Universität Stuttgart

Provenance-based visual data exploration

Von der Fakultät für Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Houssem Ben Lahmar

aus Oman

Hauptberichter: Prof. Dr. Melanie Herschel

Mitberichter: Prof. Dr. Carsten Binnig

Tag der mündlichen Prüfung: 22.01.2021

Institut für Parallele und Verteilte Systeme

2021

Contents

Abstract

Visual data exploration helps users in finding interesting information in data sets when they do not know beforehand what useful information hides in their data. It thus supports humans in understanding and interpreting data in an investigative way. Typically, visual data exploration systems assume that users have the prerequisite knowledge to issue exploration queries and to construct suitable visualizations to render their results. Nevertheless, these tasks are usually time-consuming, making manual visual data exploration a tedious and time-consuming process. Clearly, this is not convenient in real life scenarios where users often have limited time for visual data exploration.

To improve the efficiency of the visual data exploration process, this dissertation presents a visual data exploration framework that supports users based on recommendations throughout the whole exploration process. The framework leverages provenance, a term generally used to designate metadata that describes the process that leads to some data. In our setting, we capture provenance in order to document and annotate different stages of the visual data exploration process. The collected provenance is then leveraged to assist users during the visual data exploration.

The contributions of this thesis are summarized as follows. We first propose a new *evolution provenance* model that captures all important aspects related to the visual data exploration process such as users' exploration queries, users' interactions and visual encoding parameters of corresponding visualizations.

Based on this model and on another type of established provenance, we propose various novel *recommendation approaches* that assist users in querying and visualizing data. These approaches produce various types of recommendations that cover the whole data space of the explored dataset and that enable users to inspect readily rendered results.

Using our provenance-based recommendation approaches, users continuously receive sets of recommendations at the different stages of the visual data exploration process. Accordingly, we contribute a *quantification approach* that assesses the interestingness of each recommendation. The rationale behind that is to guide the user to select most interesting recommendations worth inspecting next.

All these approaches are integrated in our provenance-based framework that provides a holistic approach to support users in all stages of the visual data exploration process. We evaluate these approaches both quantitatively and qualitatively, demonstrating that our solutions improve the visual data exploration process compared to the state-of-the art and are effective in supporting users to make interesting findings during exploration.

While the previous contributions focus on using provenance for visual data exploration, we noticed during our research that there is a more general research question of analyzing sets of provenance traces. To pave the way for follow-up research, we therefore propose a *provenance aggregation approach* that summarizes provenance traces and discuss possible visual analytics tasks that may be applied on this type of summary.

Keywords: Visual data exploration, provenance, query recommendation, visualization recommendation, provenance aggregation

Zusammenfassung

Visuelle Datenexploration hilft Nutzern, interessante Informationen in Datensätzen zu identifizieren, die ihnen weitgehend unbekannt sind. Damit werden Nutzer beim Verständnis und bei der Interpretation von Daten auf investigative Weise unterstützt. Typischerweise gehen Systeme zur visuellen Datenexploration davon aus, dass deren Nutzer ausreichend qualifiziert sind, um explorative Anfragen zu formulieren und geeignete Visualisierungen für die Anfrageergebnisse zu konstruieren. Dieser manuelle Prozess ist oftmals zeitaufwändig. In realen Szenarien, in denen Nutzer oft nur wenig Zeit für die visuelle Erforschung großer Datensätze haben, ist dies nicht zweckmäßig.

Um die Effizienz visueller Datenexploration zu verbessern, stellt diese Dissertation ein Rahmenwerk für interaktive visuelle Datenexploration vor. Es verfolgt einen ganzheitlichen Ansatz zur Unterstützung der Nutzer im gesamten Explorationsprozess, indem diesen Empfehlungen für Anfragen und deren Ergebnisvisualisierung unterbreitet werden. Hierbei setzen wir auf die Verwertung sogenannter Provenance. Im Allgemeinen versteht man unter Provenance Metadaten, die den Prozess zur Erlangung von Daten beschreiben. In unserem Kontext wird Provenance erfasst, die die verschiedenen Phasen des visuellen Datenexplorationsprozesses dokumentiert. Diese Provenance wird verwendet, um Empfehlungen für Nutzer zu generieren.

Die wissenschaftlichen Beiträge dieser Dissertation lassen sich wie folgt zusammenfassen. Zunächst schlagen wir ein neues *Evolution-Provenance* Modell vor, das alle wichtigen Aspekte im Zusammenhang mit dem visuellen Datenexplorationsprozess erfasst, wie z.B. die Anfragen, die ein Nutzer im laufenden Prozess bereits analysiert hat, Interaktionen der Nutzer und visuelle Kodierungsparameter der entsprechenden Visualisierungen.

Auf der Grundlage dieses Modells und einer anderen Art von Provenance schlagen wir verschiedene neuartige *Empfehlungsansätze* vor, die den Anwendern bei der Abfrage und Visualisierung von Daten helfen. Diese Ansätze erzeugen verschiedene Arten von Empfehlungen, die den gesamten Datenraum des untersuchten Datensatzes abdecken und die es den Nutzern ermöglichen, leicht gerenderte Ergebnisse einzusehen.

Unter Verwendung unserer Empfehlungsansätze erhalten Nutzer in den verschiedenen Phasen des visuellen Datenexplorationsprozesses mehrere Alternativen empfohlen. Dementsprechend steuern wir einen *Ansatz zur Quantifizierung der Interessantheit* einer Empfehlung bei. Der Grundgedanke dahinter ist, den Nutzer dazu anzuleiten, die Empfehlungen auszuwählen, die es wert sind, als nächstes untersucht zu werden.

All diese neuartigen Ansätze werden durch unser auf Provenance basierendes Framework integriert, das einen ganzheitlichen Ansatz zur Unterstützung der Nutzer in allen Phasen des visuellen Datenexplorationsprozesses bietet. Eine quantitative und qualitative Evaluation belegt, dass unsere Beiträge den Stand der Forschung verbessern und Nutzer effektiv bei der visuellen Datenexploration unterstützen.

Während die bisher genannten Beiträge auf die Verwendung von Provenance für visuelle Datenexploration fokusieren, stellten wir während unserer Forschung fest, dass die Analyse einer Menge von Provenance Datensätzen eine allgemeinere Forschungsfrage darstellt. Aus diesem Grund schlagen wir einen Ansatz der *Provenance-Aggregation* vor, der mehrere Provenance Datensätze zusammenfasst und erörtern mögliche visuelle Analyseaufgaben, die auf diese Art von Zusammenfassung angewendet werden können.

Schlagwörter: Visuelle Datenexploration, Provenance, Empfehlungsalgorithmen, Provenance-Aggregation

Acknowledgements

This thesis has been an enriching and tough journey during it I was surrounded by many supportive people, to whom I would like to express my gratitude. First and foremost, I am indebted to Prof. Dr. Melanie Herschel who advised, supported me during those four years. Her constant support and guidance have been prominent throughout this journey, and I am extremely grateful for that.

I am thankful to my former colleagues from the Data Engineering group at the University of Stuttgart for their valuable support, technical discussions, and making the time here pleasant. I owe my warmest affection to all the members of the computer science department of University of Stuttgart. Especially to Eva for her kind help and assistance.

I want also to thank the German Research Foundation (DFG) for funding my research within the project "Quantitative methods for visual computing" (SFB-TRR 161). I express my gratitude to all the project members for their feedback and discussions made during our collaboration.

Finally, I would like to acknowledge my family for their support. I thank my mother Halima and my father Ali, who were always there to provide unconditional support. I also thank my sister Imen and my brother Karim for their support and encouragement.

Chapter 1

Introduction

1.1. Research context and motivation

With the increasing number and complexity of data processing applications, it is crucial to provide techniques to analyze and to extract value from the large amount of data produced by these applications.

One of the key activities in data science to handle this challenge is visual data exploration (known also as exploratory data analysis (EDA) [Tuk77]), a practice that uses mainly data visualization and analysis techniques to understand data and gain new insights. Systems fitting in this class provide facilities to write queries and visualize the results quickly. In visual data exploration systems, users are typically engaged in a loop as depicted in Figure 1.1 where they explore the data iteratively to gain new knowledge. At each iteration of this loop, users issue queries to retrieve the data they want to study further. They next need to construct visualizations suitable to render the issued query's result. Investigating these results may spark further interest, in turn leading to a new iteration of exploration.

However, in practice this cyclic visual data exploration process (cf. Figure 1.1) suffers from two major problems that are (i) identifying and spec-

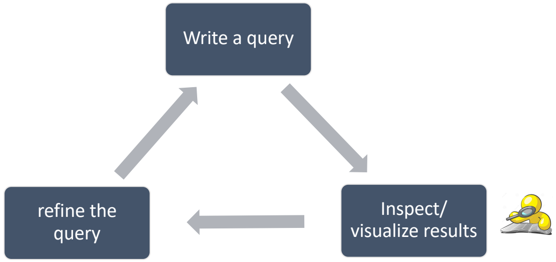


Figure 1.1.: Human-in-the-Loop Visual data exploration process

ifying the queries that narrow down the data of interest, and (ii) finding visualizations for the query results that support the human analysis process. These two problems are commonly encountered when exploring data visually. They mitigate the efficiency of the visual data exploration process and they impede users, thereby leading to the possible oversight of important information available in the explored dataset.

To cope with the two aforementioned problems, several existing work (e.g., [DP13; MHS07; MS16; MS18; MVT16; SK16b; THY+17; VRM+15; WMA+16; WQM+17]) have been proposed. In general, these existing visual data exploration work provide recommendations to assist users in their exploration jobs. More specifically, we distinguish two classes of existing visual data exploration systems based on the type of recommendation: (i) visual data exploration solutions that provide *query recommendations* to guide users to interesting portions of the studied data sets and (ii) visual data exploration approaches that offer *visualization recommendations* to construct visualizations that appropriately render investigated data.

