarantees that each admissible execution terminates. Condition (i) requires that the restrictions given by ϕ do not create an artificial deadlock, i.e., a partial execution that cannot be extended into an admissible execution. Hence, an admissible execution always exists. Any termination order that satisfies these two natural requirements may be specified using the node partition and ϕ. We now present a hardness result for sequential WFGs with cycles:

Theorem 2 *The problem to determine whether all admissible executions of a sequential WFG with loop constraints meet a given deadline is NP-hard.*

Proof. We reduce from the problem of computing the longest simple path (between any nodes) in a directed graph, which in turn is a reduction from the Hamiltonian path problem. Given such a directed graph G, we construct an annotated sequential WFG Γ as follows, cf. Fig. 7.

First we expand each node of G that is a split as well as a join (e.g., the two interior nodes of G in Fig. 7) into a separate join and a separate split with a single edge from the join to the split. These added edges are weighted with duration 0. The obtained graph is called G', cf. subgraph of the right hand side graph in Fig. 7 encircled with label $V2$. Note that each path in G' corresponds to a path in G of the same duration and vice versa.

We add a fresh source and a fresh sink, we add an edge from the source to each node in G' and an edge from each node in G' to the sink, which all have duration 0 and are unrestricted ($\phi(e) = \infty$). The termination layers are specified as in Fig. 7, all edges in G' are restricted with 1, i.e., must not be traversed more than once. It is easy to check that all conditions of the system model above are met.

Suppose the maximum duration, admissible execution in Γ can be computed in polynomial time. That execution is a path in Γ and due to the construction of Γ, it contains the longest duration path π' of G' between any nodes that contains each edge of G' at most once. This path π' of G' corresponds to a path π in G of the same duration that contains each node at most once and π must be the longest path of G with that property. Any longer path of G that visits each node at most once would correspond to path in G' of the same duration that visits every edge at most once. Hence we would be able to solve the longest simple path problem in polynomial time which is known to be NP-complete.□

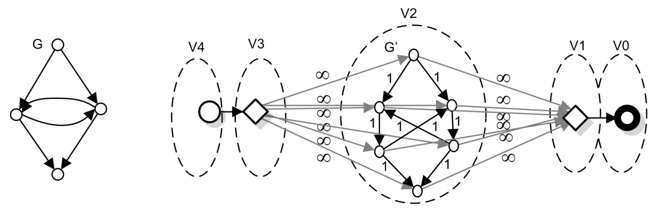

Fig. 7: Constructed WFG for proof of Thm. 2

Thm. 2 settles Cells B.1 and E.1 of Table 1.

3.3 Regular and acyclic WFG

For a regular WFG, with a structured cycle, i.e., a while or repeat loop, or more general, of the form X LOOP Y, in order to compute the maximum duration, one needs the number of iterations for each loop. If we assume that the backedge of each loop of the regular graph is annotated with a positive integer k that represents the maximum number of times the backedge can be traversed, then the maximum duration of X LOOP Y is $(k + 1) \cdot d_X + k \cdot d_Y$ where d_X denotes the maximum duration of the loop body X, and d_Y represents the duration associated to reentering the loop. For computing the minimum duration we take $k = 0$. We still obtain the minimum/maximum duration of such an annotated regular WFG in linear time (Cell C.1, C.2 of Table 1).

If the graph is not regular but it is sequential, the minimum duration execution can be computed using Dijkstra's algorithm (Cell B.2 of Table 1).

For the acyclic case, the simplest case is when the graph is regular (e.g., cf. Fig.3), in which we can compute the minimum and maximum durations recursively. If the graph is the sequential or concurrent (AND) composition of its subgraphs, then its duration is the sum of the durations of its subgraphs. If the graph is the alternative (XOR) composition of its subgraphs, then its minimum (maximum) duration is the minimum (maximum) of the minimum (maximum) durations of its subgraphs. We thus obtain an algorithm that runs in $O(|V| + |E|)$ time. If the acyclic graph is not regular, but sequential, there is still a well-known simple solution to finding the longest path between two nodes, which runs in time $O(|V| + |E|)$ (Cell A.1 of Table 1). Analogously, the same algorithm can be applied to compute the shortest simple path by taking the minimum (Cell A.2 of Table 1).

