CENTERIS - International Conference on ENTERprise Information Systems / ProjMAN - International Conference on Project MANagement / HCist - International Conference on Health and Social Care Information Systems and Technologies

A practitioner's guide to process mining: Limitations of the directly-follows graph

Wil M.P. van der Aalst[a,b]*

[a]*Process and Data Science (PADS), RWTH Aachen University, Aachen, Germany*
[b]*Fraunhofer Institute for Applied Information Technology, Sankt Augustin, Germany*

Abstract

Process mining techniques use event data to show what people, machines, and organizations are really doing. Process mining provides novel insights that can be used to identify and address performance and compliance problems. In recent years, the adoption of process mining in practice increased rapidly. It is interesting to see how ideas first developed in open-source tools like ProM, get transferred to the dozens of available commercial process mining tools. However, these tools still resort to producing Directly-Follows Graphs (DFGs) based on event data rather than using more sophisticated notations also able to capture concurrency. Moreover, to tackle complexity, DFGs are seamlessly simplified by removing nodes and edges based on frequency thresholds. Process-mining practitioners tend to use such simplified DFGs actively. Despite their simplicity, these DFGs may be misleading and users need to know how these process models are generated before interpreting them. In this paper, we discuss the pitfalls of using simple DFGs generated by commercial tools. Practitioners conducting a process-mining project need to understand the risks associated with the (incorrect) use of DFGs and frequency-based simplification. Therefore, we put these risks in the spotlight.

Keywords: process mining, process discovery, directly-follows graphs, conformance checking

_____

* Corresponding author. Tel.: +49 241 80 21901 *E-mail address:* wvdaalst@pads.rwth-aachen.de

1. Introduction

Process mining starts from event data, as shown in Table 1. Input for process mining is an *event log*. An event log views a process from a particular angle. Each event in the log refers to (1) a particular process instance (called *case*), (2) an *activity*, and (3) a *timestamp*. There may be additional event attributes referring to resources, people, costs, etc., but these are optional. With some effort, such data can be extracted from any information system supporting operational processes. Process mining uses these event data to answer a variety of process-related questions. Process mining techniques such as process discovery, conformance checking, model enhancement, and operational support can be used to improve performance and compliance [1].

Table 1. Small fragment of a larger event log capturing the Purchase-to-Pay (P2P) process.

case id (here an order)	activity	timestamp	resource	costs	customer
…	…	…	…	…	..
2019-88201	create purchase requisition	25-07-2019:09.15	John	€20.20	9950
2019-88201	create purchase order	25-07-2019:09.35	Mary	€48.30	9950
2019-88201	approve purchase order	25-07-2019:09.55	Sue	€30.70	9950
2019-88202	create purchase requisition	25-07-2019:10.15	John	€28.20	9955
2019-88202	create purchase order	25-07-2019:10.25	Mary	€29.30	9955
2019-88202	approve purchase order	25-07-2019:10.40	Sue	€37.60	9955
2019-88201	receive order confirmation	25-07-2019:11.50	Mary	€42.10	9950
2019-88201	receive goods	27-07-2019:09.35	Peter	€50.20	9950
2019-88202	receive order confirmation	27-07-2019:09.45	Mary	€42.30	9955
2019-88202	receive invoice	28-07-2019:10.10	Sue	€44.90	9955
2019-88201	receive invoice	28-07-2019:10.20	Sue	€30.80	9950
2019-88201	pay invoice	29-07-2019:11.05	Sue	€30.70	9950
2019-88202	receive goods	29-07-2019:11.35	Peter	€51.30	9955
2019-88202	pay invoice	29-07-2019:12.15	Sue	€29.20	9955
…	..	…	…	…	…

Table 1 only shows a small fragment of a larger event log with events related to the so-called Purchase-to-Pay (P2P) process. The P2P process includes all business activities related to purchase orders, e.g., requesting (requisitioning), purchasing, receiving, paying for and accounting for goods and services. The first three columns show the mandatory event attributes: case (i.e., process instance), activity, and timestamp. Additional information such as the resource performing the activity is optional.

