A Query Language for Exploring Directly-Follows Graph Collections*

María Salas-Urbano[1][0000−0003−0620−1615], Carlos Capitán-Agudo[1][0000−0003−2772−1740], Cristina Cabanillas[1,2][0000−0001−9182−8847], and Manuel Resinas[1,2][0000−0003−1575−406X]

[1] SCORE Lab, Universidad de Sevilla, Spain
[2] I3US Institute, Universidad de Sevilla, Spain
{msurbano,ccagudo,cristinacabanillas,resinas}@us.es

Abstract. Visualization tools are very useful for data exploration and offer a very intuitive way to look at interesting results in data analysis. In previous work, systems have been designed that, starting from a dataset, generate and work on sets of data visualizations and find those that show desired trends automatically, thus avoiding manual exploration of many visualizations by the user. One of the most used mechanisms to obtain interesting visualizations is the use of a query-based language. However, the use of these systems in process mining is not contemplated, which would be very useful to specifically find interesting results among multiple Directly-Follows Graphs (DFGs) extracted from event logs without carrying out the typical manipulation tasks and exploring multiple DFGs. We are interested in extending an existing query-based language, adapting it to process mining. The goal of is to automatically generate DFG collections and search for visualizations that contain patterns of interest according to some queries made by the users. Thus, interesting visualizations can be found by obtaining and comparing properties of sets of DFGs generated by the system. As an advantage, this approach allows users to obtain interesting results without the need to carry out a manual exploration of a large number of visualizations using the existing process mining tools. We have carried out the evaluation of our approach by solving a challenge provided in a BPI Challenge.

Keywords: directly-follows graphs · visualizations · process mining · data exploration · queries

1 Introduction

Process mining is used to extract knowledge from event logs available in information systems and visualize the processes. Currently there are multiple tools to perform process mining, which have specific functions for the different phases

* Work funded by grants RTI2018-100763-J-I00 and RTI2018-101204-B-C22 funded by MCIN/ AEI/ 10.13039/501100011033/ and ERDF A way of making Europe; and grant P18-FR-2895 funded by Junta de Andalucía/FEDER, UE.

of the analysis. One of the most important aspects of process mining tools is the use of Directly-Follows Graphs (DFGs) as visualizations of the processes, since this type of data visualization is a very intuitive way to uncover process insights in the data exploration and process discovery phases. A DFG (also called process map) of an event log is a graph where each node represents an activity of the process, and each arc represents a relation between nodes (activities). For instance, an arc from node A to node B means that activity B is directly followed by activity A in at least one trace in the event log, that is, one execution of the process [4]. An example of a DFG is shown in Figure 1. In this example each node and each arc is annotated with a value representing, respectively, how many times a node occurs and how many times a node directly follows another node. In this case, these values correspond to the absolute frequency, and as we can see the *Declaration APPROVED BY ADMINISTRATION* activity occurs 23 times and it is directly followed by the *Declaration REJECTED by EMPLOYEE* activity 13 times. In addition to the absolute frequency, there are other options to show in a DFG like: (1) the relative frequency, that is, the percentage of times that B directly follows A, when A is executed; and (2) the performing time, that is, the average time between the occurrence of A and the occurrence of B [4]. DFG is one of the most used visualization types to analyze processes. However, the existing process mining tools, such as Disco or Celonis, require a certain effort on the part of the user to be able to extract interesting knowledge about the process. In addition to getting misleading results [1], users must handle large datasets, perform several manipulation tasks like filtering and grouping, and evaluate whether the information obtained in the multiple DFGs obtained is interesting. Moreover, comparing different visualizations is a complex task for users since existing process mining tools do not facilitate the generation of several DFGs at the same time.

