Interactive Querying over Collections of Process Models

María Salas-Urbano[1,2] †, Cristina Cabanillas[1,2] and Manuel Resinas[1,2]

[1]SCORE Lab, Universidad de Sevilla, Seville, Spain
[2]I3US Institute, Universidad de Sevilla, Seville, Spain

Abstract

Process mining analysts often need to compare multiple process models extracted from different process variants (e.g. product categories in a purchase-to-pay scenario) to detect specific behavioral patterns, such as delays or deviations. However, current process mining tools provide limited support for systematically managing collections of process models, which makes such analyses too manual and time-consuming. To address this, we have integrated LoVizQL, a query language for obtaining collections of Directly-Follows Graphs that meet specific, user-defined conditions, into an interactive query tool. Our tool enables users to formulate queries, generate multiple process model collections simultaneously and compare them to identify process variants of interest. We describe two concrete real-world scenarios to asses the utility of the proposed approach.

CCS Concepts

• ***Information systems*** → *Process mining; Query language;* • ***Human-centered computing*** → *Visualization systems and tools;*

1. Introduction

Process mining is an interdisciplinary field at the intersection of Business Process Management (BPM) and Data Science. It involves extracting insights from process execution data from event logs [VDA16]. An event log records activity executions within business processes, typically including attributes such as case identifiers, activity names, and timestamps [DFGMM18]. To analyze these logs, process mining techniques often rely on process models that visually represent process behavior. One of the most widely used types of process model is the Directly-Follows Graph (DFG) [VDA19]. A DFG is a graph in which each node represents an activity of the process, and each edge represents a directly-follows relation between activities [DFGMM18].

In practice, analysts often compare multiple *process variants* to identify behavioral patterns, such as delays, deviations, or structural differences [CASUCR22, KMW19]. A process variant is a subset of executions of a business process that can be distinguished by some characteristic (e.g. the executions of a process in a given country) [TLRDM21]. However, current process mining tools typically require analysts to repeatedly filter the event log, generate DFGs individually, and manually inspect them side by side to understand differences between process variants. The lack of support for managing and systematically comparing collections of DFGs makes this workflow time-consuming and involves many steps. We will illustrate this with an example.

EXAMPLE 1. *Consider a process mining analyst studying the purchase-to-pay process across several countries to identify systematic delays in invoice approvals. Using conventional tools, this requires filtering the event log by country, generating a DFG for each subset, and inspecting them individually. Focusing on specific frequent process variants further multiplies this effort, making the analysis tedious and time-consuming.*

To address this limitation, the Log Data Visualization Query Language (LoVizQL) was developed [SUCACR23]. LoVizQL is a query language that obtains *collections of DFGs* that satisfy user-defined conditions. The collection of DFGs corresponds to a set of process variants obtained by applying data manipulations to an event log. However, although LoVizQL defines the query language and its semantics, it does not support the interactive construction of queries. In this paper, we propose an interactive query tool that enables the incremental construction of LoVizQL queries, allowing analysts to generate and compare multiple collections of DFGs derived from process variants. The tool supports both independent analysis of individual collections and comparative analysis across collections. We demonstrate the utility of our approach through two case studies on real-world event logs, showing how the proposed query tool supports systematic exploration of collections of process models. The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 introduces LoVizQL. Sections 4 and 5 present the tool and system overview. Section 6 describes the evaluation scenarios, and Section 7 concludes the paper.

† This work is part of the R&D projects PID2021-126227NB-C21 (PERSEO) and PID2022-140221NB-I00 (TAPIOCA), funded by MICIU/AEI/10.13039/501100011033/ERDF/EU, and PID2024-156482NB-I00 (ISA), funded by MICIU/AEI/10.13039/501100011033 and ESF+. M. Salas-Urbano is supported by PREP2022-000372 financed by MICIU/AEI/10.13039/501100011033 and ESF+.