Event data can be used to discover process models automatically. Process models can be expressed using different formalisms ranging from Directly-Follows Graphs (DFGs) and accepting automata to Petri nets, BPMN diagrams, and UML activity diagrams. Fig. 1 shows three process models that could have been discovered based on the events in Table 1.
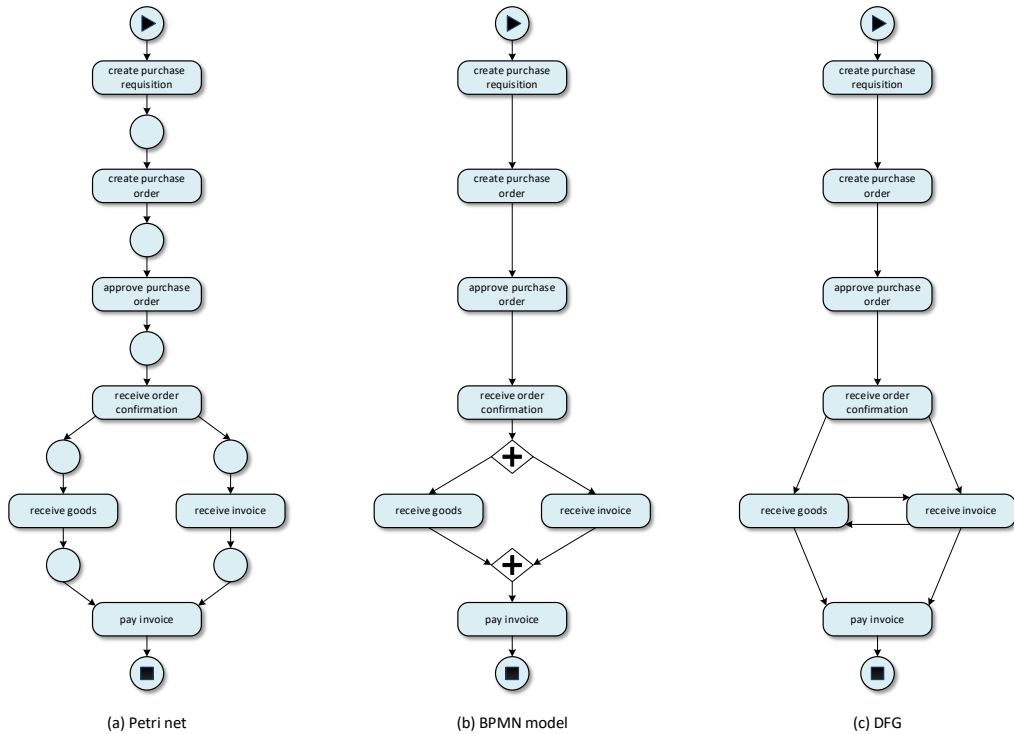
Fig. 1. Three process models discovered for the Purchase-to-Pay (P2P) process considering only the frequent "happy paths". The Petri net model (a) and the Business Process Model and Notation (BPMN) model (b) specify the same behavior. The Directly-Follows Graph (DFG) (c) allows for traces not allowed in the Petri net and BPMN model (e.g., a loop involving the activities *receive goods* and *receive invoice*).

The process models in Fig. 1 are very simple since they only consider the mainstream behavior also referred to as the "happy paths". For real P2P processes of larger organizations, there will be thousands of observed *unique traces* (also known as *process variants*). The frequency distribution of traces in an event log typically follows a power law where a small fraction of all variants accounts for most of the cases. For example, more than 80 percent of all cases can be described by less than 20 percent of all process variants (Pareto principle). For such processes, it is easy to create a simplified process model covering mainstream behavior. However, the cases not covered by such a simplified model are typically very diverse and account for most of the process variants. These non-mainstream cases are also likely to cause most of the performance and compliance problems. Conformance checking can be used to check if reality, as recorded in the event log, conforms to the model and vice versa. The process model may have been made by hand or learned using process discover (based on the frequent process variants).

