Efficient Edge Filtering of Directly-Follows Graphs for Process Mining

David Chapela-Campa[a,*], Marlon Dumas[b], Manuel Mucientes[a], Manuel Lama[a]

[a]*Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS)*
Universidade de Santiago de Compostela, Santiago de Compostela, Spain
[b]*University of Tartu, Tartu, Estonia*

Abstract

Automated process discovery is a process mining operation that takes as input an event log of a business process and generates a diagrammatic representation of the process. In this setting, a common diagrammatic representation generated by commercial tools is the directly-follows graph (DFG). In some real-life scenarios, the DFG of an event log contains hundreds of edges, hindering its understandability. To overcome this shortcoming, process mining tools generally offer the possibility of filtering the edges in the DFG. We study the problem of efficiently filtering the DFG extracted from an event log while retaining the most frequent relations. We formalize this problem as an optimization problem, specifically, the problem of finding a sound spanning subgraph of a DFG with a minimal number of edges and a maximal sum of edge frequencies. We show that this problem is an instance of an NP-hard problem and outline several polynomial-time heuristics to compute approximate solutions. Finally, we report on an evaluation of the efficiency and optimality of the proposed heuristics using 13 real-life event logs.

Keywords: Process mining, Automated process discovery, Directly-follows graph, Edge filtering

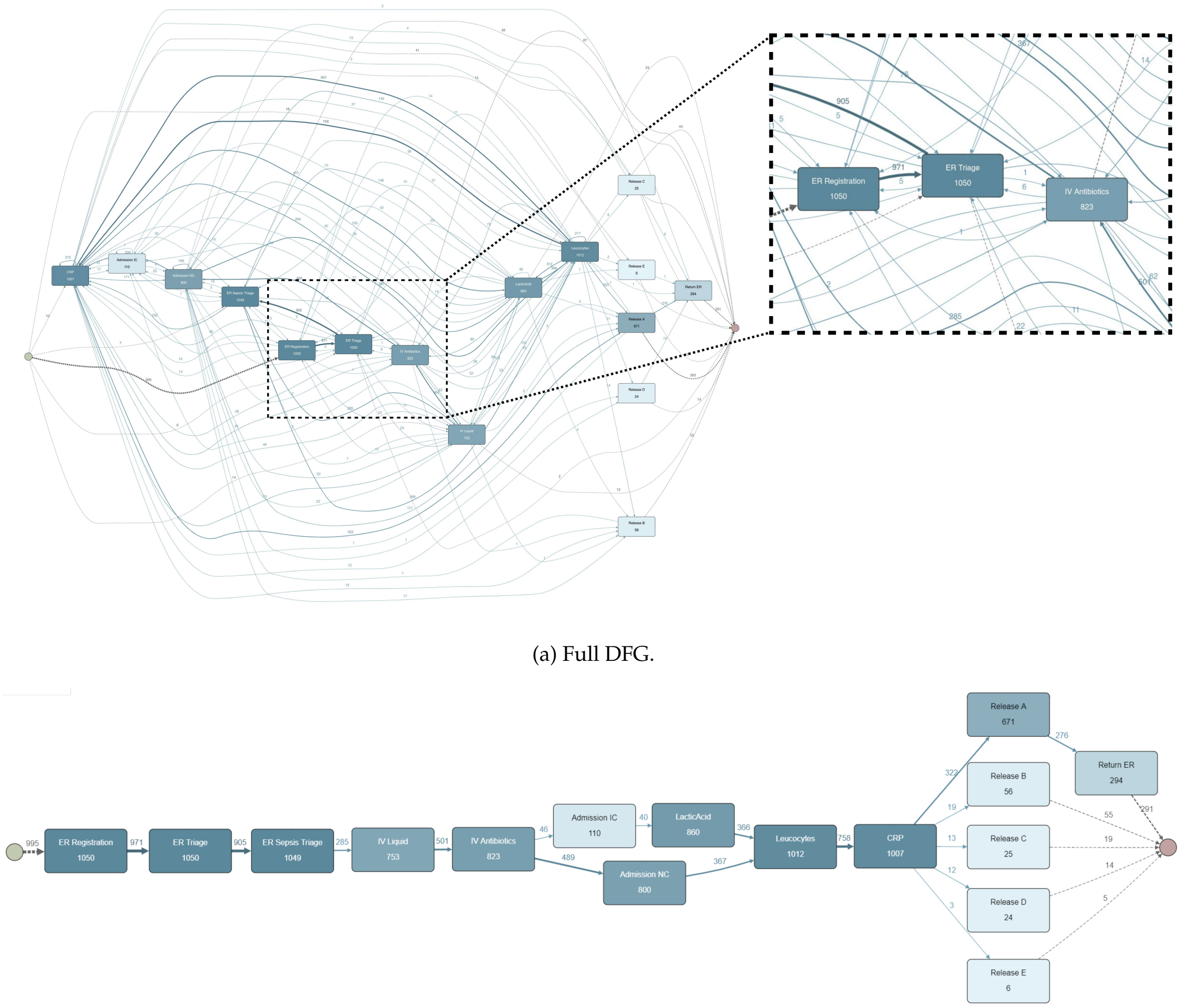
1. Introduction

Process mining (PM) is a family of techniques to discover, monitor, and improve processes based on information extracted from event logs recording the sequences of activities executed in a business process [27]. One of the main operations in the field of PM is *automated process discovery*. The goal of automated process discovery is to generate a diagrammatic representation of the process recorded in an event log. This diagrammatic representation should be as understandable as possible, since it is used by managers and analysts for exploratory analysis. At the same time, it should capture as much of the behavior observed in the event log as possible.

Existing automated process discovery techniques produce various diagrammatic representations as outputs, including process models in the Business Process Model Notation (BPMN), Petri nets, Process trees,

[*] Corresponding author

Email addresses: `david.chapela@usc.es` (David Chapela-Campa), `marlon.dumas@ut.ee` (Marlon Dumas), `manuel.mucientes@usc.es` (Manuel Mucientes), `manuel.lama@usc.es` (Manuel Lama)



(a) Full DFG.

(b) Filtered DFG computed by Apromore (v7.19).

Figure 1: Full DFG and filtered DFG of a patient treatment process in a Dutch hospital [19].

and Directly-Follows Graphs (DFG). The latter (DFGs) are commonly used in commercial PM tools [18]. A DFG is a directed graph wherein each vertex denotes an activity of the process, and each edge denotes the fact that the target activity occurs immediately after the source activity in at least one trace of the process (a directly-follows relation). In addition to being widely used in commercial PM tools as a diagrammatic representation in its own right, DFGs are also used as an intermediate artifact by several algorithms for discovering BPMN models or Petri nets [4, 17].

In real-life processes, the DFG of an event log may contain hundreds or thousands of edges, which hinders their understandability. For example, Figure 1a shows the DFG of a process recording the trajectories of patients treated for Sepsis disease in a Dutch hospital [19]. To tame this complexity, PM tools generally offer the possibility to simplify the full DFG by filtering out the most infrequent directly-follows relations, thus keeping only the most frequent ones. For example, Figure 1b shows the filtered version of the previous DFG produced by a PM tool, namely Apromore.[1]

There are two desirable properties that should be preserved when filtering a DFG. One is that the filtered DFG must contain all the vertices in the original (full) DFG. In other words, a filtered DFG must be a *spanning subgraph* of the full DFG. The importance of this property has been highlighted by van der Aalst [26] who notes that when vertices are removed during DFG filtering, some edges in the filtered DFG no longer have the semantics of a directly-follows relation, as some intermediate vertices are not shown. This may lead users to draw incorrect conclusions about the process.[2]

The second property is that every vertex in the filtered DFG must be on a path from the vertex representing the start event to the vertex representing the end event of the process. This property is called *DFG soundness* [18] and is required in order to generate BPMN models (or workflow nets) from the DFG [4]. Intuitively, this property is needed because otherwise, the DFG represents a process in which some of the activities (vertices) are unreachable from the start event, or cannot reach the end event of the process.

In this setting, this paper studies the following overarching research question: *Given the full DFG of an event log, how to efficiently compute a spanning and sound filtered DFG with a minimal number of edges, while retaining the most frequent directly-follows relations of the original DFG?* This question establishes a set of criteria to be met by the filtering approaches. The DFG filtering operation must:

C1. Produce a **sound** filtered DFG.

C2. Produce a **spanning** filtered DFG.

C3. Seek to **minimize the number of edges** of the filtered DFG, while meeting criteria C1 and C2.

C4. Seek to **maximize the sum of edges frequencies** in the filtered DFG, while meeting criterion C3.

Note that by addressing this question, we also address the problem of computing (sound and spanning) DFGs with arbitrary levels of filtering. Indeed, if we can find a filtered DFG with a minimal number of edges, we can obtain larger (less filtered) DFGs by simply adding back those edges that were left out from the minimal DFG, starting from the most frequent to the least frequent edges.

Given the above research question, the contributions of this paper are:

[1] `https://apromore.org/`

[2] Note that the techniques proposed in this paper can be applied to event logs where some activities have been removed during a pre-processing step. In this case, the techniques will build a filtered DFG that contains all the retained activities. A directly-follows relation should then be interpreted with the meaning that activity B “directly-follows” activity A modulo any removed activities. This approach may be useful if the removed activities are considered irrelevant by the user.

1. We formalize the problem of DFG filtering as an optimization problem, namely that of computing a sound (C1) and spanning (C2) subgraph of a DFG with a minimal number of edges (C3) and a maximal sum of edge frequencies (C4).
2. We show that this problem is an instance of an NP-hard problem.
3. Accordingly, we propose a set of polynomial-time heuristic approximations to solve it.
4. We evaluate the proposed heuristics in terms of their efficiency (execution time) and their optimality, i.e. their ability to minimize the number of edges and maximize the sum of edge frequencies.