While recent research succeeds to improve the efficiency of the visual data exploration process, there is still a significant gap between query recommendation systems and visualization recommendation systems. Indeed, existing work supporting users in the data querying step (cf. Figure 1.1), do not offer any support in the course of the data visualization step (cf. Figure 1.1). Opposed to that, existing work supporting users in the data visualization step of the visual data exploration process typically offer no or very limited support for the data querying step. Overall, no work so far has considered the process of exploration in its entirety and therefore misses the opportunity of leveraging knowledge gained from (previous) exploration cycles in a holistic way.

Accordingly, in this thesis, we aim at supporting users alongside the *full* process of exploration. This requires to bridge the gap between visualization and query recommendation for visual data exploration that were always considered in existing work separately in order to offer users a more efficient visual data exploration experience.

Towards achieving this goal, we consider *provenance* to support various types of recommendations proposed to users alongside the visual data exploration process. In general, the provenance describes the production process of an end product, which can be anything from a piece of data to a physical object. In our context, provenance describes the different stages (cf. Figure 1.1) performed by users throughout the visual data exploration. The overarching goal of this thesis is to study how provenance can effectively be used to offer users guidance throughout the whole visual exploration process, thereby improving the overall efficiency and quality of the exploration process. This requires to address the following problems:

- We need to identify a provenance model that tracks the visual data exploration process and devise solutions to efficiently capture this provenance.
- We need algorithms that process and analyze the provenance for the purpose of computing recommendations that guide users in their exploration.
- We need to integrate the solutions we devise into a holistic visual data exploration framework to assist users throughout the whole exploration process.

1.2. Thesis contributions

To address the problems mentioned above, this thesis makes the following contributions.

Proposal of an evolution provenance model. Any recommendation algorithm requires data to determine what users may be interested in. To recommend next queries and visualizations to users in the course of visual data exploration process, we collect provenance of (previous) user explorations made by same or other users. This meta-data about user exploration sessions comprises users' queries, users' interactions and visual encoding parameters of rendered visualizations. We call it *evolution provenance*. We describe later the evolution provenance model, which has been published in the following paper.

- [BH17]: H. Ben Lahmar, M. Herschel, *Provenance-based recommendations for visual data exploration*, in the USENIX Workshop on the Theory and Practice of Provenance, TaPP, 2017.

We point out that our evolution provenance model is general enough to capture different visual exploration use cases, as demonstrated by its adoption in the following paper.

- [BBH+19]: V. Bruder, H. Ben Lahmar, M. Hlawatsch, S. Frey, M. Burch, D. Weiskopf, M. Herschel , T. Ertl, *Volume-Based Large Dynamic Graph Analysis Supporting Evolution Provenance*, in the Multimedia Tools and Applications Journal, MTAP, 2019.

Provenance-based recommendation approaches. We aim in this thesis at bridging the gap between visual and query recommendation for visual data exploration to provide a more efficient visual data exploration experience. To this end, we propose several recommendation approaches that assist users in the whole process of visual data exploration. More specifically, we propose three recommendation approaches that are: (i) content-based query recommendation, (ii) collaborative-filtering query recommendation, and (iii) visualization recommendation. The content-based query recommendation approach leverages data provenance of users' interactions to identify information strongly related to users' current focus. Beside taking into account users' proper interests (via content-based query recommendation approach), we propose a novel query recommendation approach to take also into account global trends commonly opted previously by several users. This corresponds to our collaborative-filtering query recommendation approach. To support collaborative-filtering recommendations, we propose several techniques to incrementally merge evolution provenance records, which describe individual user exploration sessions, into a global graph. Finally, we propose a visualization recommendation approach that recommends suitable visualizations to render query results. To do that, our visual recommendation approach leverages evolution provenance that tracks all manipulations performed by the current user.

We have evaluated the performance of our three provenance-based recommendation approaches. Results show the effectiveness of our approaches in terms of runtime, and the interestingness of recommendations output by these methods. We have also investigated the performance of our proposed merge approaches in terms of merge quality, runtime, and conciseness of merged graphs. Results of experiments were beneficial to identify the suitable merge method to adopt among our proposed methods.

The discussion of algorithms and evaluation results of our proposed recommendation approaches relies mainly on the following papers.

- [BH17]: H. Ben Lahmar, M. Herschel, *Provenance-based recommendations for visual data exploration*, in the USENIX Workshop on the Theory and Practice of Provenance, TaPP, 2017.
- [BH19b]: H. Ben Lahmar and M. Herschel, *Towards integrating collaborative filtering in visual data exploration systems*, in the European Conference on Advances in Databases and Information Systems, ADBIS, 2019.
- [BH20]: H. Ben Lahmar and M. Herschel, *Collaborative filtering over*

evolution provenance data for interactive visual data exploration (under submission).

Quantification of recommendation interestingness. Using our provenance-based recommendation approaches, users receive at each iteration of the visual data exploration process a possibly large set of recommendations. Accordingly, the user could be left with a large number of recommendations to choose from to explore next. Intuitively, the choice of the recommendation to study next is difficult. Therefore, we contribute a quantification approach that assesses the interestingness of each recommendation. The rationale behind that is to guide users to select the most interesting recommendations worth inspecting next. We have evaluated our quantification approach in terms of runtime and accuracy of quantification. Results show that the computation of recommendations' interestingness scores is fast enough for an interactive visual data exploration process. Furthermore, experiment results give an insight into the accuracy performances of the diverse implementations of our quantification approach. Overall, the description of this contribution relies mainly on the following paper.

- [BH20]: H. Ben Lahmar and M. Herschel, *Collaborative filtering over evolution provenance data for interactive visual data exploration* (under submission).

Provenance-based framework for visual data exploration. We integrate and generalize our work through a provenance-based framework for visual data exploration that provides a holistic approach to support users in the whole process. Accordingly, our provenance-based framework for visual data exploration assembles our aforementioned contributions including evolution provenance capture, provenance-based recommendation approaches, and methods quantifying recommendation interestingness.

We provide in this thesis evidences about the feasibility of our proposed framework. We discuss an instance of our framework that targets the visual exploration of data warehouses. The discussion of our proposed framework and its instance extends the following paper:

- [BHBK18]: H. Ben Lahmar, M. Herschel, M. Blumenschein, D.A Keim, *Provenance-based visual data exploration with EVLIN*, in the International Conference on Extending Database Technology, EDBT, pp. 686–689, 2018.

Structure-based provenance summary. While the previous contributions focus on using provenance for visual data exploration, we noticed during our research, in particular for our collaborative-filtering recommendation approach, that there is a more general research question of analyzing a set of provenance traces. This motivated us to conduct seminal research on summarizing multiple provenance traces.

More specifically, we propose an end to end solution that infers a structure-based summary from a set of provenance traces that respect the PROV-JSON format. Furthermore, we discuss our proposed solution as well as possible visual analytics tasks that may be applied to this type of summary. Finally, we perform a preliminary experimental evaluation that studies both the runtime and conciseness of our proposed provenance summarization algorithm. The explanation of our proposed approach relies on discussions and evaluation results made in the following paper.

- [BH19a]: H. Ben Lahmar and M. Herschel, *Structural summaries for visual provenance analysis*, in the workshop on Theory and Practice of Provenance, TaPP, 2019.

1.3. Thesis outline

The remainder of this thesis is organized as follows:

Chapter 2 presents an overview of the concepts and paradigms necessary to grasp the contributions of this thesis. More precisely, we give an overview of the visual data exploration process that we tackle in this work. Furthermore, we discuss the most prominent work related to this research area. Note that work more specially relating to our individual contributions are discussed in subsequent chapters.

Chapter 3 introduces our framework ProvVDE meant to visually explore data. This framework integrates all the scientific contributions discussed later in this manuscript. For that, we describe the components present in our visual data exploration framework. Furthermore, we introduce a running example that uses EVLIN, an instance of our framework meant to explore data warehouses. The description of our visual data exploration framework in this chapter relies on scenarios discussed in our paper [BHBK18].

Chapter 4 introduces all formalizations and definitions necessary to describe the evolution provenance model that we have proposed to track visual data exploration processes performed by users using our framework. Note that the content of this chapter is based on [BH17].

Chapter 5 discusses our provenance-based methods proposed and implemented in EVLIN, the instance of our visual data exploration framework ProvVDE. Our proposed provenance-based methods include a content-based query recommendation approach, a collaborative-filtering query recommendation approach, a method proposed to quantify the interestingness of recommendations, and a visualization recommendation approach. All methods and algorithms discussed in this chapter are based on our papers [BH17; BH19b; BH20; BHBK18].

Chapter 6 presents the evaluation and validation of the proposed methods and algorithms discussed in Chapter 5 and implemented in EVLIN. For that, we introduce first the implementation aspects and then the experiments setups that we used to validate our contributions. After that, we discuss the set of quantitive and qualitative experiments performed to study the efficiency and the effectiveness of our contributions. Quantitative experiment results show the efficiency and the effectiveness of our provenance-based methods proposed for interactive visual data exploration while qualitative experiments results show that EVLIN facilitates users' exploration tasks and contributes to an effective visual data exploration experience. Note that the content of this chapter is based mainly on [BH17; BH19b; BH20].

Chapter 7 presents our approach that summarizes a set of provenance traces available in PROV-JSON format. Furthermore, we provide in this chapter some illustrative use cases that showcase possible visual analytics tasks that may be applied to this type of summary. We perform a preliminary experimental evaluation that studies both the runtime and conciseness of summaries output by our approach. The description of our proposed approach relies on discussions made in our paper [BH19a].

Chapter 8 concludes this thesis by summarizing the contributions. We further discuss possible follow-up research as well as new research challenges.