For a comprehensive introduction to process mining, we refer to [1]. Process mining extends far beyond process discovery and conformance checking. For example, it is possible to predict performance and compliance problems and recommend process interventions. However, these more advanced techniques are out of scope in this paper.

In March 1968, Edsger Dijkstra's letter "Go To Statement Considered Harmful" was published in the Communications of the ACM [4]. In his letter, Dijkstra criticized the excessive use of the GOTO statement in programming languages of the day and advocated structured programming instead. Analogously, this paper could have been called "Directly-Follows Graphs (DFGs) considered harmful" because DFGs are often used and wrongly interpreted. Currently, there are more than 30 commercial offerings of process mining software (e.g., Celonis, Disco, ProcessGold, myInvenio, PAFnow, Minit, QPR, Mehrwerk, Puzzledata, LanaLabs, StereoLogic, Everflow, TimelinePI, Signavio, and Logpickr). They all start from DFGs for discovery. However, as shown in this paper, there are several possible problems related to the naïve use of DFGs:

- Activities that have a flexible ordering (e.g., due to concurrency) lead to Spaghetti-like DFGs with loops even when activities are executed at most once.

- DFGs can be simplified using frequency-based thresholds. However, this may lead to all kinds of interpretation problems due to "invisible gaps" in the model.
- Performance information mapped onto DFGs can be misleading, e.g., the average time reported between two activities is conditional (only the situations where they directly follow each other are considered).

The remainder of this paper is organized as follows. Section 2 shows the dangers of naïvely using Directly-Follows Graphs (DFGs). This is relevant for practitioners since DFGs are the "de facto standard" in commercial process mining tools. Section 3 concludes the paper and provides some pointers for further reading.

2. Beyond directly follows graphs

We first introduce Directly-Follows Graphs (DFGs) and show how they can be discovered from event data. Then, we discuss the problems mentioned in the introduction.

2.1. Creating a Directly-Follows Graph (DFG)

There are many possible process-modeling notations. Fig. 1 shows three examples. Most of the commercial process mining tools use DFGs as a first means to explore the event data. The basic idea is very simple, but first, we introduce some terms to explain the construction of a DFG. An a-event is an *event* that corresponds to activity a. A *trace* (also called *process variant*) $\sigma = \langle a_1, a_2, a_3, \ldots, a_n \rangle$ is a sequence of *activities*. $\#_L(\sigma)$ is the number of cases in event log L that correspond to trace σ. Note that many cases may have the same trace. $\#_L(a)$ is the number of a-events in event log L. $\#_L(a, b)$ is the number of times an a-event is directly followed by a b-event within the same case. Without loss of generality, we assume that each case starts with a start event (denoted ►) and end with an end event (denoted ■). If such start and end activities do not exist, they can be added to the start and end of each case. Hence, traces (process variants) are of the form $\sigma = \langle ►, a_2, a_3, \ldots, a_{n-1}, ■ \rangle$ where the start and events only appear at the start and end.

A DFG is a graph with nodes that correspond to activities and directed edges that corresponds to directly-follows relationships. There are three parameters, i.e., τ_{var}, τ_{act}, and τ_{df}, that define thresholds for the minimal number of traces for each variant included (based on $\#_L(\sigma)$), the minimal number of events for each activity included (based on $\#_L(a)$), and the minimal number of direct successions for each relation included (based on $\#_L(a, b)$).

1. Input: event log L and parameters τ_{var}, τ_{act}, and τ_{df}.
2. Remove all cases from L having a trace with a frequency lower than τ_{var}, i.e., keep all cases with a trace σ such that $\#_L(\sigma) \geq \tau_{var}$. The new event log is L'. Note that the number of cases may have be reduced considerably, but the retained cases remain unchanged.
3. Remove all events from L' corresponding to activities with a frequency lower than τ_{act}, i.e., keep events for which the corresponding activity a meets the requirement $\#_{L'}(a) \geq \tau_{act}$. The new event log is L''. Note that the number of cases did not change, but the number of events may be much lower.
4. Add a node for each activity remaining in the filtered event log L''.
5. Connect the nodes that meet the τ_{df} threshold, i.e., activities a and b are connected if and only if $\#_{L''}(a, b) \geq \tau_{df}$.
6. Output the resulting graph. Nodes are decorated with the activity frequency $\#_{L''}(a)$ and edges are decorated with the directly-follows frequency $\#_{L''}(a, b)$.