For acyclic WFGs, we can use the algorithm for the cyclic case but without the need to perform $|V|$ iterations. Instead we exploit the fact that the elements of an acyclic

WFG are in a partial order defined by the flow of the graph. Therefore, in order to make sure that the edges are relaxed respecting the partial order, first, the graph is sorted topologically - $O(|V| + |E|)$. Secondly,the edges are relaxed in descending order with respect to the topological sorting $O(|E|)$. The algorithm that formalizes this idea is Algorithm 2.

Theorem 3 *The minimum duration execution of a sound ayclcic workflow graph can be computed in linear time $O(|V| + |E|)$.*

Note that, in the acyclic case, for computing the maximum duration execution, one only needs to select the maximum instead of the minimum in the *Relax*(e, v) procedure when $l(v) = XOR$ and $|v^{\bullet}| > 1$.

4 Workflow graphs with probabilistic choice

If not all executions of a WFG meet the deadline, we could ask whether at least a large portion of the executions does. We approach this question by assuming that decisions are resolved through a coin flip, i.e., each XOR-node v is assigned a distribution $\mu : v^{\bullet} \to [0, 1]$ such that $\mu(e) > 0$ for each $e \in v^{\bullet}$ and $\sum_{e \in v^{\bullet}} = 1$. Although some executions may not terminate, their probability[3] is zero. We can then take the duration of an execution as a random variable and ask whether the probability of an execution terminating before the deadline exceeds a given threshold. We address this question in Sect.4.1 and contrast the obtained results with results on computing the expected duration in Sect.4.2.

4.1 Probability of deadline transgression

We will show that computing whether the probability of an execution terminating before the deadline exceeds a given threshold is NP-hard. The hardness result can be obtained even for the simplest of graphs:

Theorem 4 *Given a regular, sequential, acyclic probabilistic WFG, a deadline $\alpha \in \mathbb{N}$ and a threshold $p \in [0, 1]$, computing $\mathbb{P}(c(\sigma) \leq \alpha) \geq p$ is NP-hard.*

Proof. The proof consists of a reduction from the *subset sum problem*, which is, given a set $\{d_1, \cdots, d_n\}$ of integers and an integer α, determine whether any non-empty subset sums up to exactly α. This problem is known to be NP-hard. Given these parameters

[3] We do not explicitly construct the probability space here on which the development of this chapter is formally based on. As WFGs contain concurrency, we need to consider maximal partial-order executions to obtain a single probability space and to avoid the notion of an adversary as in Markov decision processes. Note that a probabilistic WFG does not contain real non-determinism, just concurrency. The construction of such a probability space is provided elsewhere [17, 18], e.g. for Petri nets and in fact rests on the assumption that the Petri net is free-choice. In this paper, we are only concerned with the duration of an execution, which is independent of the interleaving, i.e., the ordering of concurrent events.

we consider the (regular, acyclic, sequential) probabilistic WFG in Fig. 8, where each decision outcome has probability 0.5.

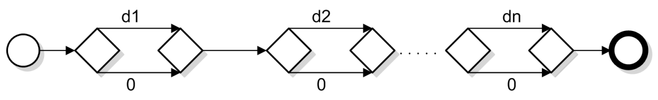

Fig. 8: A chain of XOR blocks

Suppose we can answer in polynomial time to $\mathbb{P}(c(\sigma) \leq \alpha) \leq p$. We can then also compute $\mathbb{P}(c(\sigma) \leq \alpha)$ in polynomial time. Please note that for the class of graphs of Fig. 8, $\mathbb{P}(c(\sigma) \leq \alpha) = \frac{k}{2^n}$, for some k where $0 < k < 2^n$ and n is the number of XOR gateways. One way to compute $\mathbb{P}(c(\sigma) \leq \alpha)$ is to run binary search with queries for $\mathbb{P}(c(\sigma) \leq \alpha) \leq p$ with varying p. Binary search takes $log(2^n) = n$ operations.

Because we now know $\mathbb{P}(c(\sigma) \leq \alpha)$, we can also answer in polynomial time if there exists an execution σ such that $c(\sigma) = \alpha$: We have $\mathbb{P}(c(\sigma) \leq \alpha - 1) \neq \mathbb{P}(c(\sigma) \leq \alpha)$, if and only if there exists σ such that $c(\sigma) = \alpha$. This in turn is the case exactly when there is a subset of $\{d_1, d_2, \ldots d_n\}$ which sums to α.□

The subset sum problem can be solved in pseudo-polynomial time, i.e., in polynomial time if numbers are represented in unary form. One way to represent the durations in unary form in a workflow graph is to assume that each edge needs one time unit and represent a duration of k time units by a sequence of k edges. For such a model, it is known for the case of sequential graphs, i.e., for Markov chains, that the probability of deadline transgression can be computed in polynomial time, e.g., by using the model checking algorithm of pCTL [12]. Therefore, the problem is said to be *weakly* NP-hard for sequential graphs. This can be extended to regular graphs, because each regular AND-block with subblocks X and Y can be treated as an sequence of X and Y under the assumption of a single resource. Therefore regular graphs can be reduced to sequential graphs.