Chapter 2

Background and related work

2.1. Introduction

This chapter provides an overview of the different concepts and paradigms that are necessary to define and put into perspective our contributions. We start by reviewing the visual data exploration process, its main steps, and pertinent existing work in Section 2.2. As the recommendation methods we contribute in this thesis are specialized for visual data exploration in data warehouses, we discuss basic concepts of data warehouses in Section 2.3. Finally, given that our contributions leverage provenance data, we dedicate the rest of this chapter to defining the provenance and its types. Note that Section 2.4 is mainly based on classifications, definitions, and descriptions published in our survey [HDB17].

2.2. Visual data exploration

The visual data exploration (a.k.a. exploratory data analysis [CBYE19; IPC15; WMA+16]) was defined in [CBYE19; IPC15] as the process of browsing data to efficiently extract knowledge and to obtain an overall understanding from data. Typically, it is meant for analysts who are initially unfamiliar with the studied data set. The aim in this case as stated in [SK16b], is to use the visual data exploration to get acquainted with the explored data set. As the main steps of the visual data exploration were described in Chapter 1 (cf. Figure 1.1)), we dedicate the rest of this section to explain types of visual data exploration processes and to discuss relevant related work.

2.2.1. Types of visual data exploration

Inspired by previous classifications of visual data exploration (e.g., [Won18]), we distinguish two classes of visual data exploration: (i) *Question Answering*, and (ii) *Open-ended*.

- *Question Answering*. It is a focused exploration where the user has initially a set of predetermined questions. In this case, the goal of the visual data exploration is to provide answers to the set of questions. For this type of visual data exploration, users often produce analysis reports in the form of written documents and presentation slides. They also sometimes built interactive dashboards. Example of systems providing question answering exploration include CLUE [GLG+16], seeDB [VRM+15], Tableau Show Me [MHS07], and POLARIS [STH02].
- *Open-ended*. It is a more generic form of visual data exploration. Indeed, users have full freedom when performing their exploration jobs. Usually, analysts start with a broad overview in order to get generic information about the shape and the structure of the data. This generic investigation may spark exploration of relevant insights, in turn leading to more focused exploration. In the course of open-ended exploration,

analysts' focus can vary. Accordingly, existing work propose several recommendation approaches to assist users continuously in exploring their changing interests. This holds for instance for Voyager [WMA+16; WQM+17]. Note that, our work falls also in the open-ended category.

2.2.2. Related work

Recently, several systems for visual data exploration have been proposed (e.g., [MHS07; MS16; MS18; MVT16; SK16b; THY+17; VRM+15; WMA+16; WQM+17]). Overall, these existing work tackle the problem of the tedious process of the manual visual data exploration (discussed in Chapter 1) where users may spend considerable time writing queries and constructing suitable visualizations to understand results. Accordingly, state-of-the-art visual data exploration systems propose several recommendation approaches to support users in a particular step of the visual data exploration process such as assisting users in querying or visualizing data.

Based on this observation, we distinguish two classes of existing visual data exploration work based on the type of recommendation supported: (i) those providing query recommendation and (ii) those offering visualization recommendation.

Table 2.1 summarizes existing visual data exploration work (forget so far the last line that refers to an instance of our framework, discussed thoroughly later). For each approach, it describes (i) the expressiveness of input queries (e.g., select-project-join (SPJ) queries, select-project-aggregate (SPA) queries, or cube queries corresponding to SPJA queries), (ii) the type of recommended output queries, (iii) the type of recommended visualization, and (iv) type of supported visual data exploration process. This summary clearly shows that there is a gap between query recommendation systems such as SeeDB [VRM+15], Muve [ESC18], Dive [MSK18], Ziggy [SK16b], [THY+17] or REACT [MS16] for expressive data exploration on the one hand, and visualization recommendation systems such as VisRec [MVT16], Voyager [WMA+16; WQM+17] and Tableau's Show Me [MHS07] on the other hand. Indeed, whereas the former may support

System	input query	recom. query	recom. vis.	exp. type
SeeDB [VRM+15]	cube	sub-cube	-	Question Answering
Muve [ESC18]	cube	sub-cube	-	Question Answering
Dive [MSK18]	cube	sub-cube	-	Question Answering
Ziggy [SK16b]	SJ	SPJ	-	Question Answering
[THY+17]	cube	sub-cube	-	Question Answering
REACT [MS16; MS18]	cube	OLAP queries	-	Question Answering
Show Me [MHS07]	SPA	-	diverse	Question Answering
VisRec [MVT16]	SPA	-	diverse	Question Answering
Voyager [WMA+16; WQM+17]	SPA	Changed SELECT-clause	diverse	Open-ended
EVLIN	cube	OLAP queries	diverse	Open-ended

Table 2.1.: Summary of data exploration systems leveraging query recommendation or visualization recommendation

the various types of queries, they do not offer any visualization recommendation. Typically, there is a one to one mapping between the result relation and a displayed table [MS16] or bar chart [VRM+15]. Opposed to that, visualization recommendation solutions typically offer no or very limited support for query recommendation.

Note that, Table 2.1 shows also that most existing solutions (except of Voyager [WMA+16; WQM+17]) support a visual exploration process of type *Question Answering*. This means that these work do not support an iterative process where users navigate from an exploration to another. In what follows, we discuss our visual data exploration framework that supports *Open-ended* exploration giving thereby users more freedom to get acquainted with the data.

2.3. Data warehouses

In addition to our general framework for open-ended visual data exploration, we propose recommendation techniques tailored to the exploration of data stored in a data warehouse as part of our main framework instantiation named EVLIN. Therefore, this section covers the necessary background and related work on data warehouses and visual exploration for data warehouses.

2.3.1. Main data warehouse concepts

A data warehouse is a database, which is kept separate from the organization's operational database. It encompasses consolidated historical data, resulting from the integration of data coming from different sources. These historical data are used subsequently for analysis and evaluations susceptible to improve the decision making process.

These historical data are commonly represented using a multidimensional model. This model is organized around major subjects that are called *facts*. The multidimensional model stores further quantitative information (known as *measures*) that describe the occurrence of each fact. Typically, the fact occurrences are obviously too many in the data warehouse systems. Thus, to facilitate the exploration of these facts, the multidimensional model arranges them within an n-dimensional space whose axes, called *dimensions*. This is suitable to perform various exploration operations, e.g., aggregation and grouping operations. The concept of dimension gave birth to the concept of *cube* used for representing multidimensional data. This latter contains a set of edges that correspond to dimensions. It contains also a set of cells where each cell corresponds to a fact occurrence.

Note that the literature [VS99] provides several possible implementations of the multidimensional model adopted in data warehousing systems. In our work, we pay a particular attention to a specific implementation that is ROLAP which stands for relational On-Line Analytical Processing. This is explained by the fact that we will discuss subsequently the visual exploration of data warehouses stored in relational way.

On a ROLAP implementation, the relational technology is employed to store data in multidimensional form. One popular implementation of relational OLAP is the *star schema* [Kim96]. This latter contains a set of relations called dimension tables and one relation called a fact table. Each dimension table models a hierarchy of a set of columns where each column maps to a

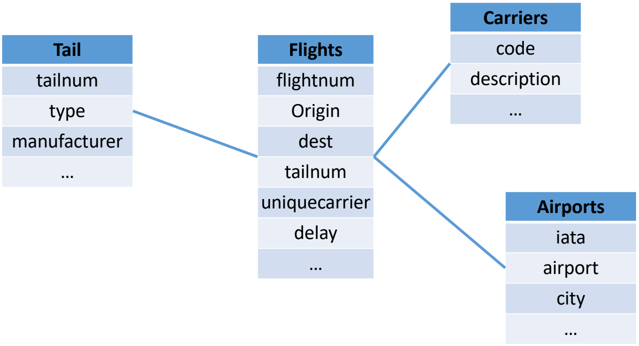


Figure 2.1.: US domestic flights data warehouse schema

granularity of the underlying hierarchy. Example 1 presents a sample star schema for a data warehouse of domestic flights done in USA.

Example 1 (Sample data warehouse) *Figure 2.1 depicts a schema of a data warehouse about US domestic flights* [1]*. It contains a fact table "Flights" recording information about one million flights done between 1987 and 2008. The facts recorded for each flight include various numerical attributes such as delays, cancellation, arrival and departure time etc. This data warehouse contains also four dimension tables containing information about departure airports, destination airports, airlines and plane types. These dimensions contain information about 3300 airports and almost 4500 plane types used by more than 1500 airline companies for the covered flights.*

In general, users who target exploring data warehouses, need to issue OLAP (On-Line Analytical Processing) queries. In this context, we find several OLAP operations that have been widely used in the literature [VS99]. In our thesis, we rely on these widely accepted operations to intuitively navigate the data warehouse data during exploration. Examples of relevant OLAP operations discussed subsequently in our thesis include the following types:

[1] https://stat-computing.org/dataexpo/2009/

- *Roll-up.* It aggregates data by climbing up a hierarchy or by reducing dimensions.
- *Drill-down.* It is the reverse operation of *Roll-up*. It adds detail to data by stepping down to lower-level data or by introducing new dimensions
- *Slice.* It selects data by fixing values or intervals for one dimension.
- *Dice.* It selects data by fixing values or intervals for many dimensions.

2.3.2. Existing work for visual exploration of data warehouses

Several popular products, e.g., QlikView [Qli] and Tableau [Tab] incorporate modern graphical methods suitable to query and explore data warehouses. More specifically, these tools provide high-quality visualizations of the subsets of data queried by the user. However, the specification of queries or data regions to explore remains a user-driven manual task.

Recent works in visual data exploration have tackled the problem of recommending interesting queries when visually exploring data warehouses. Actually, most prominent work in the context of data warehouse visual exploration, were already discussed in Section 2.2.2. Yet, we evoke again these work with highlighting the data warehouse aspect.

In fact, our review of existing work for visual exploration of data warehouses (e.g., [ESC18; MS16; MS18; MSK18; SK16b; THY+17; VRM+15; WMA+16; WQM+17]), shows that these work provide only facilities to explore visually data cubes or precomputed summaries. In other words, these work make a precomputed data cube available to all users. In this context, the user cannot navigate to other dimensions not considered in the predefined cube. Accordingly, users never explore the whole data warehouse.