Nodes and edges can also be decorated with timing information. Note that an edge connecting activities a and b corresponds to $\#_{L''}(a, b)$ observations of activity a being followed by activity b. It is easy to compute the sum, mean, median, minimum, maximum, and standard deviation over these $\#_{L''}(a, b)$ observations.

2.2. Misleading diagnostics

All commercial process-mining tools support the above algorithm (or a variant of it). However, note that for performance reasons, most tools implement the third step differently and do no create a new event log L'' where low frequent activities are removed. Instead, edges are filtered on the overall DFG while removing low frequent activities. This may lead to misleading results. Consider the trace $\langle a, b, c \rangle$ and assume that that b is a low frequent activity with $\#_L(b) < \tau_{act}$. After removing activity b, trace $\langle a, b, c \rangle$ becomes trace $\langle a, c \rangle$ and a is directly followed by c. Filtering

edges based on the graph will miss that a is directly followed by c after removing b. As a result, removed activities are not shown in the model, but still influence the statistics. Even when τ_{df} is set to 0, the frequency of a selected node may be different from the sum of the frequencies of the input edges and both may be different from the sum of the frequencies of the output edges.
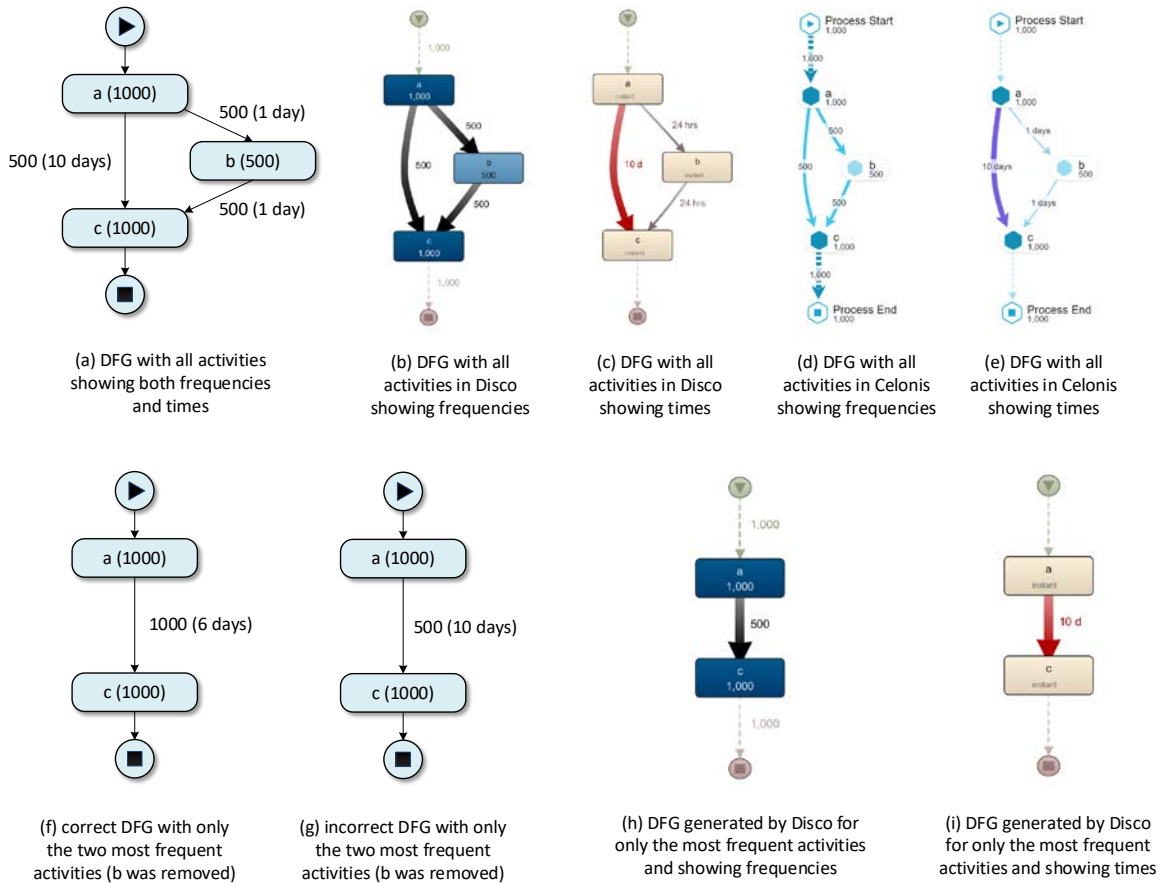