Name	Filter	Nodes	Metrics	%P	%A	Selection
`f1`		"concept:name"	Absolute Frequency			
`f2`	[Mandatory] `v1`<-"case:parts".*	"concept:name"	Absolute Frequency			`v2`<-*argmax*$_{v1}$(k=2) numberOfNodes(`f1, f2`)
`*f3`	[Mandatory] `v2`	"concept:name"	Absolute Frequency	0.1	0.1	

Table 1: *Example of query in LoVizQL.*

2. Related work

Query languages in the business process domain have been developed to facilitate the extraction of information from event logs [POBvdA17]. Several event log query languages focus on filtering cases, expressing control-flow constraints, or retrieving performance indicators. DAPOQ-LANG [GLdMRvdA16], PIQL [ÁDP*21], BPQL [MS21], and IQL [SMvZvdA22] define languages for formulating queries. Similarly, graph pattern search and motif discovery techniques [CF06, MSOI*02] have been proposed to identify recurring substructures in graphs. However, these approaches focus on defining query languages and pattern matching techniques, rather than on the interactive construction, management and comparison of collections of process models.

Several approaches support the comparison of process models through overlays, annotations, or side-by-side visualizations [WCZ*19, JJP*21, BdLvdA18]. These approaches facilitate the inspection of structural or performance differences between models, but primarily focus on visualization rather than the systematic construction and exploration of process model collections. VISCoPro [SUCACR26] supports the interactive creation and visualization of individual DFG collections derived from event logs. However, it does not support the interactive querying and comparison of multiple collections across exploration steps. Similarly, commercial tools [pro] such as Apromore, Celonis, Signavio, ARIS, and UiPath support process variant comparison, but do not provide mechanisms to define and compare collections of process models.

In the field of visual analytics, systems such as Zenvisage [SLK*17] and ShapeSearch [SLW*20] address exploratory analysis by generating visualizations that satisfy user-defined conditions. These systems allow users to specify visual patterns and automatically retrieve matching visualizations, enabling exploration of large collections without manual inspection. While these operate on traditional visualizations such as line charts, our approach focuses on process models. Moreover, instead of retrieving visualizations based on pattern similarity, our system enables users to define, store, and compare process model collections based on structural and performance properties. To the best of our knowledge, existing approaches do not support the combined querying and comparison of multiple process model collections in an interactive manner.

3. Background: LoVizQL query language

We now briefly describe LoVizQL [SUCACR23], the core of our interactive query tool. A detailed description of its syntax and semantics is provided in [SUCACR23]. LoVizQL enables analysts to obtain collections of DFGs that contain process insights without manually applying several manipulation actions and comparisons between visualizations. Queries are represented as tables, as depicted in Table 1. This table illustrates a query that identifies two process variants associated with the attribute *"case:parts"* that maximize the difference in the number of unique activities with respect to the overall process. The query is executed across three rows. The first query row (`f1`) generates the reference DFG of the complete event log. The second row (`f2`) groups the event log according to all possible values of the *"case:parts"* attribute, resulting in one process variant and its corresponding DFG per attribute value. Finally, the third row (`f3`) compares the DFGs generated in `f2` with the reference DFG `f1`. Since nodes correspond to activities in this example, the query measures the difference in the number of unique activities, and retrieves the two process variants that maximize this difference.

We now describe in detail the components of the query shown in Table 1. Each query row defines a collection of DFGs. The query fields are organized into four groups: the query identifier (*Name*), the data manipulation properties (*Filter*), the DFG creation (*Nodes*, *Metrics*, *%P* and *%A*), and the operation applied to the collection of DFGs to select those of interest (*Selection*).

LoVizQL supports sequential data manipulations to create specific collections of process variants. Each data manipulation, specified in the *Filter* field, produces either (i) a subset of the original event log or (ii) a collection of subsets. A subset results from applying a filtering condition (e.g., retaining cases with a duration greater than a specified threshold). In contrast, grouping operations can be applied to a specific attribute values or according to all values of that attribute. The example in Table 1 illustrates the latter case. In `f2`, the expression `v1 <- "case:parts".*` groups the event log by all values (represented as "*") of the attribute *"case:parts"*. As a result, the system automatically filters the event log into subsets based on each attribute value. Each generated subset corresponds to a process variant. In addition, the grouping mechanism is not limited to attribute filters. It can also be applied to filters based on process control-flow. For instance, using the endpoint filter and all values selection, the system automatically creates one subset for each observed combination of start and end activities. To support dependencies between query rows, LoVizQL incorporates variables to store elements, following the structure *"variable <- element"*. In LoVizQL, variables store the filtering or grouping criteria and can be reused in subsequent query rows. In Table 1, variable `v1` stores the values of the attribute *"case:parts"* generated during the grouping operation in row `f2`. These stored values are then reused in the selection step of row `f3`.