In contrast to existing visual exploration solutions that are limited to the exploration of a specific data cube, we discuss later an instance of our visual data exploration framework that offers users the possibility to analyze the whole data warehouse.

Note also that the same problem stated in Section 2.2.2 holds for approaches that enable the visual exploration of data warehouses. Hence, we observe a gap between systems supporting query recommendation such as [ESC18; MS16; MS18; MSK18; SK16b; THY+17; VRM+15], and existing systems supporting visualization recommendation such as Voyager [Qli; Tab; WMA+16; WQM+17] on the other hand.

To cope with this problem, we present later our framework for exploring data visually. This framework provides several facilities including the visualization recommendation and query recommendation. Furthermore, we provide an instance of our framework that validates the feasibility of our methods when exploring data warehouses visually.

Results of experiments in Chapter 6 validate the efficiency and the effectiveness of exploring visually data warehouses as a whole using the instance of our visual data exploration framework that we will describe it later.

2.4. Provenance

As introduced in Chapter 1, we aim at studying methods required to collect and leverage provenance to improve the visual data exploration process. More specifically, among the large body of work on provenance (see our survey [HDB17]), we focus in what follows on research on two specific provenance types, namely *workflow provenance*, and *data provenance*.

In what follows, we pay particular attention to the two particular types, namely data provenance and workflow provenance as they are related to our contribution.

Workflow provenance. This type of provenance is tailored to processes that form workflows. A workflow is considered as a graph, where vertices represent arbitrary functions or modules and edges capture interactions e.g., the flow of data between these vertices. The wealth of workflow provenance content allows for different forms and granularities of provenance. This results in the classification proposed in our survey [HDB17] and depicted in Figure 2.2. All three forms depicted in Figure 2.2, are independent from one another. Hence, a provenance solution may support only one, two, or all three of them. First, we have the *prospective provenance* (e.g., [ABGK13;

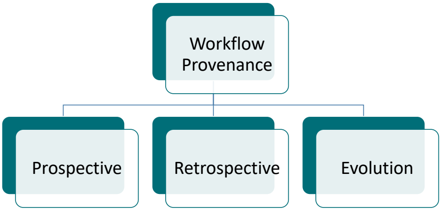


Figure 2.2.: Workflow provenance classification [HDB17]

DBK+15]) that captures the structure and static context of a workflow. It is independent of any workflow execution or input data. Hence, it is statically derived from the workflow definition.

The second type is *retrospective provenance* (e.g., [ABJ06; MBC+14; TMF+10]). It stores information about a workflow execution, i.e., with information becoming available when actually running the workflow. This information includes facts about the execution of each workflow step as well as the runtime environment. Retrospective provenance also preserves information on the resources that are accessed or generated during execution.

Finally, we have *evolution provenance* (e.g., [ABJ06; CFS+06]). This form of provenance is also called provenance of the (development) process in the domain of scientific workflows [MDB+13]. It reflects the changes made during the design of a workflow. In this case, the provenance solution tracks all modifications made to alter a workflow. We design in this thesis (see Chapter 4) a new provenance model for visual data exploration. This provenance data model falls in the category of *evolution provenance* as well as in the category of *retrospective provenance*. Indeed, our new provenance data model describes an actual run of a visual data exploration process including namely user's interactions, exploration queries, and inspected visualizations.

As *evolution provenance* and *retrospective provenance* categories are related to our contributions discussed later, we provide in Example 2 a sample of provenance that illustrates these two categories.

Example 2 (Evolution and retrospective provenance samples) *Assume that a scientist was using a scientific workflow W_f to perform genomic analysis.*

Suppose that the scientist did not obtain insightful results when using W_f. Accordingly, the scientist performed several modifications to W_f. After each modification, the scientist executed W_f to check obtained results. In this case the evolution provenance is a directed acyclic graph that summarizes the set of modifications made by the scientist. Each node in the evolution provenance graph represents a particular version of the workflow W_f. Edges of the evolution provenance graph represent the modifications applied to W_f in order to derive several new versions of this workflow.

Imagine that after several modifications, the scientist decides to inspect again a previous version W_{f_v} (made after several modifications on the input workflow W_f) in order to remember its results. In this case, the retrospective provenance is useful. It shows a graph containing the set of steps that constitutes this particular version W_{f_v}. The retrospective provenance shows results obtained using W_{f_v}. It includes also facts about the execution of each step in W_{f_v} as well as the runtime environment.

Data provenance. Data provenance tracks the processing of individual data items manipulated throughout a data derivation process. Typically, this kind of provenance is applied on structured data models and declarative query languages with clearly defined semantics of individual operators. This is important to record the provenance of individual data items after running a data transformation or to extend the data or operators to pass on their provenance annotation as the data is processed.

Data provenance research focuses on explaining data present in a query result and may be categorized in three forms as surveyed in [HDB17]. In our work, we focus on one specific category that is is *why provenance* [BKT01; CW01]. This type of provenance describes the origins of data in the result of a query (i.e., based on what source data). More specifically, it returns the set of source tuples that witness the existence of the tracked query result. Typically, this particular set of tuples is known in the literature as *lineage*.

In this thesis, we leverage why provenance to provide recommendations strongly related to the users' interests when exploring data visually. We postpone the discussion of our provenance-based recommendation approach

flightID	origin	dest	
F1	SEA	ORD	t_1
F2	SFO	JFK	t_2
F3	ORD	SFO	t_3

(a) Flights

iata	city	
SEA	Seattle	t_4
ORD	Chicago	t_5
RBD	Dallas	t_6

(b) Airports

```
SELECT F.city as city_dep , A.dest FROM Flights as F ,Airports as A WHERE A.iata=F.origin
```

Query result		Flights			Airport		Why_provenance	
city_dep	dest	prov_flightID	prov_origin	prov_dest	prov_iata	prov_city	Lineage	
Seattle	ORD	F1	SEA	ORD	SEA	Seattle	$\{t_1, t_4\}$	t_7
Chicago	SFO	F3	ORD	SFO	ORD	Chicago	$\{t_3, t_5\}$	t_8

Figure 2.3.: Sample why provenance

to Chapter 5.

To further clarify the why provenance definition, consider the following Example 3.

Example 3 (Why provenance example) *This example leverages the data warehouse of US domestic flights described in Example 1. For that, we consider the query and the relations $Flights$ and $Airport$ depicted in Figure 2.3. For convenience, we show a simplified version of relations $Flights$ and $Airport$ in Figure 2.3. Additionally, we provide in each table an identifier for each tuple at the right of the tuple. We propose a query that joins a* Flight *with an* Airport *relation to return the city of departure and the id of the airport of destination. This results in two tuples produced by joining a tuple from relation* Flight *with a tuple from relation* Airport*. For instance, the tuple t_7 is derived from tuples t_1 and t_4. In this case, tuples t_1 and t_4 present the why provenance information associated to t_7. Overall, the why provenance of each result of this query is described in the last column of the relation depicted at the bottom of Figure 2.3.*

2.5. Conclusion

In this chapter, we presented relevant background to put our research in the context of the visual data exploration and to introduce main concepts we rely on. In the next chapter, we give an overview about our framework for *open-ended* visual data exploration.

Chapter 3

Framework for provenance-based recommendation in visual computing

3.1. Introduction

This chapter formalizes our framework for holistic recommendation-based visual data exploration. Furthermore, we present the set of features and functionality that our framework makes available to the users to facilitate their exploration tasks.

We introduce our framework in Section 3.2. In Section 3.3, we describe the main modules and components present in this framework. Subsequently, we describe in Section 3.4 the set of possible navigations that can be made using our framework when exploring data visually. Furthermore, we discuss in Section 3.5 the process of visual data exploration that users can perform using an instance of our framework. Note that this chapter relies partially on descriptions and exploration scenarios presented in our paper [BHBK18].

3.2. Framework overview

We propose in this chapter ProvVDE (Provenance-based Visual Data Exploration), our novel framework meant to explore data visually. In contrast to existing work that assist users in one particular stage of the exploration (e.g., in querying or visualizing data), our framework ProvVDE adopts a holistic approach that supports users in all phases of the visual data exploration process. To do so, ProvVDE provides several features to help users in performing the visual data exploration process efficiently. Figure 3.1 depicts the general framework, which comprises several components and requires a user to interact with the system.

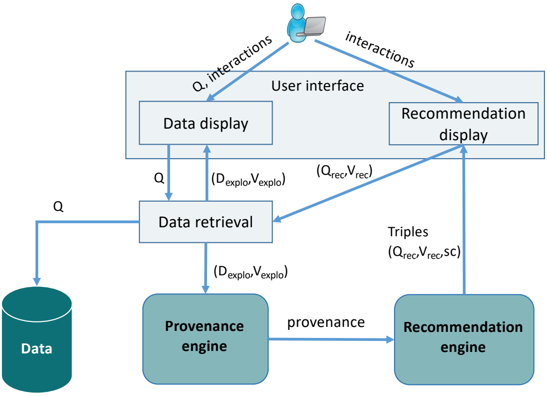


Figure 3.1.: ProvVDE's architecture

3.3. Framework components

This section describes the modules and components present in the architecture of our visual data exploration framework as depicted in Figure 3.1.

Data. The *Data* presents the input of our framework ProvVDE. It is structured data that may reside in a relational database, flat files, etc.... This input requires proper access methods to retrieve subsets of the data.

Visual user interface. The *Visual user interface component* allows users to interact with data of interest (e.g., through queries, interactions, faceted navigation, ...). It consists of two main interfaces where the first one renders data regions under investigation. The second interface encompasses recommendations suggested to the user to explore next.