(a) DFG with all activities showing both frequencies and times

(b) DFG with all activities in Disco showing frequencies

(c) DFG with all activities in Disco showing times

(d) DFG with all activities in Celonis showing frequencies

(e) DFG with all activities in Celonis showing times

(f) correct DFG with only the two most frequent activities (b was removed)

(g) incorrect DFG with only the two most frequent activities (b was removed)

(h) DFG generated by Disco for only the most frequent activities and showing frequencies

(i) DFG generated by Disco for only the most frequent activities and showing times

Fig. 2. Different DFGs generated for a simple artificial event log $L = [\langle a, b, c \rangle^{500}, \langle a, c \rangle^{500}]$ showing frequencies and times. The example shows that it is easy to misinterpret DFGs. One needs to understand the way these models are derived from event data to correctly interpret the results.

Let us consider a simple artificial event log $L = [\langle a, b, c \rangle^{500}, \langle a, c \rangle^{500}]$ to illustrate the subtle but important differences between alternative DFG computations. Event log L has 1000 cases and two variants: 500 cases follow trace $\langle a, b, c \rangle$ and 500 cases follow trace $\langle a, c \rangle$. The time in-between activities a and b and the time in-between activities b and c in the first variant is always precisely one day (i.e., two days in total). The time in-between activities a and c in the second variant is always precisely 10 days. Fig. 2 shows various DFGs generated for this event log. Fig. 2(a) shows the DFG with low values for the thresholds τ_{var}, τ_{act}, and τ_{df}. As a result, all variants, all activities, and all directly-follows relations are included. Fig. 2(b-e) show the corresponding DFGs generated by Disco (from Fluxicon, Version 2.2.1, www.fluxicon.com) and Celonis (from Celonis, Version 4.4, www.celonis.com) showing frequencies and mean times. Now assume that τ_{act} is set to a value in-between 500 and 1000. As a result, activity b is removed. Applying the algorithm provided above, we obtain the DFG in Fig. 2(f). This DFG correctly shows that all 1000 a-events are followed by a b-event with on average a delay of 6 days. However, when removing activity b without creating a new log without b-events we obtain the DFG in Fig. 2(g). This DFG suggests that activity a is not always followed by activity c (only 50%) and that the delay between both is 10 days. However, activity a is always followed by activity c and the total flow time is just 6 days. Fig. 2(h-i) show the corresponding DFGs generated by

Disco. These DFGs match the misleading DFG in Fig. 2(g). Note that this is not specific for Disco. Most commercial process mining tools generate the same DFGs. This illustrates that one should be very careful when interpreting DFGs.
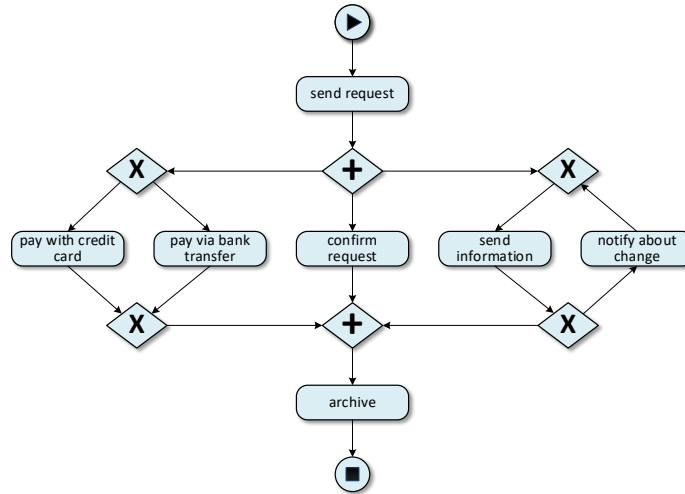

Fig. 3. Process model used to generate an event log with 10,000 cases.