4.2 Expected duration

In some use cases, it may be sufficient to compute the expected duration, which turns out to be easier than the probability of transgression. The main contribution of this section is a polynomial-time algorithm for computing the expected duration for general sound WFGs. Subsequently we discuss some subclasses which have a linear-time solution.

General sound WFGs. For probabilistic sequential graphs, i.e., Markov chains, it is known that computing the expected duration can be done in polynomial time. In this context, it is often phrased as computing the mean hitting h_{xy} time in a Markov chain, which is the expected time of a random walk starting at node x to reach node y. The mean hitting times are the minimal non-negative solution to a set of n linear equations, as in [13], of which the computational cost is $O(n^3)$, cf. Cell B.4 of Table 1.

We can use a similar approach by identifying a suitable set of equations. Due to the linearity of the expectation, we can compute the expected duration of an execution as follows:

$$\mathbb{E}(c) = \mathbb{E}(\sum_{e \in E} w(e) \cdot X_e) = \sum_{e \in E} w(e) \cdot \mathbb{E}(X_e) \tag{5}$$

where the random variable $X_e(\sigma)$ is defined as the number of times an execution σ produces a token on e, defined in equation 2, which can be any non-negative integer in a cyclic WFG. To compute $\mathbb{E}(X_e)$, we can define a set of equations. For each AND-gateway v, we have $\mathbb{E}(X_e) = \mathbb{E}(X_{e'})$ for each $e, e' \in {}^\bullet v \cup v^\bullet$. For each XOR-gateway v and each $o \in v^\bullet$, we have

$$\mathbb{E}(X_o) = \mu(o) \cdot \sum_{e \in {}^\bullet v} \mathbb{E}(X_e) \tag{6}$$

In addition, we know $\mathbb{E}(X_{e_0}) = 1$ for the source edge e_0. We can now solve this system of linear equations in time $O(|E|^3)$ and we use equation 5 to compute the final result.

To sum up, we obtained the following result for Cell E.4 of Table 1:

Theorem 5 *The expected duration of a sound workflow graph can be computed in time $O(|E|^3)$.*

As an example, we consider the cyclic WFG in Fig. 9.

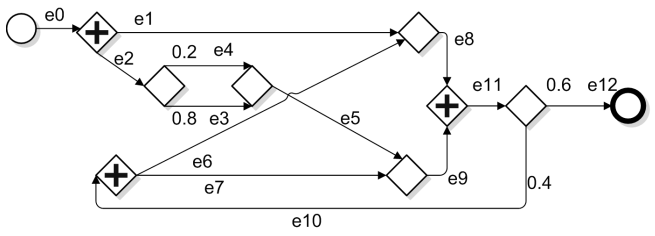

Fig. 9: Expected duration in a cyclic graph

For the example from Fig. 9, we obtain the following set of linear equations, where a variable e stands for $\mathbb{E}(X_e)$: $e_0 = e_1 = e_2 = 1$; $e_4 = 0.2 \cdot e_2$; $e_3 = 0.8 \cdot e_2$; $e_5 = e_3 + e_4$; $e_6 = e_7 = e_{10}$; $e_9 = e_5 + e_7$; $e_9 = e_8$; $e_{11} = e_9 = e_8$; $e_{12} = 0.6 \cdot e_{11}$; $e_{10} = 0.4 \cdot e_{11}$. For this example, assuming for simplicity, that all edges have duration 1, we obtain an expected duration of 12.95.

Regular graphs with cycles. For regular WFGs, the expected duration can be computed in linear time (Cell C.4 of Table 1) recursively, by exploiting the linearity of the expectation, as follows:

- Sequential and concurrent composition: The expected duration of $(X \; ; \; Y)$ and X AND Y is the sum of the expected durations of X and Y.
- Alternative composition: The expected duration of X XOR Y is $p_X \cdot d_X + p_Y \cdot d_Y$, where p_X ($p_Y = 1 - p_X$) is the probability of branching into subgraph X (Y resp.) and d_X is the expected duration of X.