To provide these two interfaces, the *Visual user interface module* consists of two sub-components. The first one is the *Data display* sub-component. It takes as input a query Q specified by the user. In this case, the *Data display* sends the query to the *Data retrieval module* in order to process it and to get results. Later, the *Data display* sub-component receives the users' query result (a dataset D_{explo} and a visualization specification V_{explo}) from the *Data retrieval module*. Ultimately, it renders D_{explo} according to the visual encodings properties specified in the visualization V_{explo}. The second sub-component is the *Recommendation display* sub-component. It takes as input a set of triples (a recommended query Q_{rec}, a recommended visualization V_{rec} and a score of interestingness sc). Ultimately, it renders an interactive visualization VIS that shows the set of recommendations that the user can study next. Note that the *Visual user interface* component is interactive. Hence, the user can issue queries whose results will be rendered by the *Data display* sub-component. Furthermore, a user can inspect interactively results rendered by the *Recommendation display* sub-component to gain more knowledge (e.g., via hovering or clicking on sub-results of interest). Overall, possible user interactions and navigations are described in the following Section 3.4.

Data retrieval. The *Data retrieval module* is responsible for processing users' requests to fetch a specific region of data. It takes as input a dataset D and a query Q (specified via the *Data display* sub-component). Then, it returns a pair of information: (i) D_{explo}, a region of data $\subset D$ that meets the user's request Q, and (ii) V_{explo} (sent from the *Recommendation display* sub-component) that contains the visual encodings properties required to render D_{explo}.

Provenance engine module. The *Provenance engine module* is responsible for collecting/processing provenance information that tracks an actual run of a visual data exploration process. It takes as input information about explored data region D_{explo} from the *Data retrieval module*. The *Provenance engine module* takes also as input visual properties information V_{explo} used to render data region D_{explo}. Pairs of explored data region D_{explo} and visual properties information V_{explo} present the provenance traces collected by the *Provenance engine module*. Captured provenance traces are then processed and analyzed also by this module to extract information interesting for users in the course of the visual data exploration process. The processed provenance information is sent ultimately to the *Recommendation engine module* that is described below.

Recommendation engine module. As its name indicates, the *Recommendation engine module* generates recommendations to assist users in exploring datasets efficiently. Essentially, it leverages provenance information captured and analyzed by the *Provenance engine module* to produce recommendations. More specifically, the *Recommendation engine module* computes two types of recommendations: (i) recommended queries deemed interesting to the current user, and (ii) recommended visualizations that are suitable to render the result of a recommended query proposed to the user.

Furthermore, the *Recommendation engine module* assigns an interestingness score sc to each recommended query Q_{rec} to help users in choosing suitable recommendations to study next. This incurs ultimately the generation of a set of triples (Q_{rec}, V_{rec}, sc) that will be rendered through the *Visual user interface module*.

3.4. Framework navigations

We have described in Section 3.3 the set of components and modules that compose our framework ProvVDE. A specific attention is given in this section to explain the set of possible navigations and interactions that permit the user to use and invoke components and modules depicted in Figure 3.1. Essentially, we distinguish the following three navigation types supported by our visual data exploration framework.

- Interact (ER_i, r_i): get a set of recommendations L_{T_i}. This corresponds to the interaction of a user with a sub-result r_i during the investigation of an exploration result ER_i displayed through the *Data display* sub-component. This triggers the computation of recommendations via the *Recommendation engine*. This latter as discussed in Section 3.3, outputs a set of triples L_{T_i} (recommended query, recommended visualization, and an interestingness score). These triples are rendered to the user via the *Recommendation display* sub-component.
- getNextRec(L_{T_i}, T_j): move to a new exploration ER_i. This corresponds to the situation where a user investigates the set of recommendation triples L_{T_i} displayed via the *Recommendation display* sub-component. At this stage, the user can select a triple T_j (corresponding to a new exploration ER_i) to inspect next.
- Recover (ER_i): jump back to an already investigated exploration result ER_i. The recover action consists of going back one step to recover the last exploration result. It includes also jumping back many previous exploration results to resume an exploration result ER_i. The recover action is invoked either when inspecting the set of recommendation triples L_{T_i} or when inspecting an exploration result ER_{curr}.

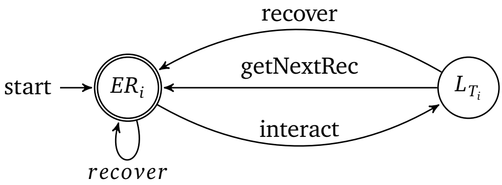


Figure 3.2.: Automaton associated to the set of possible navigations supported in ProvVDE framework

These navigations supported in our framework ProvVDE form an automaton, that is depicted in Figure 3.2. More precisely, states in this automaton present explorations queries and recommendations inspected by users in the course of the visual data exploration process. Directed edges depicted in this automaton present the set of aforementioned navigations supported by our framework ProvVDE. Overall, this automaton shows that users can perform open-ended visual data exploration processes using our framework.

3.5. Framework implementation

We present in this section EVLIN (short for exploring visually with lineage) that is an instance of our framework ProvVDE meant to explore *data warehouses*. The rationale behind that is to clarify the general idea of our framework ProvVDE by introducing a running example that is targeted towards the visual exploration of a data warehouse. This running example will later serve to further illustrate our scientific contributions proposed in this thesis.

EVLIN supports interactive visual data exploration over a multidimensional relational dataset D, typically stored in a data warehouse with star-schema. In this case, the data warehouse corresponds to the *data* input discussed in Section 3.3. We benefit also from the DBMS used to store the data warehouse in relational way. Accordingly, the DBMS corresponds to the *data retrieval component* present in the architecture of our framework ProvVDE. Hence, it is responsible for processing exploration queries investigated by the user. We point out that a sample of data warehouses that we can explore using EVLIN is already discussed in Example 1.

An initial user-specified query Q is input via a graphical user interface implemented in our system EVLIN. The seed exploration query issued by the user is defined as follows.

Definition 3.1 (Exploration query) *Given a data warehouse D with a fact table T, a set of measures M in T, a set of dimension tables A, and a set of aggregate functions F, an exploration query Q is an SQL query of the form*

$$\texttt{SELECT } a_1, ..., a_n, f(m) \texttt{ FROM } rel(D) \texttt{ WHERE } cond \texttt{ GROUP BY } a_1, .., a_n$$

where $m \in M$, $a_i \in schema(A)$, $f \in F$, $rel(D)$ refers to one or more relations in D, and $cond$ is a (conjunction of) predicates.

Example 4 (Sample exploration query) *In this example we consider the data warehouse of US domestic flights depicted in Figure 2.1 and the following seed exploration query Q that determines the number of delayed flights per state of departure.*
SELECT count(*), A.state FROM Flights `as F`*, Airports* `as A` *WHERE A.iata=F.origin* `AND` $F.depdelay > 0$ *GROUP BY A.state*

The exploration query Q over the data warehouse D is executed and its result $Q(D)$ is input to the *Recommendation engine module* present in the architecture of our framework ProvVDE (cf. Figure 3.1) and implemented in EVLIN. It is worth stressing that the *Recommendation engine module*, implemented in EVLIN leverages provenance information collected by the *Provenance engine module* implemented also in EVLIN. This latter produces two types of provenance that are why provenance and evolution provenance. While we postpone the discussion of the *Provenance engine module* to Chapter 4, we describe in what follows the recommendations output by the *Recommendation engine module*.

First, the user's initial query result $Q(D)$ is input to *Recommendation engine module*. This latter determines an adequate visualization for $Q(D)$ to be displayed to the user. Note that EVLIN targets 2D visualizations (bar, scatter,

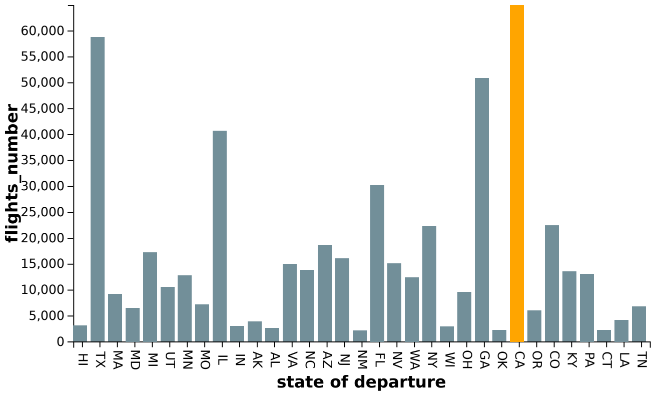


Figure 3.3.: Example of recommended visualization for $Q(D)$

line, and bubble charts), so we limit the number of grouped attributes in exploration query Definition 3.1 to two, i.e., $n \leq 2$ in practice.

Example 5 (Sample recommended visualization) *As example, Figure 3.3 displays the recommended bar-chart visualization for our sample query Q discussed in Example 4 . The x-axis of this visualization enumerates the set of states of departure where delayed flights occur. The y-axis reflects the number of delayed flights. Bars rendered in green color (ignore so far the orange color that will be discussed later) depict flights per states of departure.*

EVLIN generates interactive visualizations. Thereby, the users can inspect (by mouse hovering) the data and select a sub-result $r_i \in Q(D)$ that has attracted their attention, to pursue the exploration.

Example 6 (User's interaction) *Going back to our running example, the highest bar that designates the state "CA" drew the analyst's attention when investigating results depicted in Figure 3.3. Then, this latter double clicked this sub-result. This is reflected in Figure 3.3 where the bar associated with the state "CA" is highlighted in orange color.*

The selected sub-result r_i is processed by the *Recommendation engine module* that identifies which information may be of interest to the user at the next step of exploration. Ultimately, the *Recommendation engine module* thoroughly discussed in Chapter 5 returns a set of triples (recommended query, recommended visualization, and an interestingness score). This output is rendered to the user in the form of an *impact matrix* (e.g., Figure 3.4).