In this work, we are interested in extending an existing query-based language [8], adapting it to process mining. The goal of this approach is to use queries defined by the user to automatically generate DFG collections and search for the visualizations that contain results that are interesting to the user. These queries provide the mechanism to the users to carry out actions on visualization collections and discover interesting insights in the process, such as discovering the variant of the process that has the lowest coefficient of connectivity or complexity (that is, the highest number of arcs and the lowest number of nodes). There are many properties that can be extracted from a DFG. Many of them are discussed in [13]. In our approach, we can use these properties to measure some features of DFGs and compare them. For instance, process analysts are often interested in discovering the variants of the process that meet some requirement, the simplest or the most complex variants, in terms of number of nodes and edges. With this approach these discoveries can be made without having to manually explore the DFG of each variant and compare them.

Finally, we have used a challenge from a BPI Challenge (BPIC for short) to evaluate our system. BPIC are annual contests where some questions are asked about a business process and the participants must solve these challenges. In this

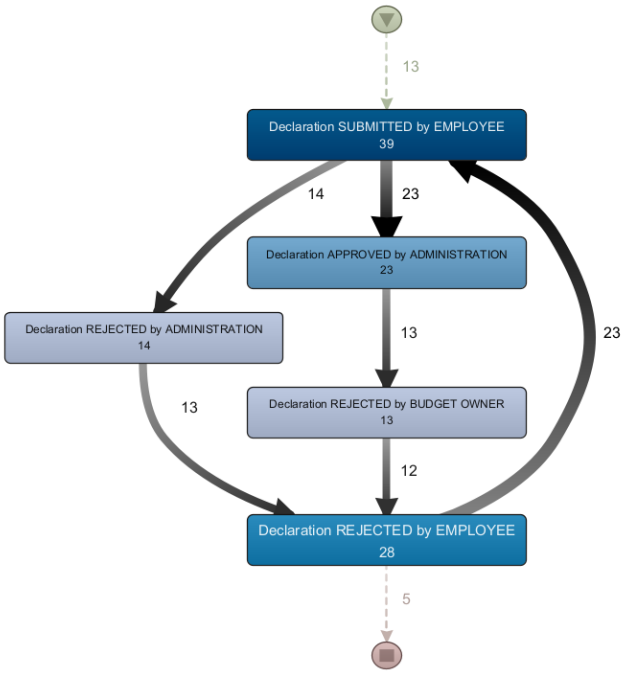

Fig. 1: Example of DFG

case, we have used one question belonging to BPIC 2016[1] and we have selected the report of one of the winners [2] in order to use our approach to get similar results as these participants. As a result, we automate the steps performed by the authors and obtain the patterns of interest efficiently.

The paper is structured as follows. Section 2 outlines the literature related to this work. Section 3 describes the query-based language that our approach is based on. Section 4 describes the system proposed in this research. Section 5 provides details of the evaluation of our system using a real example. Section 6 summarizes the conclusions drawn, limitations and future work.

2 Related Work

There are various considerations to take into account when proposing a visualization recommendation system, such as the characteristics of the data, the objective and the user's preferences, the domain, and the ease of understanding the results [9]. Some tools, such as DeepEye [5] recommends good visualizations of datasets without requiring the user to have knowledge of the shape and structure of the data. The tool uses a binary classifier to predict whether a visualization is good, as well as machine learning techniques and insights from expert

[1] BPI Challenge 2016: https://www.win.tue.nl/bpi/doku.php?id=2016:challenge

rules to determine, given two visualizations, which one is better. On the other hand, VizRec [6] proposes a hybrid method that combines information about how good a visualization is, and the content of the visualization itself. This tool recommends personalized visualizations based on rules and taking into account user preferences based on queries, that is, to recommend those visualizations that the users would select as part of their analysis.

Similarly, both [10] and [11] are also automatic systems that generate visualizations based on queries made by the users. On the one hand, [10] recommends visualizations in which there are deviations in relation to a reference, which qualify as interesting visualizations given a subset of data. However, they do not cover other visualizations of interest, such as visualizations similar to a reference or considering user preferences. On the other hand, [11] takes into account the preferences of interest of the user. Users have to introduce the dataset and they can select which variables or attributes are relevant in the study. Another research [3] creates a model that automatically generates data visualizations using sequence-to-sequence recurrent neural networks, using rules learned from examples. For training this tool, a large training set is necessary, which determines the output of the network. The system does not include user preferences. For all the systems described, there is a great diversity of possible graphical representations, but none of these approaches contemplate the study of process data and they do not use DFGs as visualizations.