The remainder of the article is structured as follows. Section 2 gives an overview of existing DFG filtering techniques. Section 3 introduces basic notions of graph theory and process mining. Section 4 formalizes the problem of computing filtered DFGs, while Section 5 presents heuristic approximations of this problem. Finally, Section 6 describes the empirical evaluation, and Section 7 draws conclusions and sketches future work directions.

2. Related work

The problem of DFG filtering has been previously studied by Leemans et al. [18], who outline an approach for DFG filtering in three steps. First, the most infrequent directly-follows relations in the full DFG are identified. Second, all traces that contain at least one occurrence of any of these infrequent relations are removed from the event log. Finally, the resulting filtered event log is used to compute a filtered DFG. By construction, the filtered DFG is sound. However, it is not a spanning DFG (not fulfilling C2). For example, if a given activity A has a directly-follows relation with say 20 activities $B_1 \ldots B_{20}$, and every one of these relations is infrequent, then all traces where A appears will be removed from the log, and hence A will not appear in the filtered DFG.

Conforti et al. [8] propose another approach in four steps. First, the most infrequent edges of the full DFG are identified and removed from the DFG. This filtered DFG is not sound —some vertices might not be reachable from the start vertex or cannot reach the end vertex of the DFG. In a second step, vertices that are not reachable from the start or that cannot reach the end are removed in order to make the filtered DFG sound.[3] Next, each trace in the event log is replayed against the filtered DFG. Because some traces cannot be perfectly replayed in the filtered DFG, some of the events in a given trace might be removed during the replay. In other words, every trace in the original log is retained in the filtered log, but some events are removed. Finally, the resulting filtered log can be used to generate a filtered DFG. This filtered DFG is

[3] The user may add a constraint stating that some of the vertices in the DFG are required and should not be removed from the filtered DFG.

sound by construction, but it is not spanning (not fulfilling C2). Indeed, some vertices are removed when the unsound filtered DFG is repaired. In addition, as a result of the replay-filtering step, events that cannot be replayed are removed and, hence, some activities —specifically those that participate only in infrequent relations— will not appear in the filtered log and in the corresponding filtered DFG.

As explained in [26], removing activities during DFG filtering changes the semantics of the DFG insofar as some edges in the filtered DFG no longer represent directly-follows relations. Furthermore, filtering out traces in the event log in order to generate a filtered DFG alters the frequency of the directly-follows relations —in other words, the frequencies of the directly-follows relations in the filtered DFG are not necessarily the same as in the original DFG. In this paper, we study the problem of computing filtered DFGs that are both sound and spanning, and such that the frequencies of the edges in the filtered DFG coincide with those in the original DFG. In other words, we seek to filter edges in the DFG without filtering any vertex.

The problem of DFG edge filtering has also been addressed in the context of automated process discovery algorithms. For instance, the Heuristics Miner [37] starts by computing the DFG and applies heuristics to assign a confidence measure to each edge of the DFG. This confidence measure captures the degree of certainty that there is a true (sequential) dependency between two activities, as opposed to an interleaved concurrency relation [4]. Next, for each vertex, the algorithm retains the incoming and outgoing edges with the highest weight. Depending on some user-defined thresholds, some additional edges associated with so-called short cycles may be retained as well. The goal of this filtering technique is to filter out directly-follows relations that correspond to (interleaved) concurrency. It does not seek to minimize the number of edges of the filtered DFG (i.e. does not fulfill C3) or to filter out infrequent relations from the DFG as we do in this paper (does not fulfill C4). Furthermore, this technique may lead to unsound DFGs (does not fulfill C1).

The Inductive Miner also incorporates a DFG filtering technique based on edge frequency [17]. This technique starts by computing an eventually follows graph —the transitive closure of the directly-follows relation [24]— and normalizing each edge frequency by dividing it by the frequency of the strongest outgoing edge of its source vertex. Edges with a normalized frequency under a defined threshold are then removed. This filtering approach may lead to filtered DFGs that are unsound (does not fulfill C1). As a post-processing step, this unsound DFG may be turned into a sound DFG by removing unreachable vertices, but then, the resulting filtered DFG is not a spanning DFG (does not fulfill C2).

In a similar vein, the Split Miner [4] includes a DFG filtering technique based on a variation of Dijkstra's shortest path algorithm. Augusto et al. filter a DFG by retaining for each vertex v the incoming edge that is part of the path $source \rightarrow v$ with maximum capacity and the outgoing edge being part of the path

[4] In the heuristics miner, a concurrency relation is asserted between two activities A and B, if these two activities are executed in any order, i.e. sometimes A follows B and other times B follows A. We note that other notions of concurrency have been proposed in the field of process mining. An in-depth treatment of concurrency notions in process mining is provided in [2].

Table 1: DFG filtering techniques and their fulfillment of the set of criteria introduced in Section 1. '✓' and '✗' denote that the filtering technique does or does not ensure to fulfill the criterion, respectively. In the case of criterion 4, '±' denotes the filtering techniques that partially fulfill the criterion —they do not focus on maximizing the sum of edges frequency, but they remove the edges regarding their frequency.

Filtering approach	Desired search criteria			
	C1	C2	C3	C4
Leemans et al. [18]	✓	✗	±	±
Conforti et al. [8]	✓	✗	±	±
Weijters et al. Heuristics Miner [37]	✗	✓	±	✗
Leemans et al. Inductive Miner [17]	✗	✓	±	±
Augusto et al. Split Miner [4]	✓	✓	±	±

$v \rightarrow$ *sink* with maximum capacity. In this context, the capacity of a path is the frequency of the least frequent edge in this path. This technique computes sound and spanning DFGs, but it does not directly attempt to minimize the number of edges in the filtered DFG (does not fulfill C3), nor to maximize the sum of the edge frequencies (does not fulfill C4), as we will further discuss in the following sections.

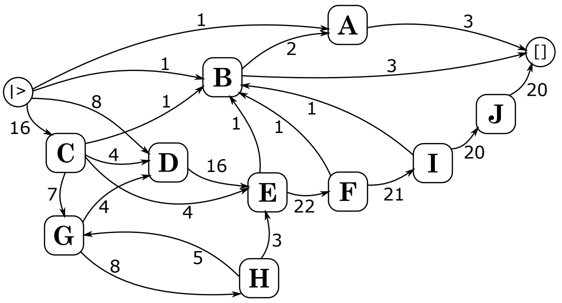
Table 1 shows the desired criteria for the DFG filtering problem introduced in Section 1, and which of the discussed techniques fulfill each of them. The only DFG filtering approach that produces a sound (C1) and spanning (C2) filtered DFG is the Split Miner DFG filtering. None of the approaches directly seeks to minimize the number of edges (C3) or to maximize the frequency of the retained edges (C4). Regarding criterion C3, existing techniques offer a threshold to control the number or the percentage of edges to be removed, but they cannot be used to maximally filter the DFG. In the case of criterion C4, some techniques (indirectly) consider the frequency of the edges during removal, but they do not always remove edges in such a way as to retain the most frequent ones, while the Heuristics Miner DFG filtering removes edges without taking into account their frequency.

3. Preliminaries

In this section, we introduce some basic notions of process mining and graph theory related to the problem that is being addressed. We consider a business process that involves a set of activities A. We denote each of these activities with α. An *event* ε denotes one execution of an activity. We write $\triangledown(\varepsilon)$ to denote the activity associated with an event ε. Usually, an event carries additional information besides an activity label, such as one or more timestamp(s), the resource who performed the activity, etc. However, in this paper, we focus on constructing DFGs where each vertex represents an activity and, hence, we do not consider such additional attributes. A *trace* $\tau = \langle \varepsilon_1, ..., \varepsilon_n \rangle$ is a list (sequence) of events, such that

Trace		Trace	
`A`	(×1)	`B A`	(×1)
`C B A`	(×1)	`D E B`	(×1)
`D E F B`	(×1)	`D E F I J`	(×5)
`D E F I B`	(×1)	`C E F I J`	(×4)
`C D E F I J`	(×4)	`C G D E F I J`	(×2)
`C G H G H E F I J`	(×3)	`C G H G D E F I J`	(×2)

(a) Event log example.



(b) Full DFG of the event log in Figure 2a.

Figure 2: Example of an event log and its corresponding full DFG.

event ε_i was executed before ε_j for every $1 \leq i < j \leq n$. A trace represents one execution of the process. We use $\triangledown(\tau, i)$ to retrieve from trace τ the activity α executed in the event ε_i. An *event log* is defined as a collection (multi-set) of traces $L = \{\!\{\tau_1, ..., \tau_m\}\!\}$ recording m executions of a process. Finally, we use the term *trace variant* to refer to each unique activity sequence $\langle\ \alpha_1, ..., \alpha_n\ \rangle$ of a log L, where the frequency of a trace variant is the number of traces $\tau \in L$ having the same activity sequence. Figure 2a shows an example of an event log composed of 26 traces grouped in 12 trace variants —each trace variant frequency is depicted in parentheses.

From an event log, we can construct a DFG capturing the consecutive (directly-follows) relations between the activities observed in the log.

Definition 1 (Directly-Follows Relation). Given two activities $\alpha_1, \alpha_2 \in A$, and an event log L, there is a *directly-follows relation* from α_1 to α_2 in L, denoted by $\alpha_1 >_L \alpha_2$, iff $\exists \varepsilon_i, \varepsilon_j \in \tau \mid j = i+1 \wedge \triangledown(\varepsilon_i) = \alpha_1 \wedge \triangledown(\varepsilon_j) = \alpha_2$. The frequency, or weight, of a directly-follows relation is the number of times it is present in the event log, and it is denoted by $|\alpha_1 >_L \alpha_2|$.

The DFG of an event log is a directed graph where each vertex represents an activity observed in the event log, and each edge represents a directly-follows relation. To make the start and the end of the process explicit, it is customary to include an explicit start vertex and an explicit end vertex in the DFG of an event log. Figure 2b depicts an example of the full DFG corresponding to the event log in Figure 2a. As an example, edge (G, D) —with a weight of four— represents the directly-follows relation $G\ D$ that appears in the trace variants $\langle\ C,\ G,\ D,\ E,\ F,\ I,\ J\ \rangle$ —grouping two traces— and $\langle\ C,\ G,\ H,\ G,\ D,\ E,\ F,\ I,\ J\ \rangle$ —grouping another two traces.