The rows represent the set of attribute-value pairs identified by the *Recommendation engine module* component as information strongly related to the user's interaction. Columns of the impact matrix reflect various OLAP operations types suitable to navigate in the data warehouse (cf. Section 2.3). Thereby, each cell maps to a recommendation. The cell color encodes the interestingness score of each recommendation. The score of each recommendation, thoroughly discussed in Chapter 5, reflects the potential utility of the recommendation as well as its popularity among previous explorations made on the same data warehouse by multiple users.

Example 7 (Impact matrix) *Referring back to our running example exploration discussed in Example 4, Figure 3.4 shows the impact matrix associated with the user's interaction discussed in Example 6. This matrix encompasses two rows corresponding to the two attribute-values pairs deemed strongly re-*

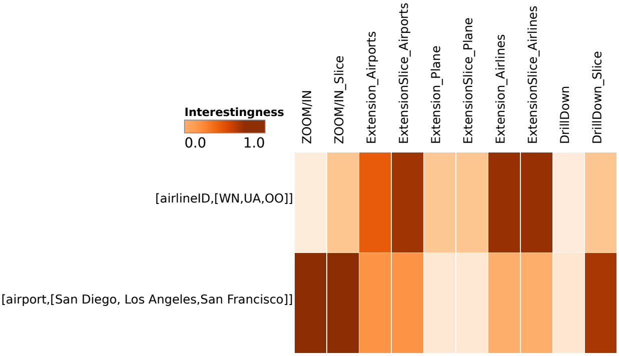


Figure 3.4.: Example of an impact matrix

lated to the user interaction described in Example 6. For instance, the second row indicates that delayed flights stand out when departing from airports in San Diego, Los Angeles, and San Francisco, whereas the first line suggests to further investigate with respect to three major airlines. For each interesting pair information, the Recommendation engine module *component generates several recommended queries having different types suitable to navigate in data warehouses such as ZoomIn, Drill and other types that will be explained in Chapter 5. For instance the* Zoom/In_Slice *recommended query associated with the pair* $(airport, \{SanDiego, LosAngeles, SanFrancisco\})$ *shows the distribution of delayed flights when departing from these three airports. Another example of recommendation is the* Zoom/In *recommended query associated with the pair* $(airlineID, \{WN, UA, OO\})$*. This latter groups delayed flights by airlines and by states of departure. The rationale behind that is to study the performance of these three airlines in California compared with other states.*

As shown above, scored recommended queries are visualized in form of an impact matrix, allowing the user to choose which of the suggested queries Q' to execute next. At this point, the next exploration step of the exploration session starts, by setting Q to Q'.

Example 8 (User's navigation) *After inspecting interestingness scores of recommendations shown in the impact matrix depicted in Figure 3.4, the user selects the cell corresponding to the "ExtensionSlice_Airports" type and associated with the attribute-values pair* $(airlineID, \{WN, UA, OO\})$ *as it has a dark red color. In this case, a new exploration step starts where* Q *is equal to the query present in the selected cell.*

3.6. Conclusion

This chapter introduced our holistic recommendation-based visual data exploration framework ProvVDE. Besides that, we presented EVLIN, an instance of our framework meant to explore data warehouses visually. In the next chapters, we continue describing our framework. Hence, we describe thoroughly two main modules present in our framework that are the *provenance engine module* and the *Recommendation engine module*. While the description of the *provenance engine module* is general, the description of the *Recommendation engine module* is tailored to a particular framework instance (in our case EVLIN for visual exploration of data warehouse). The rationale behind that is to provide evidences about the feasibility of our proposed recommendations methods.

Chapter 4

Evolution provenance data model

4.1. Introduction

We have seen in Chapter 3 that our framework ProvVDE relies on provenance, i.e., meta-data about a user's exploration process to determine and quantify recommended queries and visualizations. This chapter now introduces the novel provenance model that allows us to provide such interrelated recommendations. This relies mainly on formalizations and definitions elaborated in our work [BH17; BH19b].

For that, we discuss first in Section 4.2 existing evolution provenance data models. Then, we introduce in Section 4.3 the evolution provenance data model adopted in our framework ProvVDE.

4.2. State of the art of evolution provenance

Historically, evolution provenance was mainly linked to scientific workflows where it was used to document the (development) process. Thereby, it automatically tracks the changes made between two versions that can be inputs, parameters, or functions invoked. Callahan et al proposed the first approach [CFS+06] that captures this kind of evolution provenance. The rationale behind that was to accelerate the decision process about the correct module semantics since multiple workflow executions can be compared visually using the proposed tool called VisTrails. In the same context, Ellkvist et al. [EKA+08] propose a solution that facilitates collaboration between developers. It tracks changes made by all users in realtime during a collaborative design of the workflow and renders them in the form of branches. This particular type of provenance obtained in the two aforementioned approaches [CFS+06; EKA+08] is known as “workflow provenance” or “provenance of the development process".

We use the term “evolution provenance” proposed in [HDB17] as it was shown that this particular type of provenance also encompasses other applications besides scientific workflows. Indeed, unlike aforementioned work [CFS+06] and [EKA+08] where tracking the evolution provenance in scientific workflow engines is only limited to changes of the workflow specification, we capture in this thesis the evolution provenance for visual data exploration to describe an actual run of a visual exploration session, divided into individual exploration steps.

In the same context, we find several existing work (e.g., [GLG+16; MS16]) that collect evolution provenance for visual data exploration processes. For instance, REACT [MS16] computes evolution provenance and uses it to compute collaborative-filtering query recommendations. CLUE [GLG+16], a history based visual exploration system captures the evolution provenance of a visual data exploration process and offers users the capability to assemble states of interest (i.e., interesting explorations tasks) into a story that can be used to explain or to remember exploration tasks made to reach important observations.

Our review of evolution provenance data models adopted in existing visual data exploration work, (e.g., [GLG+16; MS16]) shows that these work focus either on collecting visualization-related properties or on collecting query-related properties. More specifically, REACT [MS16] proposes an evolution provenance data model that stores information about inspected queries and navigations types (e.g., data retrieval, cube operations, and data mining) whereas it neglects completely the storage of visualizations information rendered in the course of the exploration. Opposed to that, CLUE [GLG+16] offers an evolution provenance data model that focuses mainly on the collection of visualization-related properties. Accordingly, the evolution provenance model proposed in CLUE [GLG+16] stores users' interactions and visual encodings of rendered visualizations. Note that CLUE [GLG+16]'s evolution provenance data model has a very limited support to query-related properties. Hence, this model stores inspected data sets. This is not convenient when inspecting large datasets and may incur a costly process of evolution provenance collection and storage.

In what follows, we propose a novel evolution provenance model that ensures the collection of visualization-related properties as well as query-related properties.

4.3. Evolution provenance data model representation

We have introduced in Chapter 3 ProvVDE, our visual data exploration framework. Overall, it is clear from this description that our framework ProvVDE offers users the possibility to perform a set of consecutive exploration steps where they explore visually at each step a different region of the data. More precisely, we define an exploration step as follows.

4.3.1. Exploration step

Definition 4.1 (Exploration step) *An exploration step over data in D is defined as $X^D = \{Q, V\}$ where Q is the exploration query whose result $Q(D)$ over dataset D is rendered with an interactive visualization described by V.*

Inspired by [WPM+17], we record meta-data about *visual exploration resources* used to render visualizations in a relational database. The design of this schema draws on a set of visualization specifications such as Vega [SMWH17] and D3 [BOH11], which define a visualization using four fundamental components. (1) The *scale* component maps data values to visual values, e.g., specification of position, color or shape encodings for such data value. (2) The *axis* component provides a reference for reading the visual/data mapping defined by the *scale* component. (3) Graphical *marks* describe the graphical forms adopted to visually represent data, e.g., circles, rectangles, or points. (4) Finally, a given graphical form is further specified using a set of encoding *channels* that describe the visual mapping of each attribute. In addition to these main components, we further store information about data to render, or the width and the height of a visualization.

This gives rise to the following relational schema depicted in Figure 4.1.

In this relational schema, we present first the name of each relation. Furthermore, we list the set of attributes present in each relation. Note that underlined attributes refer to the primary keys of each proposed relation while arrows refer to the foreign keys.

In what follows, we use Sch_{VIS} to refer to this relational schema (depicted in Figure 4.1). This relational schema Sch_{VIS} stores separately information

$$\begin{aligned}
&Visualization(\underline{idVisualization}, width, length, idQuery),\\
&Mark(\underline{idMark}, markType, idVis \rightarrow Visualization),\\
&AxisUsage(\underline{idAxisUsage}, typeUsage, idAxis \rightarrow Axis,\\
&\qquad\qquad idVis \rightarrow Visualization)\\
&Axis(\underline{idAxis}, tiltle, tick, type, idScale \rightarrow Scale)\\
&Scale(\underline{idScale}, typeScale, nameScale, fieldDom, fieldRange,\\
&\qquad Literal, idDataset \rightarrow Dataset)\\
&Dataset(\underline{idDataset}, value, name, transformation, Literal, source)\\
&Channel(\underline{idChannel}, typeChannel, typeChannel, idScale \rightarrow Scale)\\
&ChannelUsage(\underline{idChannelUsage}, typeUsage, idChannel \rightarrow Channel,\\
&\qquad\qquad\quad idMark \rightarrow Mark)
\end{aligned}$$

Figure 4.1.: Sch_{VIS} : the relational schema modelling visualizations

about *Visualization*, *Axis*, *Scale*, *Mark*, and *Channel*. Given that an axis may be used by more than one visualization in the course of the visual data exploration process, Sch_{VIS} encompasses a table *AxisUsage* that stores information about relationships between visualization and used axis. Similarly, Sch_{VIS} contains a table *ChannelUsage* that stores relationships information between encoding channels and graphical marks given that an encoding channel may be used by many graphical marks.

Based on this design of visualization resources, a visualization V generated in an exploration step is defined as a query over Sch_{VIS}. The query that specifies a visualization V is defined as follows.