– Loops: The expected duration of X LOOP Y, where p is the probability of re-entering the loop and $1-p$ is the probability of exiting, can be computed by solving the system of two linear equations, which yields the following closed formula:

$$\sum_{k=0}^{\infty}(1-p)^k((k+1)d_X + d_Y)$$

Sequential WFGs. It is known for acyclic sequential graphs that the expected duration can be computed in linear time [14]. We approach the problem for acyclic sound WFG in a similar way. We compute the expected number of times each edge is taken iteratively which is possible by processing the edges in the partial order defined by the flow of the graph. Having computed the expected frequencies for each edge of the graph, the expected duration is just the inner product of the expected frequencies and the durations of the edges.

5 Conclusion

We presented new results on the deadline analysis of workflow graphs. Since workflow graphs correspond to Free-Choice Petri nets, their expressiveness is strictly between control-flow of sequential programs (state machines) and control-flow of general concurrent processes (Petri nets). While they do allow concurrency as well as choice, the overlap of concurrency and choice is restricted in a way that no race conditions can arise. While this restriction limits their application, many applications are known, in particular in business process modeling. For example, the set of 735 of industrial process models used in [5] could be mapped completely to WFGs.

We have shown that where efficient algorithms for deadline analysis of sequential programs exist, we were able to define efficient algorithms for the corresponding WFG classes exploiting the linear-algebraic properties of WFGs.

In future work, we would like to address the case of WFGs executed by more than one resource and investigate whether the problem designated by Cell D.3 is *weakly* NP-hard (as it is the case for Cell A.3).

References

1. Richard Bellman. On a Routing Problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.
2. E. Best. Structure theory of petri nets: The free choice hiatus. In *Advances in Petri Nets 1986, Part I on Petri Nets: Central Models and Their Properties*, pages 168–205, London, UK, UK, 1987. Springer-Verlag.
3. Haiyan C. and Fuji Z. The expected hitting times for finite markov chains. *Linear Algebra and its Applications*, 428(1112):2730 – 2749, 2008.
4. J Desel, J.and Esparza. *Free Choice Petri Nets*. Cambridge University Press, New York, NY, USA, 1995.
5. D.Fahland, C. Favre, J. Koehler, N. Lohmann, H. Völzer, and K. Wolf. Analysis on demand: Instantaneous soundness checking of industrial business process models. *Data Knowl. Eng.*, 70(5):448–466, 2011.

6. E. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1:269–271, 1959.
7. C. Favre, D. Fahland, and H. Voelzer. The relationship between workflow graphs and free-choice workflow nets. *Information Systems*, 2013.
8. Bruno Gaujal, Stefan Haar, and Jean Mairesse. Blocking a transition in a free choice net and what it tells about its throughput. *Journal of Computer and System Sciences*, 66(3):515 – 548, 2003.
9. H. Hansson and B. Jonsson. A framework for reasoning about time and reliability. In *Real Time Systems Symposium, 1989., Proceedings.*, pages 102–111, Dec 1989.
10. J.C. Kleinsorge, H. Falk, and P. Marwedel. Simple analysis of partial worst-case execution paths on general control flow graphs. In *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*, pages 1–10, Sept 2013.
11. M. Kwiatkowska, g. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification*, CAV'11, pages 585–591, Berlin, Heidelberg, 2011. Springer-Verlag.
12. M. Kwiatkowska and D. Parker. Advances in probabilistic model checking. In T. Nipkow, O. Grumberg, and B. Hauptmann, editors, *Software Safety and Security - Tools for Analysis and Verification*, volume 33 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 126–151. IOS Press, 2012.
13. D. Levin, Y. Peres, and E. Wilmer. *Markov chains and mixing times*. American Mathematical Society, 2006.
14. David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov chains and mixing times*. American Mathematical Society, 2006.
15. Hafedh Mili, Guy Tremblay, Guitta Bou Jaoude, Éric Lefebvre, Lamia Elabed, and Ghizlane El Boussaidi. Business process modeling languages: Sorting through the alphabet soup. *ACM Comput. Surv.*, 43(1):4:1–4:56, December 2010.
16. J. Vanhatalo, H. Voelzer, and J. Koehler. The refined process structure tree. In M. Dumas, M. Reichert, and M. Shan, editors, *Business Process Management*, volume 5240 of *Lecture Notes in Computer Science*, pages 100–115. Springer Berlin Heidelberg, 2008.
17. D. Varacca, H. Völzer, and Glynn Winskel. Probabilistic event structures and domains. *Theor. Comput. Sci.*, 358(2-3):173–199, 2006.
18. H. Völzer. Randomized non-sequential processes. In *CONCUR 2001 - Concurrency Theory, 12th International Conference, Aalborg, Denmark, August 20-25, 2001, Proceedings*, pages 184–201, 2001.
19. M. Wan and G. Ciardo. Symbolic reachability analysis of integer timed petri nets. In *SOFSEM 2009: Theory and Practice of Computer Science*, volume 5404 of *Lecture Notes in Computer Science*, pages 595–608. Springer Berlin Heidelberg, 2009.
20. U. Zwick. Exact and approximate distances in graphs a survey. In *Algorithms ESA 2001*, volume 2161 of *Lecture Notes in Computer Science*, pages 33–48. Springer Berlin Heidelberg, 2001.