Definition 2 (Full Directly-Follows Graph (DFG) of an event log). Given an event log L recording the executions of a set of activities A, its (full) directly-follows graph is a (directed) graph $DFG = (V, E)$, where $V = A \bigcup \{source, sink\}$ is the set of vertices corresponding to each activity $\alpha \in A$ plus a start activity

—*source*— and an end activity —*sink*—, and $E = \{(u, v) \in V \times V \mid (|u >_L v| > 0) \vee (u = source \wedge v = \triangledown(\tau, 1) \wedge \tau \in L) \vee (v = sink \wedge u = \triangledown(\tau, n) \wedge \tau \in L)\}$ is the set of directed weighted edges, each edge representing a directly-follows relation observed in the event log. The weight of an edge is given by a function $\omega : E \rightarrow \mathbb{N}$, where $\omega((u, v)) = |u >_L v|$ iff $u, v \in A$, otherwise $\omega((start, v)) = |T| \mid \tau \in T \wedge \triangledown(\tau, 1) = v$ or $\omega((v, end)) = |T| \mid \tau \in T \wedge \triangledown(\tau, n) = v$. Furthermore, we define the *total weight* of a DFG as the sum of its edges weight $\Omega(DFG) = \sum_{e \in E} \omega(e)$.

Herein, whenever we use the term *DFG of an event log*, we refer to the full DFG of an event log as defined above. In this paper, we also consider subgraphs of the DFG of an event log, which we call filtered DFGs (*F-DFG*s for short).

Definition 3 (Filtered Directly-Follows Graph (*F-DFG* of an event log)). Given an event log L and its (full) directly-follows graph $DFG = (V, E)$, a filtered DFG of L is a graph $F\text{-}DFG = (V', E')$ such that $V' \subseteq V \wedge E' \subseteq E$.

Following Def. 3, an *F-DFG* can be unsound and not spanning. Nevertheless, as stated in Section 1, we are interested in obtaining sound filtered DFGs. To formally define the soundness property, we use the following notations. Given a vertex $v \in V$, $\bullet v = \{(v_1, v_2) \in E \mid v = v_2\}$ denotes the set of incoming edges of v while $v\bullet = \{(v_1, v_2) \in E \mid v = v_1\}$ denotes the set of outgoing edges. We note that $\bullet source = \emptyset$ and $sink\bullet = \emptyset$. A (directed) *path* from one vertex u to another vertex v, denoted by $u \rightarrow v$, is a sequence of edges $\langle$ (v_1, v_2), (v_2, v_3), ..., (v_{k-1}, v_k) $\rangle$ where $u = v_1$ and $v = v_k$.

Given the above, Leemans et al. [18] define DFG soundness as follows.

Definition 4 (Sound DFG). A $DFG = (V, E)$ is *sound* iff $\forall v \in V$ there is a path $source \rightarrow sink$ such that v is one of the vertices in that path.

As discussed in Leemans et al., the DFG of an event log is sound by construction. On the other hand, a filtered DFG may be unsound, as the paths from a given vertex to the end vertex may be broken once some edges are removed.

A second property we seek to ensure is that a filtered DFG should be spanning. We say that a (filtered) DFG is *spanning* if it contains all the activities observed in the event log (A). Trivially, the DFG of an event log is spanning, but a filtered DFG may or may not have this property.

In this paper, we are specifically interested in extracting minimal-sized spanning filtered DFGs. To this end, we will make use of the concept of *spanning arborescence*, which is the smallest spanning sub-graph that can be extracted from a given directed graph G.

Definition 5 (Spanning Arborescence). Given a directed graph $G = (V, E)$, an *arborescence* (a.k.a. a *branching*) of G is a tuple $B = (V', E')$ where $V' \subseteq V$ and $E' \subseteq E$, such that for two distinct edges $(u, v), (u', v') \in E'$, $v \neq v'$ and B contains no cycle, i.e. it contains no path $\langle$ (v_1, v_2), (v_2, v_3), ..., (v_{k-1}, v_k) $\rangle$ such that $v_1 = v_k$. An arborescence is *spanning* iff $V' = V$.

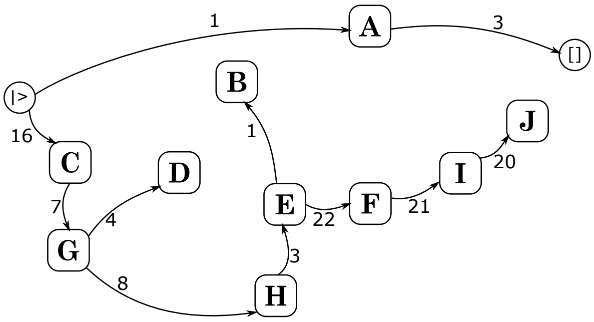


Figure 3: Example of a spanning arborescence of the DFG in Figure 2b with root in '|>'.

For example, Figure 3 depicts an example of a spanning arborescence of the DFG in Figure 2b, with root in the start vertex '|>'.

4. Problem formulation

As stated in Section 1, the goal of this paper is, given a $DFG = (V, E)$ with $|E| = n$, to efficiently compute a sound and spanning $F\text{-}DFG = (V, E')$ with minimum number of edges, and maximum total weight, denoted by *MWMF-DFG*.

Definition 6 (Maximum Weight Maximally Filtered Directly-Follows Graph (*MWMF-DFG*)). Given an event log L, and its directly-follows graph $DFG = (V, E)$, a maximum weight maximally filtered directly-follows graph is a sound filtered directly-follows graph $MWMF\text{-}DFG = (V, E') \mid E' \subset E$ such that $\forall F\text{-}DFG = (V, E'') \mid E'' \subset E\ [|E'| < |E''| \vee (|E'| = |E''| \wedge \Omega(MWMF\text{-}DFG) \geq \Omega(F\text{-}DFG))]$, i.e. the number of edges of *MWMF-DFG* is minimum, and its total weight is maximum among all other F-DFG of the same size.

The rationale for seeking a filtered DFG of minimal size is that, once we have such a DFG, we can construct filtered DFGs of any larger size (up to the size of the full DFG of the event log) by simply adding some of the edges that were filtered out. The rationale for maximizing the sum of the edge weights is to ensure that, overall, we retain the most frequent relations during the filtering.

Figure 4 depicts an example of two *MWMF-DFG*s of the DFG in Figure 2b. Note that a DFG may have multiple *MWMF-DFG*s. Both graphs contain 13 edges and have a total weight of 147. We can see that none of the edges can be removed without violating the soundness, and that there is no other sound and spanning *F-DFG* having fewer edges, or a higher total weight —for the same number of edges. Once an *MWMF-DFG* with $|E'| = k \mid k < n$ has been obtained, a set of sound and spanning $F\text{-}DFG = (V, E'')$ with $|E''| = m \mid k < m < n$, maximizing the total weight, can be easily computed by adding the edges from $E \setminus E'$ in weight descending order.

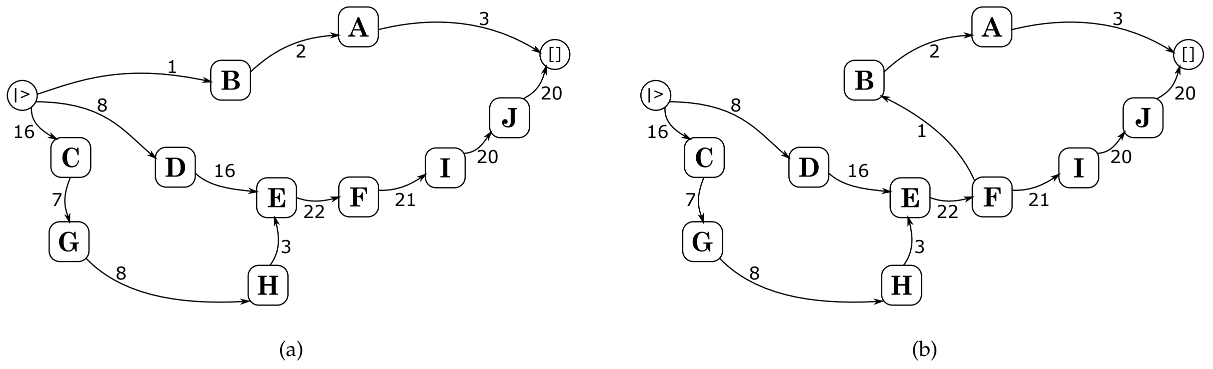


Figure 4: Example of two *MWMF-DFG* of the *DFG* in Figure 2b, formed by 13 edges and with a total weight of 147.

In order to analyze the complexity of the problem of finding an *MWMF-DFG*, we consider the unweighted version of this problem, namely that of finding a sound and spanning maximally filtered DFG (*SSMF-DFG*), where an *SSMF-DFG* is the *F-DFG* of a given DFG that is sound and spanning and has the minimum possible number of edges. Clearly, the complexity of computing an *MWMF-DFG* is the same or higher than that of computing an *SSMF-DFG*, since solving the former problem immediately gives us a solution to the latter one.

The *SSMF-DFG* problem is an instance of the Minimum Spanning Strong Sub(di)graph (*MSSS*) problem, which has been proved to be $\mathcal{NP}$-hard[5] [5, 16]. Given a strongly connected[6] graph $G = (V, E)$, the *MSSS* problem consists of finding the strongly connected spanning subgraph $MSSS = (V, E') \mid E' \subseteq E$, such as the number of edges $|E'|$ is minimum.