Definition 4.2 (Visual exploration resource) *A visualization V for an exploration step $X^D = \{Q, V\}$ is defined as the following query over the relational schema Sch_{VIS}.*

$$\begin{aligned} V \quad = \quad & \Pi_{Vis.*,Scale.*,Dataset.*,Mark.*,Channel.*,Axis.*}(\\ & \sigma_{query=Q}(Visualization) \bowtie Mark \bowtie ChannelUsage \\ & Channel \bowtie Scale \bowtie Dataset \bowtie AxisUsage \bowtie Axis) \end{aligned}$$

4.3.2. Exploration path

As described in Section 3.4, the user can navigate using our framework ProvVDE from an exploration step to another to acquire more knowledge. These navigations result in exploration paths defined as follows.

Definition 4.3 (Exploration path) *We denote the exploration path that leads to the currently investigated exploration step $X^D_{curr} = \{Q_{curr}, V_{curr}\}$ as $P_X = [X^D_0, X^D_1, \ldots, X^D_k, X^D_{curr}]$ where P_X encompasses all X^D_i $\forall i \in [0, n]$ explored by the user, and P_X is the path from the seed exploration step X^D_0 until reaching X^D_{curr}.*

4.3.3. Evolution provenance graph

The automata depicted in Figure 3.2 shows that the user can enjoy using our framework ProvVDE a flexible navigation model to explore the data. This results in many exploration paths (see Definition 4.3). These paths can be gathered to get the full story about the user's manipulations over an exploration session. The result is an *evolution provenance* graph. As its name indicates, the evolution provenance graph tracks the evolution of the visual data exploration process in the course of a user's exploration session. Hence, it records a user's exploration session including, for each exploration step, meta-data about the explored data, the query issued to reach this particular region of data, and the visualization resources used to render the explored data. Overall, we define the evolution provenance as follows.

Definition 4.4 (Evolution provenance graph) *An evolution provenance graph describes an exploration session over a dataset D. It is a labeled directed acyclic graph (DAG) $G_{XS,D}(N,E)$ where N is a set of nodes and E a set of labeled edges. Each node $n \in N$ corresponds to an exploration step X. An edge $e = (n, n', L)$ represents the transition (see navigations in Section 3.4) from one exploration step $X_D = \{Q, V\}$ to the next exploration step $X'_D = \{Q', V'\}$ whose query Q' is a recommendation directly derived from Q over the same D. L is a 2-tuple $\langle const, s\rangle$ where $const$ is metadata that describes the derivation process of Q' from Q, and s is an impact score computed as $s(e) = 1 + \sum_{e_c=(n',n_i,L')\in E} s(e_c)$.*

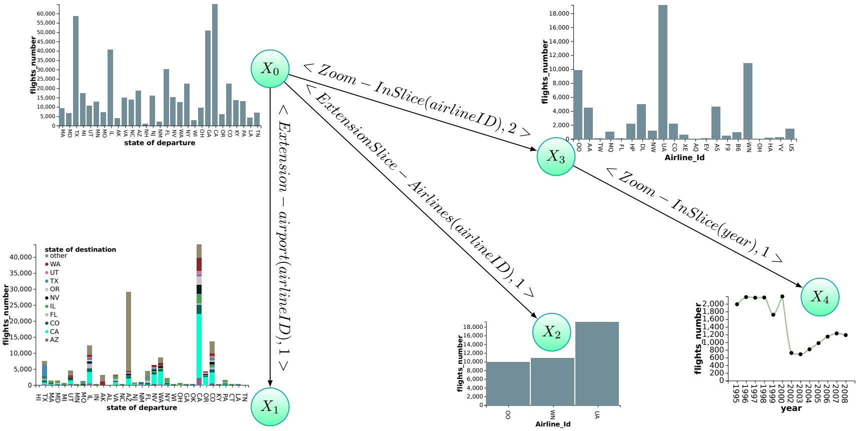
Note that when D is clear from the context, we omit the subscript D from X_D and $G_{XS,D}$.

Based on this definition, the evolution provenance graph represents the full exploration session made by a user. The utility of each edge (navigation) is measured using a score $s(e)$. This score reflects the impact of the transition from an exploration step to another on the rest of the exploration session. Accordingly, the score $s(e)$ is computed as the number of subsequent exploration steps stemming from this navigation e augmented with one.

Note that the DAG property is an important aspect of the evolution provenance graph. Indeed, using the direction of edges, we can understand the impact of each performed exploration step on the acquirement of knowledge in the rest of the exploration session. Accordingly, we leverage this information as we will show later to recommend exploration steps having high impacts, to users exploring later the same dataset. The direction property of the DAG is also useful for the storytelling process [GLG+16] where an analyst needs to explain to an audience how such insight was discovered. Finally, we point out that the acyclicity property ensured by the DAG, is also an important property in the context of visual data exploration as it maintains the rationale of recommendation by prohibiting suggesting exploration steps already investigated by the user.

Example 9 (Sample evolution provenance graph) *Figure 4.2 shows an example of evolution provenance graph. It describes a user's exploration session made using EVLIN.Nodes in this graph represent exploration steps. Edges in this graph contain two labels: (i) the first refers to the type of recommended query and the attribute-value pair used do construct the recommendation (cf. Example 7), and (ii) the second label is the score referring to the number of subsequent explorations reached when traversing this edge. The user initially queries the set of delayed flights as described in Examples 4 and 5. Later, the user double clicks on state California (see Example 6) as it has the highest number of delayed flights. This triggers the computation of recommendation visualized in Example 7.*

At this stage, the user selects first the Extension *recommendation with airport of destination query associated to the* $\{(airline\_Id, \{WN, UA, OO, AS\})$ *pair. This leads to the generation of node* X_1 *and an edge e from* X_0 *to* X_1*. These information are stored in the evolution provenance graph. Now let us assume that the user goes one step back to explore other recommendations visualized in Example 7. In this case, the edge e from* X_0 *to* X_1 *has an impact score equal to 1 as the exploration step* X_1 *has no subsequent steps. Now, the user inspects two recommendations related to the exploration step* X_0*. This leads to the generation of two new navigations (edge from* X_0 *to* X_2 *and edge from* X_0 *to* X_3*)*



X_0
X_1
X_2
X_3
X_4
$< Zoom - InSlice(airlineID), 2 >$
$< ExtensionSlice - Airlines(airlineID), 1 >$
$< Extension - airport(airlineID), 1 >$
$< Zoom - InSlice(year), 1 >$
flights_number
state of departure
state of destination
other
WA
UT
TX
OR
NV
IL
FL
CO
CA
AZ
Airline_Id
year

Figure 4.2.: Sample evolution provenance graph

in the evolution provenance graph. Subsequently, the user continues exploring information related to the result inspected in the step X_3. Accordingly, the user navigates to the exploration step X_4. Based on user's interactions, the score labeling the edge from X_0 to X_3 is set to 2, to reflect the number of exploration steps, pursued upon reaching X_3, indicating thereby how many potentially interesting steps can be reached from this exploration step.

Now, we discuss the mapping process performed to model the flow of information incurred from the automata depicted in Figure 3.2 following our evolution provenance data model. This relies mainly on inference rules depicted in Figure 4.3.

Essentially, our evolution provenance graph maintains a pointer that we call *context*. This latter presents the current entry to expand the provenance graph. Accordingly, any subsequent exploration step performed later, will be considered as a direct descendant of the *context*. Our evolution provenance graph is initialized by a first exploration step as specified in the *Initialization* inference rule (shown in Figure 4.3). This first exploration step is introduced in the evolution provenance graph as a node. It is then designated as a *context*. This latter is updated in the course of the exploration session via

$$\text{(I}\textsc{nitialization})\ \frac{evolutiongraph = \emptyset}{V \leftarrow V \cup \{ES_i\}, context \leftarrow ES_i} \qquad \text{(}\textsc{context inference})\ \frac{Interact(ES_i) = L_{T_i}}{context \leftarrow ES_i}$$

$$\text{(}\textsc{context update}1\text{)}\ \frac{recover(ES_i) = ES_{i-k}}{context \leftarrow ES_{i-k}} \qquad \text{(}\textsc{context update}2\text{)}\ \frac{recover(L_{T_i}) = ES_{i-k}}{context \leftarrow ES_{i-k}}$$

$$\text{(N}\textsc{ode inference})\ \frac{getNextRec(L_{T_i}, T_j) = ES_{new}}{V \leftarrow V \cup \{ES_{new}\}} \qquad \text{(E}\textsc{dge inference})\ \frac{getNextRec(L_{T_i}, T_j) = ES_{new}}{E \leftarrow E' \cup \{(context \rightarrow ES_{new})\}}$$

Figure 4.3.: Evolution provenance graph inference rules.

inference rules *context update1* and *context update2* when the user performs a navigation of type *interact* or of type *recover* (cf. Figure 3.2). Finally, the *getNextRec* navigation (cf. Figure 3.2) is responsible for the generation of new edges and nodes that will be added to the evolution provenance graph. Hence for this particular type of navigation, the user inspects the set of recommendations triples (L_{T_i}) output by our framework and selects a particular triple T_i (corresponding to a new recommended exploration step ES_{new}) to study next. This leads to the inference of a new node added to the evolution provenance graph (see inference rule *node inference*). Similarly, a new edge ($context \rightarrow ES_{new}$) between the *context* and the new node corresponding to ES_{new} is added to the evolution provenance graph. This is done using the inference rule *edge inference* described in Figure 4.3.

Finally, we point out that the evolution provenance graph is stored in a relation database alongside, the visualization resources discussed previously in Section 4.3.1.

4.3.4. Multi-user graph

The evolution provenance graph above describes the visual data exploration process of a single user. As we shall see when discussing our recommendation algorithms, a graph summarizing multiple exploration sessions made by multiple users can support recommendations based on collaborative-filtering.

We therefore define such summarized graph, called multi-user exploration graph next as follows.