Appendix (at discretion of the reviewer, not in Proceedings version)

Preliminaries for Lemma 1

Let m and m' be two markings of Γ. A tuple (E_1, v, E_2) is called a *transition* if $v \in V$, $E_1 \subseteq {}^{\bullet}v$, and $E_2 \subseteq v^{\bullet}$. A transition (E_1, v, E_2) is *enabled* in a marking m if for each edge $e \in E_1$ we have $m(e) > 0$ and any of the following propositions:

- $l(v) = \text{AND}$, $E_1 = {}^{\bullet}v$, and $E2 = v^{\bullet}$.
- $l(v) = \text{XOR}$, there exists an edge $e \in E$ such that $E_1 = \{e\}$, and there exists an edge $e' \in E$ such that $E_2 = \{e\}$.

A transition T can be executed in a marking m if T is enabled in m. An *execution sequence* of Γ is an alternate sequence $\sigma =< m_0, T_0, m_1, T_1 \cdots >$ of markings m_i of Γ and transitions $T_i = (E_i, v_i, E'_i)$ such that, for each $i \geq 0$, T_i is enabled in m_i and m_{i+1} results from the execution of T_i in m_i.

The *Parikh vector* of an execution $\sigma =< m_0, T_0, m_1, T_1 \cdots >$, written $\vec{\sigma}$ maps every transition T to the number of occurrences of T in σ. More formally, it is the multi-set of edges such that $\vec{\sigma}\ (e_{source}) = 1$ and $\forall e \in E \setminus \{e_{source}\}$ it holds $\vec{\sigma}\ (e) = k$ such that $k = |\{i \mid T_i = (E, v, E') \wedge e \in E'\}|$

The *cluster* $[v]$ of a node $v \in V$ is a subset of $E \cup V$ such that $[v] = \{v\} \cup {}^{\bullet}v \cup v^{\bullet}$.

A permutation of a finite execution sequence, is a finite execution sequence with the same transitions as the original sequence, and the same initial marking as the original. Since the two execution sequences have the same Parikh vector, it follows from the marking equation lemma in [4], that the final marking of the permutation of the execution sequence is the same as that of the original.

Proof of Lemma 1

Let σ^* be the execution sequence of Γ of minimum cost. If σ^* is not loop-free, then $\sigma^* =< m_0, T_0, \cdots, m_i, T_i, \cdots, m_j, T_j, \cdots, m_n, T_n >$ such that $T_i = (E_1, v, E_2)$ and $T_j = (E_1, v, E_2)$ (there exists a node v such that a transition in $[v]$ is executed more than once).

Let σ' be a permutation of σ^*, $\sigma' =< m_0, T_x, \cdots, m'_i, T_i, \cdots, m'_j, T_j, \cdots, m_n, T_y >$. We construct σ' from σ^* in the following way. When we reach the marking m_i such that the transition $T_i \in [v]$ is enabled, we first execute all the other transitions from σ^* that are enabled until the only enabled transitions belong to $[v]$. We then obtain a marking m'_i, and we can execute transition T_i. Next, we proceed similarly for when T_j is enabled, and we obtain a marking m'_j.

It holds that $m'_i = m'_j$ due to Lemma 5, stated by the authors in [8]. Since $m'_i = m'_j$, σ' and implicitly $\sigma*$ can not be the executions of minimum cost, as one can construct a lower cost execution sequence by removing the execution sequence in σ' that led to the repetition of the marking.