In contrast, Seelinger et al. [7] have developed an unsupervised visual recommender system for process analysis. This tool takes an event log as the input and automatically analyzes the dataset to provide interesting visualization recommendations. It is based on grouping the log into subsets of traces and they search for those which are significantly different to the overall dataset. This tool shows visualizations of these subsets. We propose an alternative approach to this research by allowing the user to indicate his goal through the query-language by the user and basing these queries on collections of DFGs.

The closest work to ours and on which we have based our approach, is Zenvisage [8]. It is a visual analytical system capable of automatically finding desired visual patterns with respect to a reference or behaviour patterns. Its objective is to avoid situations in which thousands of graphs have to be manually reviewed to check the conditions with which the desired visual pattern is obtained. The focus is put on several types of visual representations, such as bar charts, histograms or scatter plots. We complement this research using as input an event log and DFGs as visualizations which are not considered in Zenvisage because they are not interested in representing processes.

3 Background on ZQL queries

We have been inspired by the Zenvisage tool [8] to develop our approach. Its goal is to obtain desired visual patterns (e.g. similar or different visualizations with respect to a reference, typical or anomalous behaviour) using the Zenvisage Query Language (ZQL). The result of each query is always a visualization col-

lection that meets a number of conditions and it is usually used in subsequent queries. They define the queries with 3 sets of fields.

1. *Name*. It is the identifier for the visualization collection.
2. *Visualization Collection (X, Y, Z, Viz)*. It contains several fields with which to define a set of visualizations, i.e. the attributes of the x and y axes (X and Y), the subset of data being displayed (Z), and the type of plot (Viz). With Z, users can specify which event log attribute to filter by, as well as which values of that attribute. The notation followed is <attribute>.<attribute value or values>. It is possible to perform a filter for each value of an attribute, by using the abbreviation *, as shown in Table 1. Furthermore, users can choose to add other fields to Z in order to filter traces by more than one attribute. Thus, it is possible to have Z1, Z2, Z3, etc. in the queries. In relation to Viz, it is frequent to leave this field empty due to the fact that the system is able to determine what type of plot is better based on the X and Y attributes.
3. *Process*. This is an operation to apply on the visualization collection, such as filters, comparisons and ordering, based on some condition of similarity or dissimilarity, typical or anomalous behavior. The basic structure of the Process field is as follows: $<optimization>_{variable}<condition>$ $<function>$, where $<optimization>$ refers to the optimization function used and can be *argmin*, *argmax* or *argany* if the user wants to obtain minimum, maximum or any value that meets the condition, respectively. In addition, the *variable* parameter refers to the variable that contains the set of values to be optimized. The $<condition>$ limits the number of possible outcomes: k=N returns the top-k values, and k>R or k<R return values which are greater than or lesser than a threshold value k. Finally, $<function>$ refers to the type of function used (T(f) or D(f,f')):
   (a) T(f) takes the display f and returns a real value that measures some property of f. This operation is used on collections of visualizations, to find maximum or minimum values, or any value that satisfies the specified condition.
   (b) D(f,f'). Likewise, this function is used on collections of visualizations or sets of values. D(f,f') takes two visualizations (f,f') and measures the distance or dissimilarity between them using distance functions such as the Euclidean distance or the Kullback-Leibler divergence.

An example of use of this tool is shown in Table 1. This query returns the sales visualizations for all products which have a negative trend. First, f_1 returns a set of visualizations showing the sales (Y) per year (X) for each product (Z). The operation defined in the Process field is applied to this set of visualizations, through which the visualizations are filtered based on an overall trend. Specifically, products whose visualizations show a negative trend (t<0) are filtered and stored in the variable v2. Next, f_2 represents the sales visualizations by year for these filtered products (v2).