After creating the process variants, the *Nodes* and *Metrics* fields (cf. Table 1) define the properties of the corresponding DFGs. The *Nodes* field selects the event attribute for the DFG nodes (e.g., activity or role), while the *Metrics* field determines how the nodes and edges are annotated using frequency or performance measures,

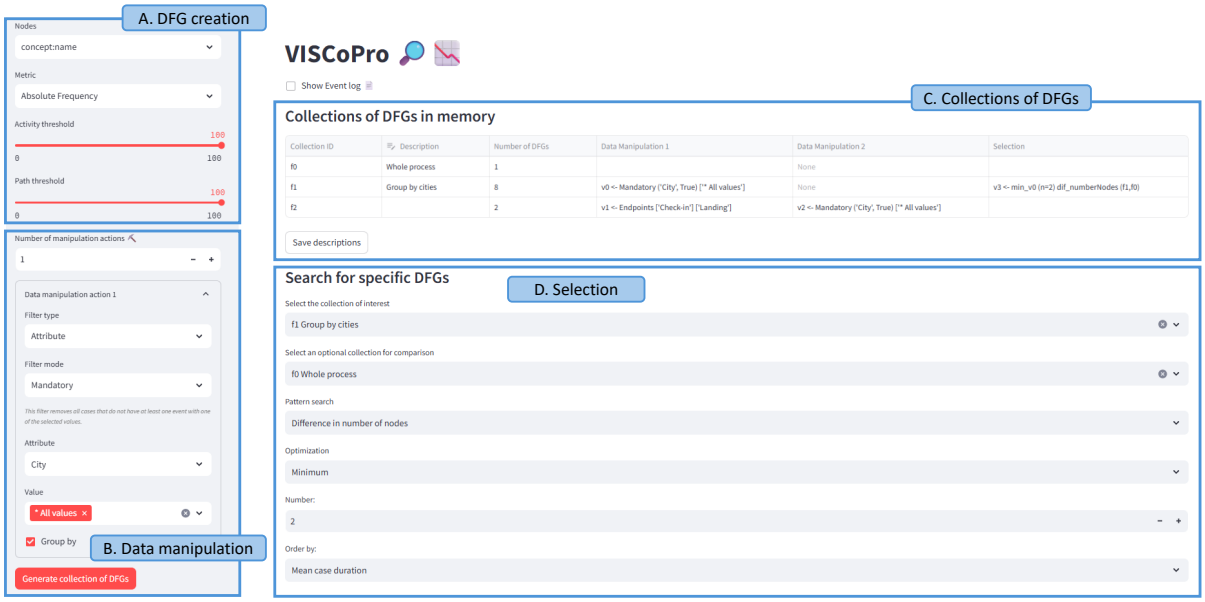

Figure 1: *Interactive query tool interface for DFG exploration: Overview of components.*

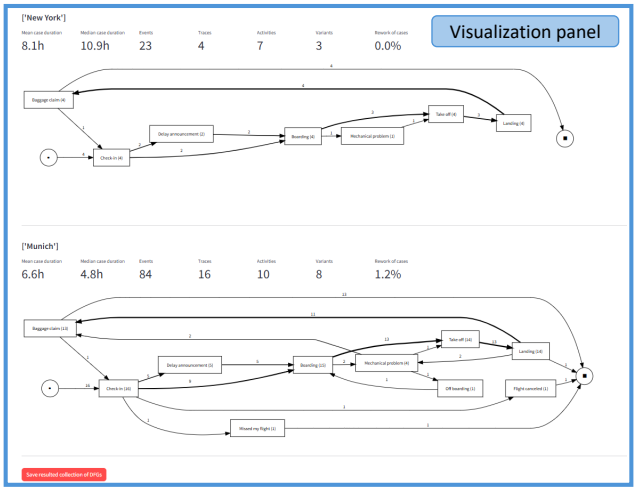

Figure 2: *DFGs of the two cities minimize the difference in the number of nodes compared to the DFG of the complete process.*

such as mean cycle time. Additionally, the *%P* and *%A* fields indicate the percentage of transitions and activities, respectively, included in each DFG of the collection.