To map the *SSMF-DFG* problem into an instance of the *MSSS* problem, we make a $DFG = (V, E)$ strongly connected by adding the edge $(sink, source)$ to E—we call it the augmented graph of *DFG*. The $MSSS = (V, E')$ of this augmented graph has a path between every two vertices $u, v \in V$. Thus, it has a path from every vertex $v \in V$ to *source*. As $\bullet source = (sink, source)$, all these paths must contain *sink* and, thus, for all $v \in V$, there must be a path $v \rightarrow sink$. In the same way, there must exist a path $source \rightarrow v$ for all $v \in V$. Thus, by removing $(sink, source)$, we obtain a maximally filtered DFG.

This proves that, if we solve the *SSMF-DFG* problem, we would have solved the *MSSS* problem for a subclass of graphs —those having an edge (u, v) such that $|u \bullet| = 1$ and $|\bullet v| = 1$.

[5] The *MSSS* problem has been shown to be polynomial for certain restricted families of graphs: graphs with cycles of size no more than three edges, extended semicomplete graphs, and quasi-transitive graphs [5]. Nevertheless, none of these results apply to the problem presented in this paper, as the only restrictions in the graph are those implied in Definition 2. Furthermore, it has been proved that for any graph with a cycle of more than 17 edges, the problem is MAX $\mathcal{SNP} - hard$, implying that there cannot exist a polynomial-time approximation scheme for this problem, unless $\mathcal{P} = \mathcal{NP}$ [16].

[6] A graph $G = (V, E)$ is *strongly connected* if there is a path $u \rightarrow v$ for every two vertices $u, v \in V$.

5. Approach

In this section, we propose several polynomial-time heuristics to approximate the *MWMF-DFG* problem (cf. Section 4). As the proposed techniques seek to generate an *MWMF-DFG* but do not necessarily find it, we will refer to the outputs of these techniques as *MWMF-DFG* approximations, or simply *F-DFG*s.

5.1. Greedy approach

A naive approach to approximate the computation of an *MWMF-DFG*, is to remove edges from the full DFG, one after another, in ascending weight order, i.e. from least frequent to most frequent. To ensure that the removal of an edge does not lead to an unsound *F-DFG*, we propose to perform two breadth-first searches after each removal: one forward from *source* to *sink*, and another backward from *sink* to *source*. If all the vertices are reached in both searches, the result is sound and the filtering can continue with the next edge. Otherwise, the edge has to be put back before considering the removal of the next edge. Note that not all edges have to be considered for removal.

Performing two breadth-first searches for every edge is inefficient, and thus we should avoid it if possible. We note that the removal of an edge $(u, v) \in E$ such that $|u\bullet| = 1 \vee |\bullet v| = 1$ —i.e. (u, v) is the only outgoing edge of u, or the only incoming edge of v— will necessarily lead to u or v not having any incoming or outgoing edge, thus violating the soundness property. Accordingly, we will not consider such edges for removal.

The above observations lead to a straightforward *Greedy* algorithm for DFG filtering.[7]

```
Greedy:
  let $G = (V, E)$ be a sound DFG
  let $\omega$ be a weight function over $E$
  put in $E_f$ all the edges in $E$
  while $E_f$ has unvisited edges:
    remove unvisited $(u, v) \in E_f$ with min $\omega((u, v))$
    mark $(u, v)$ as visited
    if ($|u\bullet| > 1$ and $|\bullet v| > 1$):
      remove $(u, v)$ from $E_f$
      if (($V, E_f$) is not sound):
        add $(u, v)$ back to $E_f$
  return $(V, E_f)$
```

[7] This greedy approach is used in some commercial tools, including APROMORE.

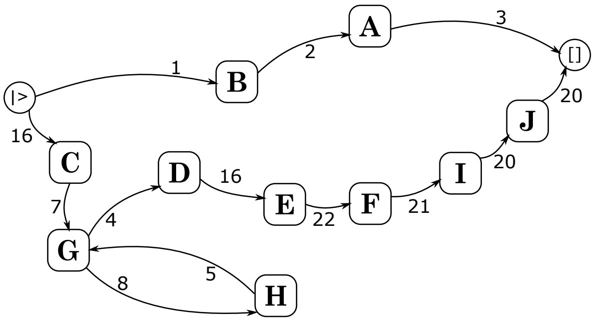


Figure 5: *MWMF-DFG* approximation formed by 13 edges and with a total weight of 145, computed by Greedy for the DFG in Figure 2b.

This technique computes sound and spanning *F-DFG*s. The spanning property is ensured because the algorithm does not remove any vertex. Soundness is ensured by checking that this property is maintained after each edge removal.

Nevertheless, this removal of edges may affect future removals and block the removal of other edges, which may lead to a non-optimal result in both dimensions (size and total weight). Figure 5 shows the *F-DFG* obtained by applying Greedy to the DFG in Figure 2b. The removal of (H, E) from the full DFG forces (G, D) and (H, G) to be kept in the *F-DFG* to ensure soundness, and makes $(|>, D)$ redundant. In this context, an edge (u, v) from a DFG is *redundant* when there is an alternative path $u \rightarrow v$ not including (u, v) in it. Conversely, Figure 4 shows that keeping (H, E) in the *F-DFG* and removing (G, D) allows us to obtain a higher total weight with the same number of edges.

Given that two breadth-first searches are being executed for each edge, the time complexity of the Greedy approach is $O(E(V + E))$, being E the number of edges and V the number of vertices in the DFG. Assuming the number of edges is greater than the number of vertices —otherwise, no filtering is needed— we can express the time complexity of Greedy as $O(E^2)$.

5.2. *Two Way Edmonds approach*

An *MWMF-DFG* is a spanning *F-DFG* with minimum number of edges and maximum total weight, where for all vertices $v \in V$, there must be a path *source* $\rightarrow v$ and another path $v \rightarrow$ *sink*. A spanning arborescence of a DFG is a tree rooted at a vertex r, where for all $v \in V$ there is a path $r \rightarrow v$. Chu and Liu [7] and Edmonds [11] have independently presented an approach to efficiently compute, given a graph, a spanning arborescence such as the sum of its edges weight is maximum (or minimum) — a.k.a., an optimum branching.

We propose a technique, henceforth referred to as *Two Ways Edmonds* (TWE), that consists of merging the maximum weight spanning arborescence with root in *source* ($B_{\rightarrow}$), and the reversed maximum weight spanning arborescence with root in *sink* ($B_{\leftarrow}$). [8]

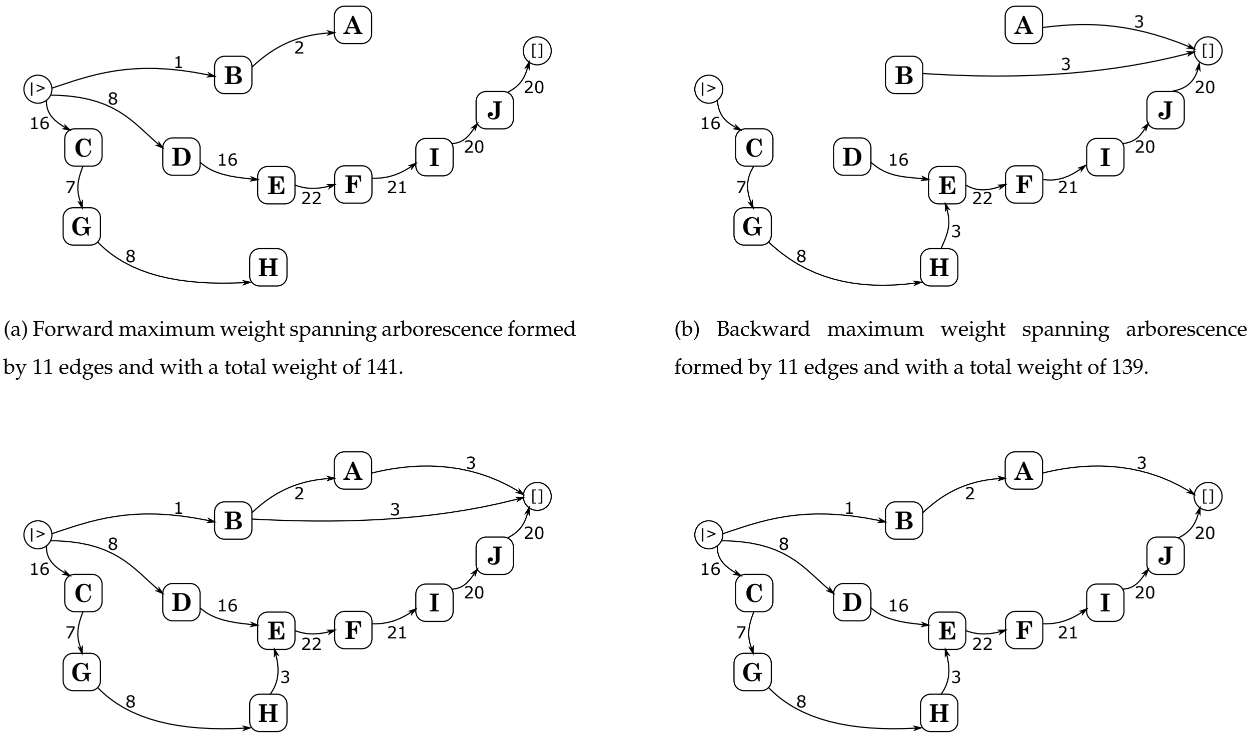
```
Two Ways Edmonds:
  let $G = (V, E)$ be a sound DFG
  let $\omega$ be a weight function over $E$
  let $E^{-1}$ be the reverse operation which changes the direction of each edge in $E$
  $B_{\rightarrow} = (V, E_{\rightarrow})$ # spanning arborescence of $G$ with root in $source$
  $B_{\leftarrow} = (V, E_{\leftarrow})$ # spanning arborescence of $G' = (V, E^{-1})$ with root in $sink$
  $E' = E_{\rightarrow} \bigcup (E_{\leftarrow})^{-1}$
  return $(V, E')$
```

This technique computes sound and spanning *F-DFG*s. The spanning property is ensured because each of the two arborescences (the forward $B_{\rightarrow}$ and the backward $B_{\leftarrow}$) are spanning, and thus their union contains all the vertices in the full DFG. Soundness is also ensured by construction. By combining $B_{\rightarrow}$ and $B_{\leftarrow}$ we ensure that for all $v \in V \setminus \{source, sink\}$ there is a path $source \rightarrow v$ —contained in $B_{\rightarrow}$— and a path $v \rightarrow sink$ —contained in $B_{\leftarrow}$.