Table 1: Zenvisage query example

Name	**X**	**Y**	**Z**	**Process**
f_1	'year'	'sales'	v1 <- 'product'.*	v2 <- argmax_{v1}[t<0]T(f_1)
*f_2	'year'	'sales'	v2	

4 Our proposal

In this paper, we propose an approach which automates the search for interesting aspects in the process mining process from queries made by the user. In this section we introduce the query-based language and how it works.

4.1 Query structure

We have extended the ZQL language to make it best suited to manipulate the data of the event logs. It allows performing operations on the results of each query. For example, the system allows us to filter the traces of an event log by each value of a given attribute. Then, it is possible to calculate some property on these subsets of traces and compare these measures.

First, we define the structure and each field established for any query. The structure of the queries that we have defined is different from the structure used in ZQL queries. This is because our system receives an event log as input, unlike Zenvisage [8] which receives a common dataset. Event logs have a common structure, which is characterized by including 3 attributes: case id, activity and timestamp. Each process instance has a different case id, with which all activity instances can be related to a specific process instance. Moreover, for each activity instance it is known when it was executed, based on its timestamp [4]. Each query is defined by the following fields:

1. *Name.* As in ZQL language, it identifies the visualization collection.
2. *%Activity and %Path.* Using these percentages, it is possible to filter the number of activities and transitions between them displayed in a DFG based on their frequency.
3. *Filter type.* The user can choose the mode they want to use with the type of filter. There are 3 modes available and they are explained in Section 4.1.1.
4. *Z.* The user selects which event log attribute they want to filter by, following the same approach as ZQL queries.
5. *X.* This field is used to determine what metric the user wants to use and show in the DFG. As we mentioned in Section 1, there are typically 3 options to show: absolute frequency, relative frequency, and performing time.
6. *Process.* Users can filter results or visualizations based on a trend or search for similar or different results using this field. The are two different types of operations that users can carry out (T(f) and D(f,f')), following the same approach as in the ZQL queries (cf. Section 3).

4.1.1 Parameters of the filter: Filter type and Z field. As we have mentioned before, users can select the type of filter that they wish to apply to the data, in the same way that it is selected in most process mining tools like Disco or Celonis. Thus, we have defined 3 filtering modes based on the existing types of filters in Disco.

1. Mandatory, if the user wants to filter the traces that contain one or several values of some attribute. That is, this filter maintains the traces that present the selected values (Z field) in the filtered dataset.
2. Keep selected, if the user wants to remove events that do not have one of the selected values.
3. Forbidden, if the user wants to determine which values of some attribute must not occur to keep some case in the filtered dataset. This mode is opposite to the mandatory mode.

Then, users must specify in the *Filter type* field what type of filter among these 3 they want to carry out. In addition to the type of filter, users can specify the subset of traces they want to use in *Z*.

4.1.2 Process As we mentioned above, in this field we can define some operation to filter results. We have defined the syntax of these operations in the same way as it is done in Zenvisage [8].

Zandkarimi et al [13] describe several event log measures. These can be obtained directly from event logs, such as calculating the number of events or the average trace length, or they can be also obtained from DFGs, such as calculating the number of nodes or its density. All these event log measures can be used as T(f) in the Process field. So, users can define conditions about the number of traces, the number of nodes, or the cyclicity of a DFG collection.

In relation to D(f,f'), it is possible to find differences between these properties using it. For instance, to find the DFGs with the smallest difference between their cyclicity and their density. In addition, users can use other measures of distance between graphs, such as Levenstein's edit distance, isomorphism, maximum common subgraph or minimum common supergraph [12].

As a result of the Process field, some examples are $argmax_{v1}$[k=3](f_1) to find the value of a set of results v_1 which maximizes T(f_1); or $argmin_{v1}$[k=3]F(f_1) to find the three values of v_1 that minimize T(f_1). Moreover, we can compare DFG properties. So, we can find the DFG with the greatest difference between its number of nodes and edges, using the following function: $argmax_{v1}$[k=1]D(f_1,f_2).