Finally, the *Selection* field enables analysts to search for DFGs of interest within or across collections. Analysts first define a pattern, which defines a measurable property of the DFGs, such as the number of nodes, frequencies, rework, or performance metrics. These properties are currently restricted to a predefined set of available pattern functions supported by the system. Then, a selection function is applied to optimize this property by maximizing or minimizing its value within a collection or computing differences between two collections. The number of retrieved results is also specified, such as the top-*k* DFGs that best satisfy the defined condition. Table 1 illustrates this comparative search mechanism. In row `f3`, the expression `v2 <- argmax_v1(k=2) numberOfNodes(f1,f2)` iterates over the values stored in `v1`. For each process variant in `f2`, the system compares its DFG against the reference DFG generated in `f1` and computes the difference in the number of nodes. The values in `v1` that maximize this difference are stored in `v2`, enabling their reuse in subsequent query rows.

4. Interactive querying over collections of DFGs

This section describes an interactive query tool that uses LoVizQL to allow users to explore collections of DFGs. This tool is an extension of VISCoPro [SUCACR26]. Our contribution builds on this by providing a query-driven interaction model to systematically define, store, and compare multiple collections of DFGs across query rows. The tool is publicly available [visa]. Figure 1 shows the interface loaded with an event log.

As shown in Figure 1, the interface is built around the query table (cf. Figure 1-C). In LoVizQL, each query row defines a collection of DFGs. As the user interacts with the tool, this table is incrementally constructed: each configuration of data manipulation and DFG properties defines a new query row, and its output is stored as a collection. In this way, users do not need to explicitly write queries, since they are generated through interaction with the interface. This design allows analysts to automatically capture intermediate results, which they can then reuse in subsequent steps. This facilitates the systematically exploration and comparison DFG collections across query rows.

First, the user defines filtering and grouping operations over the

event log (cf. Figure 1-B), which determines the set of process variants included in the collection. Next, the user specifies how each process variant is represented as a DFG, including node attributes, performance metrics, and frequency-based filtering (cf. Figure 1-A). These two steps rely on existing VISCoPro functionalities. We further extend VISCoPro by enabling the use of LoVizQL variables within data manipulation steps. This allows filtering and grouping conditions to be stored and reused across queries.

Beyond the creation of individual collections, the tool allows analysts to query and analyze multiple stored collections (cf. Figure 1-D). To formulate a query, users first select one or more DFG collections, define a pattern function corresponding to a measurable DFG property (e.g., number of nodes, frequencies, rework, or performance metrics), and specify an optimization function for this property. Queries may also define the number of results to retrieve, such as the top-*k* DFGs satisfying the specified condition. When a single collection is selected, analysts can search for those DFGs that maximize or minimize a given property within this collection, such as the process variants with the longest execution times. When two collections are selected, one acts as the collection of interest and the other as the reference collection. The system then performs pairwise comparisons between DFGs and identifies those in the collection of interest that best meet the selected optimization function. For instance, analysts can identify the top-3 process variants whose DFGs differ the most from the reference collection in terms of number of nodes. This specification is added to the *Selection* field of the collection of interest and stored in a variable that can later be reused to define new DFG collections in subsequent query rows. The original functionalities of VISCoPro do not support querying and comparing across DFG collections.

Finally, the visualization panel (cf. Figure 2) uses existing VISCoPro functionalities to display multiple DFGs simultaneously. It visualizes the results of each query row, allowing analysts to inspect the generated DFG collections and avoid empty queries.

5. System overview

This section provides an overview of the architecture of our tool. It is implemented in Python and built using the Streamlit [str] framework to support interactive visual analytics. The source code is publicly available [visb].

The architecture is modular and consists of two main components: a front-end interface and a back-end processing layer. The front-end, which is implemented in Streamlit, manages user interactions, including actions such as data manipulation, DFG parameter selection, pattern definition, and result visualization. It provides dynamic components for filtering, grouping, optimization-based searches, and interactive comparisons of DFGs. The back-end is responsible for processing event logs, generating collections of process variants, constructing corresponding DFGs, and executing selection queries. We have used PM4PY [pm4] library, which provides several process mining functionalities. In addition to these capabilities, our system implements the necessary logic to manage DFG collections, handle variables, and perform comparisons across multiple DFGs according to patterns defined by the user. The DFG construction and comparison processes are executed dynamically based on stored variables. When a search is performed, the system iterates through the variable values, calculates the result of the specified pattern, and ranks the results accordingly.