Figure 6c depicts the *F-DFG* obtained by applying TWE to the DFG in Figure 2b. First, TWE computes the maximum weight spanning arborescence with root in *source* ($B_{\rightarrow}$, depicted in Figure 6a) by applying the Chu-Liu-Edmonds algorithm to the complete DFG. Then, TWE reverses the edges of the complete DFG, applies the Chu-Liu-Edmonds algorithm, and turns back the edges again to obtain the reversed maximum weight spanning arborescence with root in *sink* ($B_{\leftarrow}$, depicted in Figure 6b). As we can see, the result of this combination can contain redundant edges that can be removed to improve the filtering —e.g. $(B, [])$. Accordingly, we propose an alternative technique consisting of applying Greedy to the *F-DFG* computed by Two Ways Edmonds (TWE+G) to remove these redundant edges, whose result can be seen in Figure 6d.

The time complexity of TWE is given by the time complexity of Chu-Liu-Edmonds algorithm, which is $O(EV)$, being E the number of edges and V the number of vertices in the DFG. Nevertheless, more efficient algorithms have been proposed with time complexities of $O(Elog(V))$ by Tarjan [25] (after a correction made by Camerini et al. [6]), and of $O(E + Vlog(V))$ by Gabow et al. [13]. Regarding TWE+G, each arborescence has $V - 1$ edges, and thus, the Greedy addition runs over an *F-DFG* with a maximum of $2V - 2$ edges. This reduces both the number of breadth-first searches, and the possibility to remove an edge that causes to end in a local optimum. Hence, the time complexity of TWE+G is $O(EV + V^2)$.

[8] A similar approach has been proposed in [12] as a 2-approximation to the *MSSS* problem, where the arborescence and reversed arborescence having the same root are merged to obtain a spanning strongly connected subgraph.



(a) Forward maximum weight spanning arborescence formed by 11 edges and with a total weight of 141.

(b) Backward maximum weight spanning arborescence formed by 11 edges and with a total weight of 139.

(c) *MWMF-DFG* approximation formed by 14 edges and with a total weight of 150, computed by Two Ways Edmonds.

(d) *MWMF-DFG* approximation formed by 13 edges and with a total weight of 147, computed by Two Ways Edmonds + Greedy.

Figure 6: Maximum weight spanning arborescences and *MWMF-DFG* approximations computed by TWE and TWE+G for the DFG in Figure 2b.

5.3. *SplitMiner filtering approach*

Augusto et al. proposed in [4] a process discovery algorithm named Split Miner. One of the first steps of this algorithm is to filter a DFG seeking to minimize the number of edges while maximizing the total weight of the graph. This approach retains, for each vertex v, the incoming edge that is part of the path $source \rightarrow v$ with maximum capacity and the outgoing edge being part of the path $v \rightarrow sink$ with maximum capacity. In this context, the capacity of a path is the frequency of the least frequent edge in this path. Henceforth, we will refer to this technique as *Split Miner filtering* (SMf).

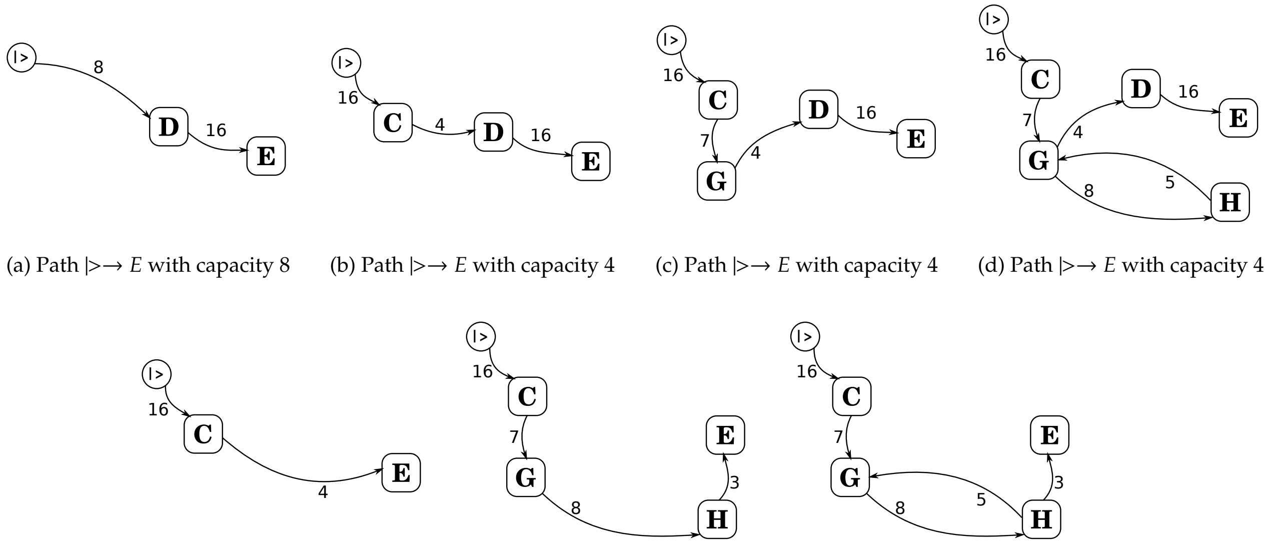
Note that Augusto et al.'s proposal aims to discover a BPMN process model, and thus, they apply different pruning strategies before the filtering phase. The starting point of their filtering technique is not the complete DFG, but a pruned DFG where the short cycles —i.e. cycles formed by two edges such as (a, b) and (b, a)— are pruned by removing the edge with less weight, or both edges if a concurrency relation between the vertices in the cycle is detected. In our approach, we do not need to remove such short cycles, hence we do not apply the short-cycle pruning step of the Split Miner algorithm, but only the filtering step.

We present below a declarative description of the SMf algorithm. The original one is a variant of Dijkstra's shortest path algorithm and is presented in [4].

```
Split Miner filtering:
  let $G = (V,E)$ be a sound DFG
  let $\omega$ be a weight function over $E$
  let $E'$ be an empy edge set
  for $v$ in $V$:
    add to $E'$ the $(u,v) \in E$ having max capacity
    add to $E'$ the $(v,u) \in E$ having max capacity
  return $(V,E')$
```

This algorithm computes sound and spanning *F-DFG*s. The spanning property is ensured because the algorithm does not remove any vertex in the filtering. Soundness is ensured because every vertex v has an incoming edge that is part of a path from *source* to v as well as an outgoing edge that is part of a path from v to *sink*.

We note that this algorithm may remove the edge with higher weight because the decision is based on the capacity of the paths, not on the edges weight. As an example, to choose which of the incoming edges of E to retain, Figure 7 shows all the paths |>→ E and their capacities. In this case, during the filtering, the retained incoming edge of E is (D, E), not because it is the edge with the highest weight, but because it is part



(a) Path |>→ E with capacity 8

(b) Path |>→ E with capacity 4

(c) Path |>→ E with capacity 4

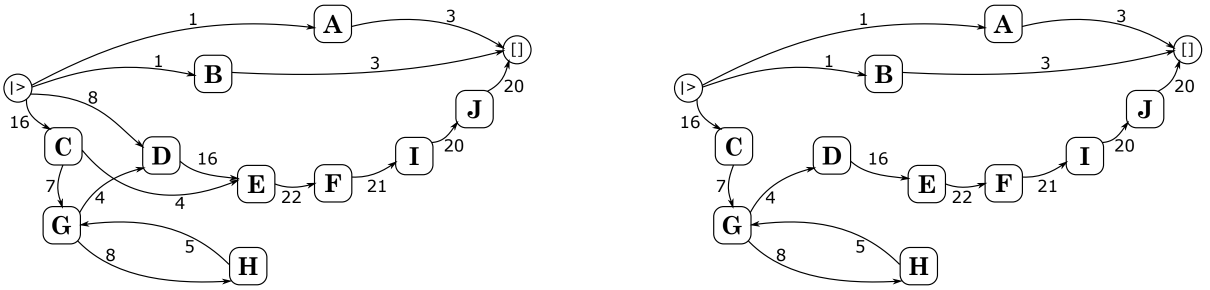
(d) Path |>→ E with capacity 4

(e) Path |>→ E with capacity 4

(f) Path |>→ E with capacity 3

(g) Path |>→ E with capacity 3

Figure 7: Paths |>→ E and their capacities —i.e. for each path, the least of its edges weight.



(a) *MWMF-DFG* approximation formed by 16 edges and with a total weight of 159, computed by Split Miner filtering.

(b) *MWMF-DFG* approximation formed by 14 edges and with a total weight of 147, computed by Split Miner filtering + Greedy.

Figure 8: *MWMF-DFG* approximations computed by SMf and SMf+G for the DFG in Figure 2b.

of the path with maximum capacity (Figure 7a). The same process is executed to retain one incoming and one outgoing edge per vertex. Figure 8a shows the *F-DFG* obtained by applying the Split Miner filtering to the DFG in Figure 2b. We can see that the edge (B, A) has been discarded in favor of $(|{>}, A)$ because the capacities of all paths $|{>} \rightarrow A$ are 1, and the latter was chosen first.

Similar to the TWE approach, the SMf result may contain redundant edges —e.g. (C, E)—, and thus, we also propose an alternative technique consisting of applying Greedy to the *F-DFG* computed by Split Miner filtering (SMf+G) in order to further minimize the number of edges. The output of SMf+G is depicted in Figure 8b, where we can see that the removal of the edge $(B.A)$ by SMf makes the edges $(|{>}, A)$ and $(B, [])$ necessary to maintain the soundness, causing SMf+G to not reach an optimal solution.