4.2 System operation

In relation to the operation of the system, it works through queries as previously mentioned. We can see an example of queries in Table 2. The goal of these queries is to discover the activity which is contained in all traces of a subset of traces that has the lowest coefficient of connectivity or complexity (i.e., the highest number of arcs and the lowest number of nodes).

Table 2: Query example

Name	**%A**	**%P**	**Filter type**	**Z**	**X**	**Process**
f_1	100	100	Mandatory	v1<-'activity'.*	Absolute frequency	v2<-argmin$_{v1}$[k=1] CoefficientConnectivity(f_1)
*f_2	100	100	Mandatory	v2	Absolute frequency	

At the beginning, with the first query (f_1) the following is done: each activity is stored in the variable v1 and the traces that contain the activity v1 are filtered for each case, so that multiple subsets of traces are obtained. Then, the DFG for each subset of traces is obtained with 100% of activities and 100 %paths. As a result of f_1, a visualization collection is obtained, that is, one DFG for each filtered subset of traces (for each activity). Then, the operation described in Process is applied on this collection. Specifically, the coefficient of connectivity for each of these DFG is calculated and the system looks for that DFG that minimizes said function, and therefore the activity by which it has been filtered to obtain said subset of traces and its DFG. This activity is stored in v2. Finally in the f_2 query, the DFG showing the absolute frequency values is displayed for the traces that contain the previously identified activity (v2).

5 Evaluation

We have selected one question from the BPIC 2016. In this challenge, the event log is provided by an employee insurance agency, which is interested in learning about how its channels are used, when customers move from one contact channel to the next and why, and if there are profiles of clear customers to identify in behavioral data. Then, the organization proposes the following question: *Are there clear distinct usage patterns of the website to be recognized? In particular, insights into the way various customer demographics use the website and the Werkmap pages of the website are of interest.* We have chosen this question due to the fact that it is a representative example, since its purpose is the search for interesting insights in the process. Moreover, we have selected the analysis carried out by one of the winners of this year [2], and we have identified the actions performed by the analysts to provide the answer and how they carry out these actions using current process mining tools from the descriptions in the report submitted to the challenge. Then, we have created the corresponding queries using our approach, in order to obtain similar results.

To do this, in the selected answer [2], the authors performed the followings steps manually:

1. Separation of the dataset into segments according to the *AgeCategory* log attribute. To do this, the authors segment the event log into 4 datasets based on the possible values that the AgeCategory attribute takes.

Table 3: Challenge 1 - BPI Challenge 2016

Name	**%A**	**%P**	**Filter type**	**Z**	**X**	**Process**
f_1	0	99	Mandatory	v1<- 'AgeCategory'.*	Relative frequency	v2<-argmin$_{v1}$[k=1] Cyclicity(f_1)
f_2	0	99	Mandatory	v2		

2. Import of segmented data into Disco.
3. Extraction of most frequent events from segments. The authors filter the activities whose relative frequency is equal to or greater than 1% for each segment.
4. Analysis of reduced event sets for each segment. They generate 4 DFGs, one for each grouped data subset.
5. Derivation of usage patterns from the DFG of each segment. They compare the DFGs based on the following features: start activities, the flow of activities, and unique activities.

We show in Table 3 the resolution of this same challenge using our approach based on our query language. In the first (f_1) query, each value of *AgeCategory* attribute is stored in the variable v1 and we obtain subsets of traces that contain each of the activities. Next, the DFG is obtained for each subset of traces, taking into account that the authors are only interested in those activities whose relative frequency is greater than 1%. In the query, these requirements are specified in the %A and X fields (cf. Table 3). As we can see, with the generation of this visualization collection, we carry out the first 4 steps carried out by the authors of the selected report. In this case, the advantage of our approach is the easy handling of subsets of traces and the creation of DFGs, without the need to import and manipulate each of these data in some tool (Disco in this case). Finally, to perform their step 5 about the usage patterns from the DFG of each segment, we can add different operations to the Process field. One of the findings made by the analysts is the identification of the DFG with the lowest number of cycles. This manual task of comparing DFGs is done more quickly using our approach, and to do this, we have added the respective operation to Process, as we can see in Table 3. In this way, any other function to compare these DFG can be defined in Process. Finally, the DFG of interest found by the system is displayed through the query f_2.