6. Demonstration scenarios

Our scenarios illustrate how the proposed tool supports: (i) identifying relevant insights from DFG collections; (ii) building multi-step exploration workflows through grouping, filtering, and comparison; (iii) reusing variables across queries; and (iv) systematically comparing multiple DFGs.

6.1. Dataset

We will use a real-world event log from the Business Process Intelligence Challenge (BPIC), an annual contest in which an organization provides an event log and some business questions. Specifically, we will use the *International Declarations* event log from 2020 BPIC [vD20]. It includes 6,449 cases and 72,151 events related to the reimbursement of international travel expenses at Eindhoven University of Technology. In this process, employees submit reimbursement requests after taking international trips. Each request goes through a sequence of approval steps involving roles such as travel administrator, budget owner, and supervisor. The log records the activities performed, timestamps, organizational units, and role information for each event.

6.2. Scenario 1: Structural analysis

A common question in process mining is whether certain organizational units in a process lead to different complex process behavior. In the context of the International Declarations log, we address the following question: *Which departments generate the less structurally complex process variants in relation to the overall process?*

To answer this question, Figure 3 shows the query corresponding to this scenario. First, we generate a collection with only the reference DFG from the complete event log to represent the overall international reimbursement process (`f0`). Next, we create a collection of DFGs by grouping cases according to the presence of specific departments using the *"organizational:unit"* attribute (`f1`). Each DFG in the collection corresponds to one department and includes all cases corresponding to it. The set of departments considered is stored in variable `v0`, which will be used later. In total, it contains 27 values. In both queries (`f0` and `f1`), the DFGs are built using the activity attribute to define the nodes, and the absolute frequency is used as metric.

Then, we use the selection panel to search for specific DFGs. We define the number of nodes as the pattern function of interest and choose to maximize the difference in number of nodes between the two collections (`f1,f0`), i.e., between each DFG (`f1`) and the reference DFG (`f0`). The system iterates over the values stored in `v0` and computes the structural difference for each pair of corresponding DFGs. Additionally, we search for the top two departments that maximize this difference. Then, the tool returns the two DFGs whose structure deviates the most in terms of the number of activities involved. These results are visualized side by side, allowing analysts to immediately inspect structural differences. Finally,

Collection ID	Description	Number of DFGs	Data Manipulation 1	Selection
f0	Whole process	1		
f1	Group by department	27	v0 <- Mandatory ('case:Permit OrganizationalEntity', True) ['* All values']	v1 <- max_v0 (n=2) dif_numberNodes (f1,f0)
f2		2	v2 <- Mandatory ('case:Permit OrganizationalEntity', True) ['organizational unit 65478, organizational unit 65488']	

Figure 3: *A LoVizQL query returning the two departments whose process variants differ most from the complete process.*

Collection ID	Description	Number of DFGs	Data Manipulation 1	Data Manipulation 2	Selection
f0		8	v0 <- Endpoints ['Permit SUBMITTED by EMPLOYEE'] ['Payment Handled']	v1 <- Mandatory ('org:role', True) ['* All values']	v2 <- max_v1 (n=3) rework(f0)
f1		8	v3 <- Endpoints ['Permit SUBMITTED by EMPLOYEE'] ['Payment Handled']	v4 <- Mandatory ('org:role', True) ['* All values']	v5 <- max_v4 (n=3) transduration(f1)

Figure 4: *A LoVizQL query returning the roles with the highest rework and the longest activity durations.*

organizational unit 65488 and *organizational unit 65478* departments, which are the values associated with the selected DFGs, are stored in variable `v1` and can be reused for subsequent analyses. This allows for incremental exploration, enabling analysts to restrict further filtering or comparisons to the identified subgroups without redefining the original grouping conditions. For instance, in the last query (`f2`) we used `v1` to group the event log.