To perform the capacity calculation of each element, Augusto et al. propose a variant of Dijkstra's shortest path algorithm, which re-inserts a vertex to the search list when its capacity has been updated. This approach has a time complexity of $O(E + fV)$, where E is the number of edges, V is the number of vertices in the DFG, and f is the maximum number of incoming edges to a vertex in the graph. Similar to TWE+G, the *F-DFG* obtained by SMf has a maximum of $2V - 2$ edges. Hence, the time complexity of SMf+G is $O(E + V^2)$.

6. Evaluation

This section reports on an experimental evaluation of the proposed techniques in terms of their computational efficiency and in terms of their ability to maximally filter the full DFG of an event log while retaining the most frequent behavior recorded in the original log. The section first discusses the datasets used in the evaluation followed by the evaluation setup and the findings.

6.1. Datasets

In the interest of enhancing the ecological validity of the empirical evaluation, we took as a starting point a collection of real-life events logs available in the 4TU Centre for Research Data[9]. As of 30 November 2020, this collection contains 48 (real-life) event logs. We selected a subset of them using the following criteria:

- We discarded datasets in the collection that do not capture the execution of a business process as defined in [10], i.e. a collection of activities performed by multiple actors in view of providing an outcome that is of value to a customer. Given this definition, we discarded event logs recording the execution of software tests or software models (*JUnit 4.12* log, *Apache Commons Crypto*, *Statechart workbench*), a group of clickstream logs (*BPIC 2016*), an autonomous vehicle log (Nasa CEV), and a log of a lighting controller.
- We discarded event logs with trivial behavior, specifically a log with one single trace variant (*Credit Requirement*) and a log consisting of only three activities (*BPIC 2013*).
- Some of the event logs are part of a group. For example, there is a group of five logs that were used in the BPIC 2015 challenge. All these logs represent the same process executed at different organizations (in this case, five municipalities). In such situations, we only retained one representative log per group. Specifically, we only retained the first of the five *BPIC 2015* event logs. From the *Unrineweginfectie* group, recording urinary tract infection (UTI) cases, we only retained *Logboek 3* since it is the one with the largest number of trace variants. Similarly from the BPIC 2020 group, we retained the one with the largest number of trace variants (*Travel permit data*). Finally, from the BPIC 2014 group, we retained *Activity log for incidents* since the other two tables in this dataset do not have a timestamp column.
- We discarded event logs that correspond to pre-processed variants of other event logs included in the collection. Specifically, we discarded the CoSeLoG group of logs as this group contains pre-processed versions of logs in the *BPIC 2015* group. We also discarded a dataset containing pre-processed versions of other event logs in the collection, which were used in an automated process discovery benchmark.

Based on the above inclusion criteria, we ended up with 13 event logs. Table 2 shows the characteristics of the logs and of their full DFGs. The size of the logs varies from dozens to tens of thousands of trace variants and from hundreds to millions of events. The full DFGs range in size from 12 to 626 vertices and from 25 to 4821 edges. Note that there are two datasets with a notably higher number of vertices and edges (*Hospital Log* and *BPIC 2015*), but they have a ratio of edges to vertices (i.e. density) comparable to other logs. When it comes to density, the two most distinctive datasets are *UTI Cases* with 2 edges per vertex (lowest density) and *BPIC 2014* with 19 edges per vertex (highest density).

[9] `https://data.4tu.nl/search?q=:keyword:%20%22real%20life%20event%20logs%22`

Table 2: Characteristics of the datasets used in the experimentation: for each event log the number of traces (# *traces*), number of trace variants (# *trace variants*), number of events # *events*, and minimum (*Min.*) and maximum (*Max.*) trace length; and for the full DFG of each event log the number of vertices (# *vertices*), number of edges (# *edges*), and the ratio of the number of vertices to the number of edges ($|E|/|V|$).

	Event log					Full DFG		
Dataset	# traces	# trace variants	# events	Trace length Min.	Trace length Max.	# vertices	# edges	$\lvert E\rvert/\lvert V\rvert$
Hospital Log [28]	1,143	981	150,291	1	1,814	626	4,295	6.86
BPIC 2012 [29]	13,087	4,366	262,200	3	175	26	137	5.27
BPIC 2014 [30]	46,616	22,632	466,737	1	178	41	798	19.46
BPIC 2015 [31]	1,199	1,170	52,217	2	101	400	4,821	12.05
BPIC 2017 [32]	31,509	15,930	1,202,267	10	180	28	191	6.82
BPIC 2018 [35]	43,809	28,457	2,514,266	24	2,973	43	619	14.40
BPIC 2019 [33]	251,734	11,973	1,595,923	1	990	44	538	12.23
CCC 2019 [22]	20	20	697	26	59	31	150	4.84
BPIC 2020 [34]	7,065	1,478	86,581	3	90	53	568	10.72
Sepsis Cases [19]	1,050	846	15,214	3	185	18	135	7.50
Road Traffic [9]	150,370	231	561,470	2	20	13	78	6.00
UTI Cases [14]	1,650	50	6,973	2	35	12	25	2.08
Hospital Billing [20]	100,000	1,020	451,359	1	217	20	158	7.90

6.2. Experimental setup

We evaluated the five approaches presented in Section 5: Greedy (G), Two Ways Edmonds (TWE), Two Ways Edmonds combined with Greedy (TWE+G), Split Miner filtering (SMf), and Split Miner filtering combined with Greedy (SMf+G). The SMf technique is implemented in Java. This implementation takes as input a parameter η corresponding to the desired level of filtering. We set $\eta = 1.0$ so the resulting *F-DFG* is the most filtered one that this technique can produce. All the other techniques were implemented in Kotlin and they have no configuration parameters. The implementations of all five techniques are available at `https://github.com/david-chapela/dfg-edge-filtering`.

We recall that the problem addressed in this paper is that of efficiently computing a sound and spanning subgraph of a full DFG with a minimal number of edges and a maximal sum of edge frequencies, as formulated in Section 4. Accordingly, we evaluate the goodness of the presented algorithms in terms of the number of edges of the filtered DFGs they produce as well as the total weight of the filtered DFGs. We note that it is unfair to compare *F-DFG*s of different sizes in terms of their total weight, as an *F-DFG* with more edges will presumably have a higher total weight. To address this concern, before comparing the *MWMF-DFG* approximations produced by different algorithms for a given log, we normalize the sizes of these *MWMF-DFG* approximations so that they all have the same size. Let N be the size of the largest of the

MWMF-DFG approximations produced by different algorithms for a given log. To normalize, we take each *MWMF-DFG* approximation that has a size less than N, and we add back, one by one, the filtered edges in weight descending order until the size of the extended *MWMF-DFG* is equal to N. The total weight of the resulting extended *MWMF-DFG* is called the *normalized total weight*. We use this latter measure, instead of the total weight of the *MWMF-DFG* approximations.

Naturally, we also seek to obtain a filtered DFG that retains, to the maximal possible extent, the behavior recorded in the event log from which the full DFG is extracted. We also seek to obtain a precise filtered DFG, i.e. a filtered DFG that captures the smallest possible amount of extra behavior (i.e. behavior not observed in the event log). Accordingly, we also report on the fitness and precision of the filtered DFGs w.r.t. the original event logs. To measure fitness, we translate each maximally filtered DFG into an equivalent Petri net (by treating the filtered DFG as an automaton) and we then apply the alignment-based fitness measure defined in [1].[10] To measure precision, we use a recently proposed approach for measuring precision of Petri nets against event logs [15]. This precision measure evaluates the amount of behavior captured by the model but not observed in the event log. The method for calculating this measure takes as input a parameter called "number of skips" which corresponds to the maximum allowed number of non-conforming movements (skips) that may exist between a trace produced by the model and a "matching" trace in the log. If the number of skips required to match a given trace of the model with any one trace in the log is greater than this number, the technique considers that this model trace cannot be matched (i.e. this behavior does not occur in the log). We measured the precision with a number of skips of 3 and 5. We did not measure with higher values because increasing the number of skips impacts the computation time and causes more datasets to time-out, which makes the precision metric less useful for comparison purposes. Unfortunately, even when restricting the number of skips to a small number (3 or 5), the implementation of the precision measure in [15] does not scale up to the size of the largest event logs in this empirical evaluation. To cope with this limitation, we set a time-out of 4 hours when measuring precision. No precision values are reported in the case of a time-out.

Finally, we measured the efficiency of each approach by means of their execution time (runtime) from the moment the event log is read from secondary storage to the moment the maximally filtered DFG has been constructed in memory. All the experiments were conducted on Intel(R) Core(TM) i5-8250U with 8GB of RAM running JVM 11.

6.3. Results

In this section, we present the results obtained in the performed evaluation of the proposed techniques.

[10] This approach to measure fitness of filtered DFGs was proposed in [18].

Number of edges

Table 3 shows the number of edges of the *MWMF-DFG* approximation computed by each algorithm. SMf presents the worst results. It yields the smallest *F-DFG* only in the dataset with the lowest edge to vertex ratio —UTI Cases— where all techniques converge to the same result. TWE outperforms SMf in eight datasets, but both of them perform worse than the other algorithms. The best results are obtained by G, TWE+G, and SMf+G which produce filtered DFGs with very comparable number of edges. We note that SMf+G never yields better results than TWE+G, and is outperformed by it in three datasets. Conversely, TWE+G outperforms G in three datasets —BPIC 2014, BPIC 2015 and CCC 2019—, G outperforms TWE+G in one dataset —Hospital Log—, and both obtain the same results in the remaining nine datasets.