One could argue that the misleading results in Fig. 2 stem from an incorrect implementation of the DFG algorithm. However, also correctly generated DFGs have the problem that different interleavings of the same set of activities automatically leads to loops even when things are executed only once. Consider, for example, the process model shown in Fig. 3. The process starts with activity *send request* and ends with activity *archive*. In-between these two activities there are three independent parallel branches: (1) a choice between activity *pay with credit card* and activity *pay via bank transfer*, (2) activity *confirm request*, and (3) a possible loop involving activities *send information* and *notify about change*. We used CPN Tools to simulate the process and generated 10,000 cases following the process in Fig. 3. In total, the event log has 117,172 events and 7 unique activities. The 10,000 cases correspond to 1159 process variants. The most frequent variant occurs 96 times. 30% of all variants occurred only once. 80% of the cases are described by 31% of the variants.
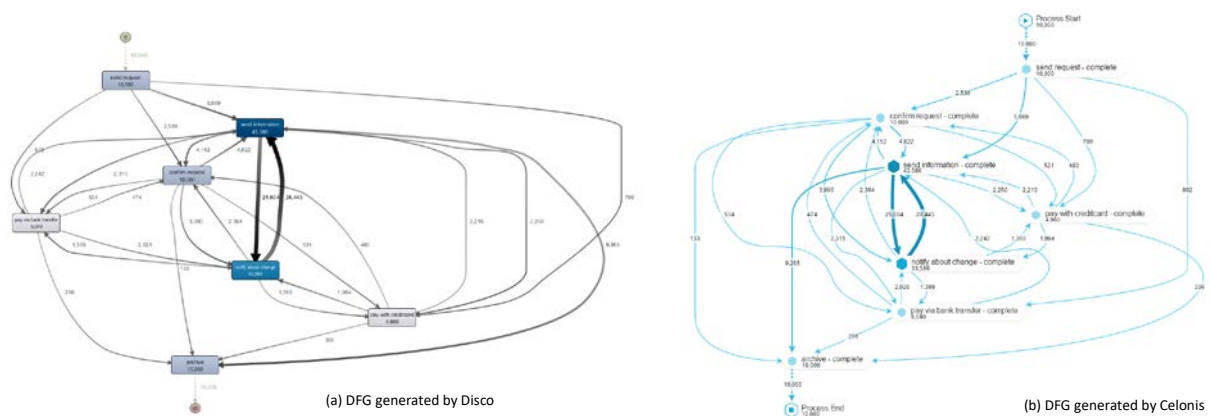

Fig. 4. DFGs generated by Disco and Celonis without any filtering (i.e., the lowest possible values were used for the thresholds τ_{var}, τ_{act}, and τ_{df}). Apart from layout differences, both tools produce the same process model. The diagrams are not intended to be readable, but aim to show the many loops. The DFGs are almost fully connected, not showing the underlying process structure.