6 Conclusions

In this work we present our approach, which is an extension of the ZQL language for event logs, which makes it easy to identify interesting insights in DFGs, automating data manipulation and analysis tasks. However, it requires certain knowledge by the users, since they must define the queries in the system and guide the search for interesting results. Another limitation of the system is the

types of operations and functions that can be carried out in the *Process* field of the queries. Since it is work in progress, this functionality should be extended by adding new possible functions. Moreover, our goal is to create an automatic tool that also helps the user with the development of the queries, offering some recommendations for it.

As future work, we plan to develop the tool and test it. A possible option could be to perform the evaluation with analysts who have participated in the BPI Challenges, so that they could replicate their analysis using the tool and evaluate the results obtained in relation to the analysis previously done using the specific measures for visualizations described in Section 1, such as relevance, surprise and non-obviousness.

References

1. van der Aalst, W.M.P.: A practitioner's guide to process mining: Limitations of the directly-follows graph. Procedia Computer Science **164**, 321–328 (2019)
2. Dadashnia, S., Niesen, T., Hake, P., Fettke, P., Mehdiyev, N., Evermann, J.: Identification of distinct usage patterns and prediction of customer behavior. BPI Challenge (2016)
3. Dibia, V., Demiralp, Ç.: Data2vis: Automatic generation of data visualizations using sequence-to-sequence recurrent neural networks. IEEE computer graphics and applications **39**(5), 33–46 (2019)
4. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A., et al.: Fundamentals of business process management. Second Edition. Springer (2018)
5. Luo, Y., Qin, X., Tang, N., Li, G.: Deepeye: Towards automatic data visualization. In: International Conference on Data Engineering (ICDE). pp. 101–112 (2018)
6. Mutlu, B., Veas, E., Trattner, C.: Vizrec: Recommending personalized visualizations. ACM Transactions on Interactive Intelligent Systems (TiiS) **6**(4), 1–39 (2016)
7. Seeliger, A., Nolle, T., Mühlhäuser, M.: Process explorer: an interactive visual recommendation system for process mining. In: KDD Workshop on Interactive Data Exploration and Analytics (2018)
8. Siddiqui, T., Kim, A., Lee, J., Karahalios, K., Parameswaran, A.: Effortless data exploration with zenvisage: an expressive and interactive visual analytics system. arXiv preprint arXiv:1604.03583 (2016)
9. Vartak, M., Huang, S., Siddiqui, T., Madden, S., Parameswaran, A.: Towards visualization recommendation systems. ACM Sigmod Record **45**(4), 34–39 (2017)
10. Vartak, M., Rahman, S., Madden, S., Parameswaran, A., Polyzotis, N.: Seedb: Efficient data-driven visualization recommendations to support visual analytics. In: International Conference on Very Large Data Bases (VLDB). vol. 8, p. 2182 (2015)
11. Wongsuphasawat, K., Qu, Z., Moritz, D., Chang, R., Ouk, F., Anand, A., Mackinlay, J., Howe, B., Heer, J.: Voyager 2: Augmenting visual analysis with partial view specifications. In: Chi conference on human factors in computing systems. pp. 2648–2659 (2017)
12. Zager, L.A., Verghese, G.C.: Graph similarity scoring and matching. Applied Mathematics Letters **21**(1), 86–94 (2008)
13. Zandkarimi, F., Rehse, J.R.: Fig4pm: A library for calculating event log measures (extended abstract). In: International Conference on Process Mining (2021)