In order to statistically analyze if there is a difference between the techniques insofar as the number of edges is concerned, we performed a Friedman ranking test followed by a Holm post-hoc test using STAC [23]. The Friedman ranking test calculates a rank for the approaches based on their performance, where the lower the rank, the better. On the other hand, the Holm post-hoc method computes the significance of the difference between every two techniques. Table 4 shows the results of the statistical tests. We applied the Holm post-hoc test using the first ranked approach —TWE+G— as a control method, thus calculating the p-value for each comparison of TWE+G with each of the other approaches. We can see that TWE+G has been ranked as the best technique, with significance values proving that TWE+G is superior to TWE and SMf. Regarding TWE+G, SMf+G, and G, there is not enough evidence to conclude that their performance is different.

Table 3: Number of edges of the *MWMF-DFG* approximations computed by each of the proposed techniques, for the datasets presented in Table 2. The gray cells mark, for each dataset, the best result(s) out of the five filtering techniques.

	# edges				
Dataset	G	TWE	TWE+G	SMf	SMf+G
Hospital Log	918	1121	924	1155	928
BPIC 2012	33	36	33	38	33
BPIC 2014	65	74	63	74	65
BPIC 2015	496	633	489	675	495
BPIC 2017	37	40	37	41	37
BPIC 2018	57	70	57	72	57
BPIC 2019	58	73	58	73	58
CCC 2019	33	35	32	39	32
BPIC 2020	76	85	76	85	76
Sepsis Cases	22	24	22	25	22
Road Traffic	14	17	14	17	14
UTI Cases	17	17	17	17	17
Hospital Billing	26	30	26	31	26

Table 4: Non-parametric tests for the performance of the five techniques regarding the number of edges.

Algorithm	Friedman Ranking	Holm Adj. *p-value*
TWE+G	1.885	
SMf+G	2.154	1.000
G	2.192	1.000
TWE	4.077	2.8E-3
SMf	4.692	6.0E-5

Total weight

Table 5 provides the normalized total weights of the *MWMF-DFG* approximations produced by G, TWE+G, and SMf+G. We do not include TWE and SMf in this table because we showed above that their performance regarding the number of edges is clearly below that of the other techniques. We observe that SMf+G does not outperform the other two techniques in any dataset, although it achieves the best result in eight datasets, ex aequo with other techniques. G outperforms the other two techniques in one dataset and achieves the best results (ex aequo with other techniques) in other nine datasets. Finally, TWE+G outperforms the other two techniques in three datasets and achieves the best results (ex aequo with other techniques) in other seven datasets.

Fitness

Table 5 depicts the fitness of the *MWMF-DFG* approximations computed by each of the techniques. The relative performance of the techniques in terms of fitness is similar to their relative performance in terms of normalized total weight in all datasets except three (BPIC 2018, BPIC 2020, and Sepsis cases). In the BPIC 2018, the three techniques yield the same fitness value, although TWE+G obtained a worse normalized total weight. This can be explained by the small difference in the normalized total weight. The same occurs in the Sepsis cases dataset, where SMf+G yields slightly lower normalized total weight but similar fitness. Regarding the BPIC 2020, although the three techniques yield the same normalized total weight, TWE+G yields higher fitness.

We note that the fitness values shown in Table 5 are relatively low. This result is not surprising given that the DFGs of the filtered logs contain only a small fraction of the edges in the original log. For example, the normalized filtered DFGs of the Hospital Log only have 21% (918/4,295) of the edges of the full DFG. Even more strikingly, the normalized filtered DFGs of the BPIC 2015 log have only 10% (496/4,821) of the edges of the full DFG. While maximally filtered DFGs clearly enhance the understandability of the visualizations produced by process mining tools, this understandability comes at the cost of accuracy. Whenever possible, users of DFGs should consider adjusting the level of edge filtering in order to strike a balance between understandability and accuracy.

Table 5: Total weight, Alignment-based fitness, and precision of the *MWMF-DFG* approximation computed by G, TWE+G, and SMf+G, normalized by adding back some of the filtered edges in weight-descending order until the three filtered DFGs have the same size (denoted by # edges). The gray cells mark, for each dataset, the best result(s) out of the five filtering techniques.

		Normalized total weight			Alignment-based fitness			Precision (3 skips)		
Dataset	# edges	G	TWE+G	SMf+G	G	TWE+G	SMf+G	G	TWE+G	SMf+G
Hospital Log	928	69,528	52,891	31,039	0.22	0.15	0.03	-	-	-
BPIC 2012	33	107,036	107,036	107,036	0.55	0.55	0.55	0.81	0.81	0.81
BPIC 2014	65	154,708	215,645	172,383	0.48	0.54	0.49	-	-	-
BPIC 2015	496	16,389	20,317	17,429	0.18	0.30	0.26	-	-	-
BPIC 2017	37	615,759	594,525	615,759	0.29	0.28	0.29	-	-	-
BPIC 2018	57	620,672	620,629	620,672	0.32	0.32	0.32	-	-	-
BPIC 2019	58	545,986	545,986	545,986	0.47	0.47	0.47	-	-	-
CCC 2019	33	454	456	450	0.75	0.76	0.75	0.69	0.68	0.66
BPIC 2020	76	55,597	55,597	55,597	0.45	0.48	0.45	0.76	0.76	0.76
Sepsis Cases	22	8,013	8,013	7,852	0.64	0.64	0.64	0.72	0.72	0.70
Road Traffic	14	528,471	528,471	528,471	0.61	0.61	0.61	0.96	0.96	0.96
UTI Cases	17	7,776	7,776	7,776	0.91	0.91	0.91	1.00	1.00	1.00
Hospital Billing	26	347,248	347,248	347,248	0.68	0.68	0.68	0.94	0.94	0.94

We also note that the SMf+G approach leads to low fitness in the case of the Hospital Log, while the pure greedy (G) and the TWE+G approach achieve comparatively better fitness. This can be explained by the fact that this event log contains a large number of edges with a frequency of one in the full DFG, while some of the edges have a high frequency. Most of the nodes in this log have one incoming and one outgoing edge with high frequency, and then a number of other edges with a frequency of one or other low frequencies. This structure lends itself to the strategy followed by the greedy approach, which in such cases will naturally tend to retain the main pathways in the DFG. Conversely, the SMf approach tries to keep edges that are part of a path from a vertex to the sink (or from the source to a vertex) with the largest capacity. As a result, SMf may drop some paths that contain edges with relatively high frequency when these same paths also contain edges with very low frequency.

Precision

Table 5 depicts the precision (with the number of skips set to three) of the *MWMF-DFG* approximations computed by each of the techniques. A dash in this column indicates that the precision measurement timed-out after 4 hours. We observe that the results of the three algorithms are very close to each other in five out of the seven datasets. Regarding the other two, SMf+G obtains slightly lower precision values. Nevertheless, the values in these cases are comparable. We have also measured the precision setting the number of skips to five. When the number of skips is five, the values of precision are generally higher in absolute numbers (than with three skips), but in relative terms the results are similar.

We performed a statistical test to determine if we can assert a statistically significant difference between the techniques in terms of normalized total weight and fitness. In both cases, total weight and alignment-based fitness, the Friedman ranking placed TWE+G first, followed by G and SMf+G. However, the post-hoc significance results do not allow us to conclude that this difference is statistically significant. Regarding the precision, there are too few data points to do a statistical test.

Runtimes

Table 6 shows the average runtime of the evaluated techniques for each of the datasets —measured as the average of ten executions. TWE and TWE+G present the lowest runtimes in all cases except the two datasets having more than 400 activities —*Hospital Log* and *BPIC 2015*. Regarding these two datasets, G presents the worst runtimes exceeding 50 seconds in both cases, while SMf obtains the best runtimes with values under 100 milliseconds, followed by SMf+G with values under 2 seconds, and by TWE+G with runtimes of 4 and 5 seconds.

Table 7 shows the results of the Friedman ranking and Holm post-hoc tests regarding Table 6 runtime values. We have used the first ranked approach —TWE— as a control method for the Holm post-hoc tests. As we can see, the most efficient technique is TWE, followed by TWE+G. Regarding the post-hoc tests, there is a significant difference to conclude that TWE outperforms G, SMf, and SMf+G, but not to determine it outperforms TWE+G.

Table 6: Runtimes of the proposed techniques measured as the average of ten executions. The gray cells mark, for each dataset, the best result(s) out of the five filtering techniques.

	Runtime (ms)				
Dataset	G	TWE	TWE+G	SMf	SMf+G
Hospital Log	54,721	2,692	5,148	97	1,379
BPIC 2012	20	7	13	64	66
BPIC 2014	145	57	67	119	127
BPIC 2015	50,077	2,241	4,054	77	587
BPIC 2017	39	12	19	157	160
BPIC 2018	127	49	77	454	462
BPIC 2019	110	34	50	99	111
CCC 2019	30	10	12	11	14
BPIC 2020	127	27	35	30	38
Sepsis Cases	16	6	16	26	27
Road Traffic	6	2	5	11	12
UTI Cases	2	1	2	10	11
Hospital Billing	20	4	7	18	20

Table 7: Non-parametric tests for the performance of the five techniques regarding the runtimes.

Algorithm	Friedman Ranking	Holm Adj. *p-value*
TWE	1.308	
TWE+G	2.538	0.204
G	3.000	2.9E-4
SMf	3.885	0.045
SMf+G	4.269	2E-5

Threats to validity

The reported evaluation has a number of threats to validity. First, a potential threat to internal validity in regard to the evaluation of runtime execution times is the fact that we conducted experiments using a single computing environment. The results might differ on other computing environments. To mitigate this threat to validity, we executed each experiment tend times and reported the average. We did not observe major variations between different executions. To ensure the reproducibility of the results, we have relied on publicly available logs and we have publicly released the implementations of the proposed techniques.

Another threat to potential validity is the fact that we relied on only one measure of fitness. However, this measure is widely used in the field of automated process discovery [3]. Related to the above, we compared the total weights using a normalization approach. There may be other approaches to perform such a comparison.

A potential threat to external validity is given by the use of a limited number of event logs (13 logs). To mitigate this risk, we selected these log using carefully justified criteria, in such a way that the selected logs are representative of a broader pool of real-life logs. Furthermore, the event logs cover different industry domains (banking, IT services, healthcare, etc.) and they have a wide range of characteristics with respect to size and complexity.