Fig. 4 shows two DFGs created for the event log just described. One DFG was generated by Disco and the other DFG was generated by Celonis. The two DFGs are identical apart from their layout. Due to the different ways in which activities can be ordered, the DFG has many edges and these edges form loops also among activities that are executed at most once. Activity *pay with credit card* and activity *pay via bank transfer* form a length-two loop with *confirm request* although none of these activities were executed multiple times for the same case. To address the complexity and remove loops, one is tempted to simplify the DFG by increasing the thresholds τ_{var}, τ_{act}, and τ_{df}. Increasing the value for threshold τ_{var} is quite harmless because it is clear that the resulting process model only applies to the most frequent variants. However, also in the most frequent process variants activities do not need to occur in a fixed order. In an attempt to remove the loops one may also increase the value for thresholds τ_{act} and τ_{df} leading to new problems.
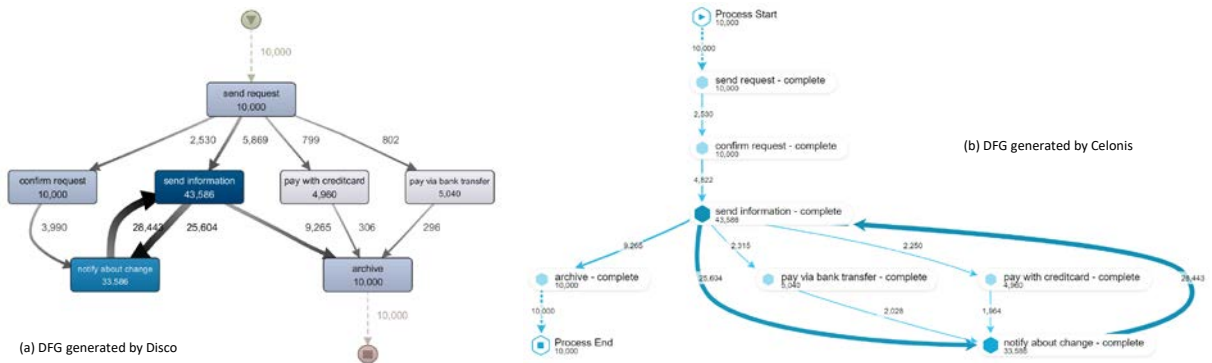

Fig. 5. DFGs generated by Disco and Celonis using the highest possible threshold for τ_{df} allowed by the software while retaining all activities and variants (i.e., the lowest possible values are used for the thresholds τ_{var} and τ_{act}).

Fig. 5 shows two additional DFGs generated by Disco and Celonis. To simplify the models as much as possible, we removed as many edges as allowed by the software. Due to the different implementations of the DFG algorithm, the two resulting process models are different. Moreover, both are misleading. At first glance, the DFG generated by Disco (Fig. 5(a)) seems closest to the process model in Fig. 3. However, the connection between *confirm request* and *notify about change* does not make any sense. Also, the routing logic (AND/XOR-split/join) is missing and the numbers are very misleading. For example, *send request* is 10,000 times followed by *confirm request* and not just 2,530 times. The DFG generated by Celonis (Fig. 5(b)) has even more problems. The loops involving the two payment types are counter-intuitive. Also, it is odd that payment seems to require activity *confirm request* and activity *send information* (whereas in the real process payments often precede these two activities). It is also quite disturbing that two tools generate two completely different DFGs allowing for contradictory conclusions.

The DFGs in Fig. 5 can also be used to analyze the bottlenecks in the process. However, most of the traces cannot be replayed on the DFG and the reported times between two activities are conditional. For example, the average time between *send request* and *confirm request* is 3.5 days (set in simulation model), but Disco and Celonis both report 1.5 days (considering only 2,530 of 10,000 cases). Hence, one cannot rely on DFG-based performance diagnostics.

Fig. 6 shows that one can use other representations that do not have the problems just mentioned. The three process models in Fig. 6 were discovered using three different process discovery techniques implemented in ProM [1]. All three models are behaviorally equivalent to the original process model that was used to generate the event log. Comparing Fig. 6 with the DFGs depicted in Fig. 4 and Fig. 5 illustrates the limited expressiveness of DFGs and the risks of simplifying DFGs using thresholds.