6.3. Scenario 2: Role analysis

A common challenge in reimbursement processes is identifying which roles are associated with corrections or revisions. In this context, we investigate the following question: *Which roles in the international reimbursement process involve activity rework? Do these roles contribute to time-consuming processes?*

To address these questions, Figure 4 shows the query corresponding it. In the first query row (`f0`) we restrict our analysis to complete process executions. We do this by retaining only the cases that start with *"Permit SUBMITTED by EMPLOYEE"* and end with *"Payment Handled"*. This ensures that only complete reimbursement process is considered. Next, we group the cases according to the presence of specific roles, where each group contains the cases in which a given role participates. Since a case may involve multiple roles, the same case can belong to more than one group. The set of role values is stored in variable `v1`. These two manipulation actions define the first collection of eight DFGs (`f0`), which uses the activity attribute for nodes and absolute frequency as metric for both nodes and edges. Then, we use the selection panel to define *"Identify rework"* as the pattern function and search for the top-3 roles involved in process variants with the highest amount of rework in their associated DFGs. The system iterates over the role values (`v0`) and computes the rework function for each DFG. The resulting DFGs allow analysts to identify *"Director"*, *"Pre-approver"* and *"Missing"* which are the roles most frequently involved in cases with repeated steps.

To better understand the behavior of these roles, we define a second query row (`f1`) using the same data manipulation actions as in `f0`, but using mean cycle time as the metric for nodes and edges. We aim to determine if cases with higher rework are associated with longer executions between activities. This helps us determine if rework correlates with slower reimbursement processes. In this case, we use the pattern function *"Identify activities by duration"* to search and retrieve the top-3 DFGs with the longest transitions between activities. The system iterates over the role values (`v4`) to search for those that meet this condition. Finally, `v5` stores the three role values whose associated DFGs contain the longest transitions between activities: *"Budget owner"*, *"Pre-approver"* and *"Missing"*. These results provide additional insight into the reimbursement process. The first query row identifies roles involved in process variants with high rework, while the second highlights roles associated with longer transitions between activities. The roles *Pre-approver"* and *Missing"* appear in both analyses, suggesting a relation with higher rework and execution times.

7. Conclusions

In this paper, we present an interactive query tool integrating LoVizQL to support the systematic exploration and comparison of multiple collections of process models. Our approach enables analysts to incrementally construct, store, and reuse results across exploration steps, facilitating interactive analysis workflows over multiple collections of DFGs. Since queries are generated through interaction, the tool simplifies query specification while still preserving the expressive capabilities of LoVizQL, following recent interaction-driven approaches for exploratory analysis [LLS*19]. However, our approach has some limitations. The expressiveness of the analysis is limited by the available pattern functions, and reliance on visual inspection may hinder scalability when handling large collections or complex DFGs.

As future work, we plan to extend the set of pattern functions to support more advanced forms of analysis, such as the identification of frequent activity sequences, representative behaviors across collections, and the detection of outliers. Additionally, we aim to explore how visual analytics techniques can facilitate these tasks by integrating automated pattern detection with interactive exploration. Finally, we plan to incorporate graph pattern search mechanisms to support more expressive comparisons.

References

[ÁDP*21] ÁLVAREZ J. M. P., DÍAZ A. C., PARODY L., QUINTERO A. M. R., GÓMEZ-LÓPEZ M. T.: Process instance query language and the

process querying framework. In *Process Querying Methods*. Springer, 2021, pp. 85–111. 2

[BdLvdA18] BOLT A., DE LEONI M., VAN DER AALST W. M.: Process variant comparison: using event logs to detect differences in behavior and business rules. *Inf. Syst. 74* (2018), 53–66. 2

[CASUCR22] CAPITÁN-AGUDO C., SALAS-URBANO M., CABANILLAS C., RESINAS M.: Analyzing how process mining reports answer time performance questions. In *International Conference on Business Process Management* (2022), Springer, pp. 234–250. 1

[CF06] CHAKRABARTI D., FALOUTSOS C.: Graph mining: Laws, generators, and algorithms. *ACM computing surveys (CSUR) 38*, 1 (2006), 2–es. 2

[DFGMM18] DI FRANCESCOMARINO C., GHIDINI C., MAGGI F., MILANI F.: Business process management. 1