7. Conclusion and future work

In this paper, we have formalized the problem of DFG simplification as an optimization problem where we seek to obtain a filtered DFG with the least possible number of edges while maximizing the frequency of the retained edges —i.e. the total weight. We have shown that this problem is an instance of an $\mathcal{NP}$-hard problem from the graph theory field. Accordingly, we have presented a set of polynomial-time heuristics to approximate the problem, and we have conducted an evaluation to compare the optimality and the runtime performance of these heuristics.

Based on the results, we found that none of the heuristics outperforms the others across all datasets. In 9 out of the 13 datasets, three of the heuristics (G, TWE+G, and SMf+G) yield the lowest number of edges. The TWE+G approach produces the lowest number of edges in other three datasets —in one of which it does so ex aqueo with SMf+G—, while Greedy outperforms the other approaches in the remaining dataset. Regarding the total weight, similar observations can be made. TWE+G obtains the best results in most of the datasets, outperforming G in three of them, but being outperformed by it in other three. SMf+G never outperforms the other techniques, but it yields the same results in seven datasets. Similar observations can be made when comparing the relative performance of the proposed techniques in terms of fitness. Regarding precision, all techniques present very comparable results, with slightly lower values of SMf+G in two of the datasets. We have performed a set of non-parametric statistical tests to compare the techniques regarding the filtering performance, concluding that there is not enough evidence to say there is a difference between TWE+G, G, and SMf+G. On the other hand, these three techniques outperform TWE and SMf in a statistically significantly manner.

Regarding the execution time, the most efficient technique is TWE, closely followed by TWE+G. TWE+G outperforms G in execution time in all the cases, and it underperforms SMf+G only in the two datasets that contain with more than 400 activities. This behavior was predictable given the worst-time complexity analysis of the algorithms (cf. Section 5). Indeed, TWE is more dependent on the number of vertices than SMf —with time complexities of $O(EV)$ and $O(E + fV)$, respectively.

In summary, barring situations where the number of activities in the process is high, TWE+G offers the best trade-off between filtering performance and execution time. In the case of processes with hundreds of activities, SMf+G is a preferable option, as it sacrifices filtering performance but yields lower execution times.

One of the purposes of DFG filtering is to produce DFGs that are easier to comprehend. In this paper, we measured the simplicity of a DFG in terms of the number of edges. While the number of edges has been shown to be correlated with understandability in the context of process modeling [21], a lower number of edges does not always imply higher understandability. An avenue for future work is to conduct user studies to determine how different DFG filtering techniques compare to each other in a practical setting. Several algorithms for automated discovery of process models (BPMN models or Petri nets) take a DFG as a starting point, e.g. Split Miner [4], Inductive Miner [17], Fodina [36]. A possible direction for future work is to study how the DFG filtering approaches proposed in this paper can be integrated into these automatic process discovery algorithms and what trade-off —e.g. in terms of accuracy or simplicity measures— they provide relative to the existing DFG filtering methods integrated into these algorithms.

Acknowledgments.

We thank Luciano García-Bañuelos for proposing the idea of combining the results of Chu-Liu-Edmonds' algorithm to filter a DFG. We also thank Adriano Augusto for providing us with the implementation of the Split Miner filtering technique. This research was funded by the Spanish Ministry of Economy and

Competitiveness (TIN2017-84796-C2-1-R) and the Galician Ministry of Education, Culture and Universities (ED431G/08). These grants are co-funded by the European Regional Development Fund (ERDF/FEDER program). D. Chapela-Campa is supported by the Spanish Ministry of Education, under the FPU national plan (FPU16/04428 and EST19/00135). This research is also funded by the Estonian Research Council (grant PRG1226).

References

[1] A. Adriansyah, B. F. van Dongen, W. M. P. van der Aalst, 2011. Conformance checking using cost-based fitness analysis. In: Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2011, Helsinki, Finland, August 29 - September 2, 2011. IEEE Computer Society, pp. 55–64.

[2] A. Armas-Cervantes, M. Dumas, M. L. Rosa, A. Maaradji, 2019. Local concurrency detection in business process event logs. ACM Transactions on Internet Technology 19 (1), 16:1–16:23.

[3] A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, F. M. Maggi, A. Marrella, M. Mecella, A. Soo, 2019. Automated discovery of process models from event logs: Review and benchmark. IEEE Transactions on Knowledge and Data Engineering 31 (4), 686–705.

[4] A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, A. Polyvyanyy, 2019. Split miner: automated discovery of accurate and simple business process models from event logs. Knowledge and Information Systems 59 (2), 251–284.

[5] J. Bang-Jensen, G. Z. Gutin, 2009. Digraphs - Theory, Algorithms and Applications, Second Edition. Springer Monographs in Mathematics. Springer.

[6] P. M. Camerini, L. Fratta, F. Maffioli, 1979. A note on finding optimum branchings. Networks 9 (4), 309–312.

[7] Y. J. Chu, T. H. Liu, 1965. On the shortest arborescence of a directed graph. Scientia Sinica 14, 1396–1400.

[8] R. Conforti, M. L. Rosa, A. H. M. ter Hofstede, 2017. Filtering out infrequent behavior from business process event logs. IEEE Transactions on Knowledge and Data Engineering 29 (2), 300–314.

[9] M. M. de Leoni, F. Mannhardt, Feb 2015. Road traffic fine management process.

[10] M. Dumas, M. L. Rosa, J. Mendling, H. A. Reijers, 2018. Fundamentals of Business Process Management, Second Edition. Springer.

[11] J. Edmonds, 1967. Optimum branchings. Journal of Research of the national Bureau of Standards B 71 (4), 233–240.

[12] G. N. Frederickson, J. JáJá, 1981. Approximation algorithms for several graph augmentation problems. SIAM Journal on Computing 10 (2), 270–283.

[13] H. N. Gabow, Z. Galil, T. H. Spencer, R. E. Tarjan, 1986. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. Combinatorica 6 (2), 109–122.

[14] P. Gunst, Aug 2020. Urineweginfectie (uwi-casus) logboek.

[15] A. A. Kalenkova, A. Polyvyanyy, 2020. A spectrum of entropy-based precision and recall measurements between partially matching designed and observed processes. In: E. Kafeza, B. Benatallah, F. Martinelli, H. Hacid, A. Bouguettaya, H. Motahari (Eds.), Service-Oriented Computing - 18th International Conference, ICSOC 2020, Dubai, United Arab Emirates, December 14-17, 2020, Proceedings. Vol. 12571 of Lecture Notes in Computer Science. Springer, pp. 337–354.

[16] S. Khuller, B. Raghavachari, N. E. Young, 1995. Approximating the minimum equivalent digraph. SIAM Journal on Computing 24 (4), 859–872.

[17] S. J. J. Leemans, D. Fahland, W. M. P. van der Aalst, 2013. Discovering block-structured process models from event logs containing infrequent behaviour. In: N. Lohmann, M. Song, P. Wohed (Eds.), Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers. Vol. 171 of Lecture Notes in Business Information Processing. Springer, pp. 66–78.

[18] S. J. J. Leemans, E. Poppe, M. T. Wynn, 2019. Directly follows-based process mining: Exploration & a case study. In: International Conference on Process Mining, ICPM 2019, Aachen, Germany, June 24-26, 2019. IEEE, pp. 25–32.

[19] F. Mannhardt, Dec 2016. Sepsis cases - event log.

[20] F. Mannhardt, Aug 2017. Hospital billing - event log.

[21] J. Mendling, H. A. Reijers, W. M. P. van der Aalst, 2010. Seven process modeling guidelines (7PMG). Information & Software Technology 52 (2), 127–136.

[22] J. Munoz-Gama, R. R. de la Fuente, M. M. Sepúlveda, R. R. Fuentes, Feb 2019. Conformance checking challenge 2019 (ccc19).

[23] I. Rodríguez-Fdez, A. Canosa, M. Mucientes, A. Bugarín, 2015. STAC: A web platform for the comparison of algorithms using statistical tests. In: A. Yazici, N. R. Pal, U. Kaymak, T. Martin, H. Ishibuchi, C. Lin, J. M. C. Sousa, B. Tütmez (Eds.), 2015 IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2015, Istanbul, Turkey, August 2-5, 2015. IEEE, pp. 1–8.

[24] S. Smirnov, M. Weidlich, J. Mendling, 2012. Business process model abstraction based on synthesis from well-structured behavioral profiles. International Journal of Cooperative Information Systems 21 (1), 55–83.

[25] R. E. Tarjan, 1977. Finding optimum branchings. Networks 7 (1), 25–35.

[26] W. M. van der Aalst, 2019. A practitioner's guide to process mining: Limitations of the directly-follows graph.

[27] W. M. P. van der Aalst, 2016. Process Mining - Data Science in Action, Second Edition. Springer.

[28] B. van Dongen, Mar 2011. Real-life event logs - hospital log.

[29] B. van Dongen, Apr 2012. Bpi challenge 2012.

[30] B. van Dongen, Apr 2014. Bpi challenge 2014: Activity log for incidents.

[31] B. van Dongen, May 2015. Bpi challenge 2015 municipality 1.

[32] B. van Dongen, Feb 2017. Bpi challenge 2017.

[33] B. van Dongen, Jan 2019. Bpi challenge 2019.

[34] B. van Dongen, Mar 2020. Bpi challenge 2020: Travel permit data.

[35] B. van Dongen, F. F. Borchert, Mar 2018. Bpi challenge 2018.

[36] S. K. L. M. vanden Broucke, J. D. Weerdt, 2017. Fodina: A robust and flexible heuristic process discovery technique. Decision Support Systems 100, 109–118.

[37] A. J. M. M. Weijters, J. T. S. Ribeiro, 2011. Flexible heuristics miner (FHM). In: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, part of the IEEE Symposium Series on Computational Intelligence 2011, April 11-15, 2011, Paris, France. IEEE, pp. 310–317.