Lemma 5 *If Γ is a sound workflow graph and v a node in Γ, then there exists a unique reachable marking m_v such that the only enabled transitions are the set of transitions in $[v]$.*

In the original paper, Lemma 5 is phrased in the context of free choice Petri Nets, but due to the equivalenece between workflow graphs and free choice Petri Nets stated in [7], the result holds for workflow graphs as well.□

Proof of Lemma 2

We prove the lemma by induction on k, the number of calls of $Relax(e, v)$.
Base case: $P(0) : \delta(e_{sink}) = w(e_{sink})$, therefore clearly, $\delta(e_{sink}) = d^*(e_{sink})$, and for all $e \in E \setminus \{e_{sink}\}$ $\delta(e) = \infty$, and therefore $\delta(e) > d^*(e)$.
Induction step: Suppose $P(k)$ is true, and thus after the k-th call of $Relax(e, v)$ we have $\delta(e) \geq d^*(e)$ for all e. At the $k + 1$-th call of $Relax(e, v)$, only $\delta(e)$ may get updated, while all the other $\delta(e'), e' \in E \setminus \{e\}$ remain unchanged.

We will show that from the definition of $d^*(e)$ and from the induction hypothesis, it follows that $\delta(e) \geq d^*(e)$, for each of the relaxation cases. We will present the reasoning for one of the cases, as the justification for the remaining ones is analogous.

Let v be a node, with $l(v) =$ XOR and $|v^\bullet| > 1$. Let $\{e\} = {}^\bullet v$ and $\{e_1, e_2, \cdots, e_m\} = v^\bullet$. Before the $k + 1$-th relaxation step, it holds that $\delta(e_i) \geq d^*(e_i)\ \forall i, 1 \leq i \leq m$. After the $k + 1$-th relaxation step, $\delta(e)$ gets updated, such that, $\delta(e)$ becomes $w(e) + min_{e_i}(\delta(e_i))$, $1 \leq i \leq m$. Therefore, due to the induction hypothesis, $\delta(e) \geq w(e)+min_{e_i}(d^*(e_i))\ \forall i, 1 \leq i \leq m$ (i). From the definition of $d^*(e)$, it holds that $d^*(e) = w(e)+min_{e_i} d^*(e_i), 1 \leq i \leq m$ (ii). From (i) and (ii) it follows that $\delta(e) \geq d^*(e)$.□

Note that at each relaxation step we can only decrease the value of $\delta(e)$. Once $\delta(e) = d^*(e)$, it doesn't change (it can not decrease further) as otherwise it would contradict the claim that $\delta(e) \geq d^*(e)$.

Proof of Lemma 3

We prove the lemma by induction on the number of edges in the set, k.
Base case: $P(0) : \delta(e_{sink}) = w(e_{sink})$, therefore $\delta(e_{sink}) = d^*(e_{sink})$.
Induction step: Suppose $P(k)$ is true, and thus after having relaxed $e_{sink}, \cdots, e_{k-2}, e_{k-1}$ in order, we have accordingly that $\delta(e_{sink}) = d^*(e_{sink}), \cdots, \delta(e_{k-2}) = d^*(e_{k-2}), \delta(e_{k-1}) = d^*(e_{k-1})$. We will show that when we relax the edge e_k, given the definition of $d^*(e)$ and the induction hypothesis , it follows that $\delta(e_k) = d^*(e_k)$. We present the reasoning for one of the cases, as the justification for the remaining ones is analogous.

Let $\{e\} = {}^\bullet v$ where v is a node, with $l(v) =$ XOR and $|v^\bullet| > 1$. As above, assume $< e_{k-1} \cdots, e_{sink} >$ are the edges that get marked after e gets marked, in an execution $\sigma \in LF$ for which $\sigma(e) = 1$ and $d_\sigma(e) = d^*(e)$. Assume without loss of generality that $e_{k-1} \in v^\bullet$.

From the induction hypothesis we have $\delta(e_{sink}) = d^*(e_{sink}), \cdots, \delta(e_{k-2}) = d^*(e_{k-2})$, $\delta(e_{k-1}) = d^*(e_{k-1})$ (i). From the relaxation procedure, when e is relaxed, $\delta(e) = w(e) + min_{e'}(\delta(e'))$ where $e' \in v^\bullet$. Since e_{k-1} is the edge that gets marked after e gets marked in σ for which $d_\sigma(e) = d^*(e)$, we have that $e_{k-1} = argmin_{e'}(\delta(e'))$ (ii). Therefore, from (i) and (ii), we have that $\delta(e) = w(e) + d^*(e_{k-1})$, and therefore $\delta(e) = d^*(e)$.□