[GLdMRvdA16] GONZÁLEZ LÓPEZ DE MURILLAS E., REIJERS H. A., VAN DER AALST W. M.: Everything you always wanted to know about your process, but did not know how to ask. In *International Conference on Business Process Management* (2016), Springer, pp. 296–309. 2

[JJP*21] JALALI A., JOHANNESSON P., PERJONS E., ASKFORS Y., KALLADJ A. R., SHEMEIKKA T., VÉG A.: dfgcompare: a library to support process variant analysis through Markov models. *BMC Med. Inform. Decis. Mak. 21*, 1 (2021), 356. 2

[KMW19] KLINKMÜLLER C., MÜLLER R., WEBER I.: Mining process mining practices: an exploratory characterization of information needs in process analytics. In *International Conference on Business Process Management* (2019), Springer, pp. 322–337. 1

[LLS*19] LEE D. J.-L., LEE J., SIDDIQUI T., KIM J., KARAHALIOS K., PARAMESWARAN A.: You can't always sketch what you want: Understanding sensemaking in visual query systems. *IEEE transactions on visualization and computer graphics 26*, 1 (2019), 1267–1277. 5

[MS21] MOMOTKO M., SUBIETA K.: Business process query language. In *Process Querying Methods*. Springer, 2021, pp. 345–376. 2

[MSOI*02] MILO R., SHEN-ORR S., ITZKOVITZ S., KASHTAN N., CHKLOVSKII D., ALON U.: Network motifs: simple building blocks of complex networks. *Science 298*, 5594 (2002), 824–827. 2

[pm4] Pm4py - process mining for python. https://processintelligence.solutions/pm4py. 4

[POBvdA17] POLYVYANYY A., OUYANG C., BARROS A., VAN DER AALST W. M.: Process querying: Enabling business intelligence through query-based process analytics. *Decision Support Systems 100* (2017), 41–56. 2

[pro] Process mining software. https://www.processmining-software.com/. 2

[SLK*17] SIDDIQUI T., LEE J., KIM A., XUE E., YU X., ZOU S., GUO L., LIU C., WANG C., KARAHALIOS K., ET AL.: Fast-forwarding to desired visualizations with zenvisage. In *CIDR* (2017). 2

[SLW*20] SIDDIQUI T., LUH P., WANG Z., KARAHALIOS K., PARAMESWARAN A.: Shapesearch: A flexible and efficient system for shape-based exploration of trendlines. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (2020), pp. 51–65. 2

[SMvZvdA22] SCHUSTER D., MARTINI M., VAN ZELST S. J., VAN DER AALST W. M.: Control-flow-based querying of process executions from partially ordered event data. In *International Conference on Service-Oriented Computing* (2022), Springer, pp. 19–35. 2

[str] Streamlit. https://streamlit.io/. 4

[SUCACR23] SALAS-URBANO M., CAPITÁN-AGUDO C., CABANILLAS C., RESINAS M.: Lovizql: a query language for visualizing and analyzing business processes from event logs. In *International Conference on Service-Oriented Computing* (2023), Springer, pp. 13–28. 1, 2

[SUCACR26] SALAS-URBANO M., CAPITÁN-AGUDO C., CABANILLAS C., RESINAS M.: Interactive creation, visualization, and exploration of process model collections. *Software and Systems Modeling* (2026), 1–26. 2, 3

[TLRDM21] TAYMOURI F., LA ROSA M., DUMAS M., MAGGI F. M.: Business process variant analysis: Survey and classification. *Knowledge-Based Systems 211* (2021), 106557. 1

[vD20] VAN DONGEN B.: BPI Challenge 2020. 4TU.ResearchData, Mar 2020. doi:10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b51. 4

[VDA16] VAN DER AALST W.: Data science in action. In *Process mining: Data science in action*. Springer, 2016, pp. 3–23. 1

[VDA19] VAN DER AALST W. M.: A practitioner's guide to process mining: Limitations of the directly-follows graph, 2019. 1

[visa] Interactive query tool. https://interactivequerytool.streamlit.app/. 3

[visb] Interactive query tool (github repository). https://github.com/msurbano/interactive-query-tool. 4

[WCZ*19] WANG J., CAO B., ZHENG X., TAN D., FAN J.: Detecting Difference Between Process Models Using Edge Network. *IEEE Access 7* (2019), 142916–142925. 2