(a) C-net generated by ProM's Heuristic Miner (HM)

(b) Petri net generated by ProM's Alpha Miner (AM)

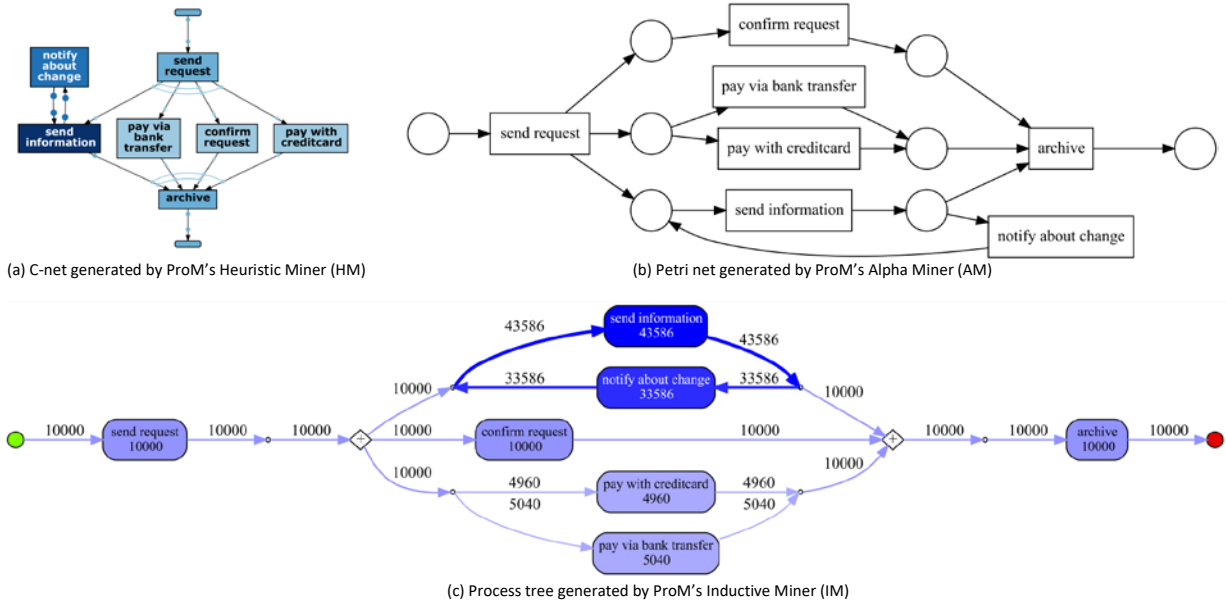(c) Process tree generated by ProM's Inductive Miner (IM)

Fig. 6. Three process models created by three different mining algorithms implemented in ProM (HM, AM, and IM) and using three different representations (C-nets, Petri nets, and process trees). All three models are behaviorally equivalent to the original process model in Fig. 3.

3. Conclusion

The process mining discipline is maturing. This is not only reflected by the uptake in industry, but also by the success of the inaugural International Conference on Process Mining (ICPM) in Aachen in June 2019. ICPM 2019 attracted over 400 participants and the number of new scientific papers on process mining is increasing every year. However, most practitioners are still using very basic approaches generating simple Directly-Follows Graphs (DFGs). This paper showed that DFGs can be very misleading and that practitioners need to understand the way that process models are discovered. DFGs are often wrongly interpreted and can be generated in different ways leading to very different conclusions. Also, bottleneck information may be deceiving, especially after model simplification.

For more information, we refer to [1]. In [2] we also discuss the role of using different abstractions (DFGs being one of them). In [3] we discuss another topic highly relevant for process mining practitioners: The selection of an appropriate set of case notions. Often multiple case notions are intertwined. In [5] various practical hints are given to deal with recurring problems such as data quality.

Acknowledgments

References

[1] Aalst, Wil van der. Process mining: Data science in action. Springer-Verlag, Berlin, 2016.

[2] Aalst, Wil van der. Process discovery from event data: Relating models and logs through abstractions. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 8(3) 2018.

[3] Aalst, Wil van der. Object-centric process mining: Dealing with divergence and convergence in event data. Proceedings of the 17th International Conference on Software Engineering and Formal Methods (SEFM 2019), LNCS, Springer-Verlag, Berlin, 2019.

[4] Dijkstra, Edsger. Go to statement considered harmful. Communications of the ACM, 11(3):147-148, 1968.

[5] Fluxicon. Process mining in practice, http://processminingbook.com, 2018.