Definition 4.5 (Multi-user exploration graph) *A multi-user exploration graph $G_{MU}(N_{MU}, E_{MU})$ is the result of merging many evolution provenance graphs $\{G_{XS1}, \ldots, G_{XSn}\}$. Formally, a multi-user exploration graph $G_{MU}(N_{MU}, E_{MU}) = \bigcup_{i=1}^{n} G_{XSi}$ where each $G_{XSi} = (N_{XSi}, E_{XSi})$ is an evolution provenance graph. G_{MU} has a surjective map M: $\bigcup_{i=1}^{n} G_{XSi} \longrightarrow G_{MU}$ such that:*

- $\forall n \in N_{XSi}, \exists\ n^* \in N_{MU}$ *such that* $M(n) = n^*$.
- $\forall e = (n_1, n_2, L) \in E_{XSi}, \exists\ e^* = (n_1^*, n_2^*, L^*) \in E_{MU}$ *such that the metadata of the two edges are similar, i.e.* $e.L[const] = e^*.L^*[const]$ *and we have two matchings:* $M(n_1) = n_1^*$ *and* $M(n_2) = n_2^*$.
- $\forall n_1 \in N_{XSi}$ *and* $n_2 \in N_{XSj}$, $M(n_1) = M(n_2)$ *implies* $i \neq j$.

The first two properties of the surjective map M ensures that each edge (similarly for node) in an input evolution provenance graphs G_{XSi} has necessary a corresponding edge (similarly for node) in the multi-user exploration graph G_{MU}. Finally, the third property of the surjective map M ensures that no two nodes belonging to the same input evolution provenance graph map to the same target node in G_{MU}.

In other words, the multi-user exploration graph of a collection of evolution provenance graphs is a single labeled directed acyclic graph where each node (edge) represents many nodes (edges), each belonging to a different input evolution provenance graph G_{XSi}. $|M^{-1}(n)|$ denotes the set of nodes in $\bigcup_{i=1}^{n} G_{XSi}$ that are mapped to a node $n \in N_{MU}$. The connectivities and the directions defined in each input G_{XSi} are maintained in the G_{MU}. The multi-user exploration graph follows the definition of evolution provenance graphs (see Definition 4.4). We merely adjust the definition of edge scores. For an edge $e = (n', n, L)$, we compute its score as $s(e) = 1 + |M^{-1}(n)| + \sum_{e_c=(n,n_i,L)\in E} s(e_c)$. Intuitively, $s(e)$ reflects the frequency of n (the number of exploration steps in the individual graphs that are mapped to n) as well as the number of exploration steps reached by this navigation e.

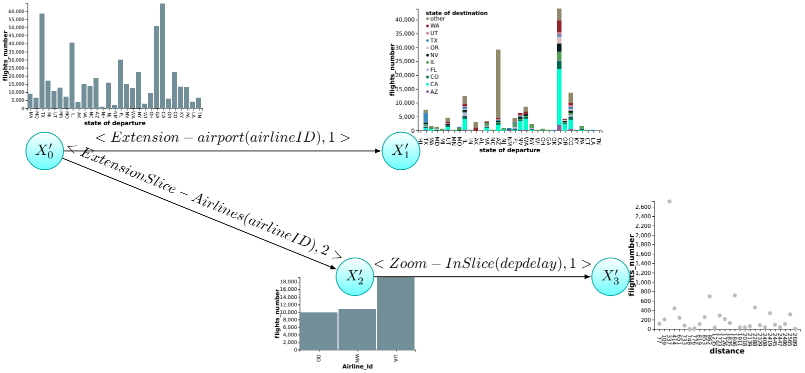


Figure 4.4.: Another example of an exploration session graph

Example 10 (Example of multi-user graph) *Consider another user exploring the same region of data studied in Example 9 using EVLIN the instance of our framework ProvVDE. The evolution provenance tracking the exploration session of this second user is depicted in Figure 4.4.*

Towards constructing a multi-user graph, we merge the evolution provenance graph shown in Figure 4.4 with the evolution provenance graph discussed in Example 9. Assume in this scenario that our merge approach fuses the following exploration step pairs $\{(X_0, X'_0), (X_1, X'_1), \text{and } (X_2, X'_2)\}$ *into nodes* X_{00}, X_{11}, *and* X_{22} *respectively. The resulting multi-user graph is depicted in Figure 4.5. Merged exploration steps* ES_{merged} *are highlighted in red. Accordingly, edges* e_{merged} *pointing to* ES_{merged} *are also updated (e.g., edge pointing to the node* X_1*). More precisely, we augment impact scores of* e_{merged} *to reflect the frequency of each item present in* ES_{merged}*. Consequently, scores of edges pointing to merged exploration steps* X_{11} *and* X_{22} *are increased.*

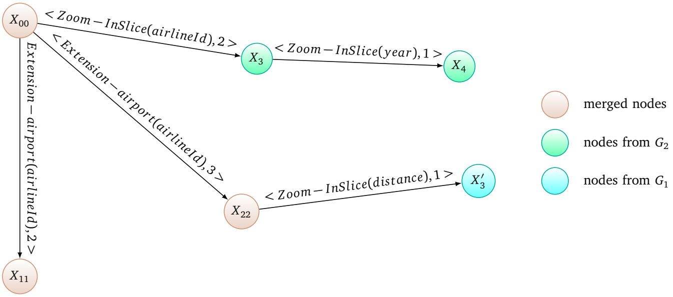

X_{00}
$< Zoom - InSlice(airlineId), 2 >$
X_3
$< Zoom - InSlice(year), 1 >$
X_4
$< Extension - airport(airlineId), 3 >$
$Extension - airport(airlineId), 2 >$
X_{22}
$< Zoom - InSlice(distance), 1 >$
X'_3
X_{11}
merged nodes
nodes from G_2
nodes from G_1

Figure 4.5.: Example of multi-user graph

4.4. Conclusion

In this chapter, we formalized the evolution provenance model adopted in our visual data exploration framework ProvVDE. Our novel evolution provenance model ensures the collection of visualization-related properties as well as query-related properties inferred through visual data exploration processes. It is worth stressing that the evolution provenance is the basis on which recommendations shall be computed in our framework. Accordingly, we show in the next chapter how such provenance can indeed be used for recommendations. This is done by devising algorithms specialized to the exploration of data stored in data warehouses.

Finally, we point out that our evolution provenance model was also adopted in the context of interactive analysis of dynamic graphs. Further details about the incorporation of our evolution provenance model in the context of dynamic graphs are available in [BBH+19].

Chapter 5

Provenance-based visual exploration of data warehouses

5.1. Introduction

So far, we have described the evolution provenance model supported in our framework. In this chapter, we discuss the *recommendation engine* module present in the architecture of our framework (cf. Figure 3.1). To do that, we specialize to a specific scenario, i.e., data warehouse exploration to leverage clear semantics of typical recommendations that can be suggested when exploring data warehouses. More specifically, we discuss the following novel provenance-based contributions implemented in EVLIN (the instance of our framework ProvVDE that specializes to the visual exploration of data warehouses).

- We propose a *content-based query recommendation algorithm* that leverages data provenance to guide users to different portions of the studied data sets.

- We propose a method to quantify recommendation quality. Given the high diversity and possibly large number of recommended queries produced by our content-based query recommendation approach, we propose to support users to navigate through the exploration space by quantifying the “interestingness” of recommended queries. The computed scores are then visualized in an interactive impact matrix, pointing users to potentially interesting recommendations to study next.
- We discuss several techniques to incrementally merge evolution provenance graphs, which describe individual user exploration sessions, into a global graph. The different approaches trade off merge efficiency and effectiveness. The proposed approaches are meant to analyze global trends adopted by users when exploring visually the same data.
- We propose a *collaborative-filtering method* for query recommendation that leverages the merged graph (discussed in the previous point) to promote recommendations widely inspected previously by users. Our collaborative-filtering query recommendation approach is blended with our content-based query recommendation approach to improve the quantification process of recommended queries by taking into account globally interesting trends in addition to (limited) local information.
- We outline a method that recommends visualizations for a query result, once a user has selected a (recommended) query. The goal of these recommendations is to provide visualizations that render the data appropriately and facilitate thereby the interpretation of results. This increases potentially the understandability and thereby the efficiency of visual data exploration.

In what follows, we discuss in Section 5.2, our content-based query recommendation approach. Section 5.3 covers our approach proposed to quantify the interestingness of recommended queries. Then, we discuss in Section 5.4 our collaborative-filtering query recommendation approach that aggregates evolution provenance and uses it to promote users' global trends. Furthermore, we describe also in the same Section 5.4 how we leverage the collaborative-filtering recommendation approach to improve the recommendation quantification process discussed in Section 5.3 by providing. Subsequently, we discuss in Section 5.5 our visualization recommendation approach. Alongside the description available in each aforementioned section, we review important existing work related to each contribution.

Note that the content of this chapter is based on methods and approaches described in [BH17; BH19b; BH20; BHBK18].

5.2. Content-based query recommendation

In this section, we discuss our content-based query recommendation approach for visual exploration of data warehouses. Essentially, our proposed *content-based query recommendation* approach guides users in their investigation of a data warehouse D by suggesting queries as next exploration steps, given an initial query Q.

Our proposed content-based query recommendation solution follows the pipeline shown in Figure 5.1 to identify the set of recommended queries that may interest the user. It takes as input an exploration query Q, a data warehouse D, and an interaction referring to a user's selection of a sub-result $r \in Q(D)$ (see Example 6, Page 42) to return ultimately a set of recommended queries.

In what follows, we give a glimpse of the main steps present in the pipeline depicted in Figure 5.1 before digging into details of each step.

In the first step, our content-based query recommendation process computes the data provenance of the sub-result r. This results in a data set that we call in what follows *Lineage*. This latter contains all tuples involved to get the sub-result r (see Section 2.4). After that, our content-based query recommendation process analyzes the *Lineage* to extract attribute-values pairs (a, L_v) (with a an attribute and L_v a set of values of the active domain of attribute a) that are strongly related to Q and the user's interaction r. Lastly,

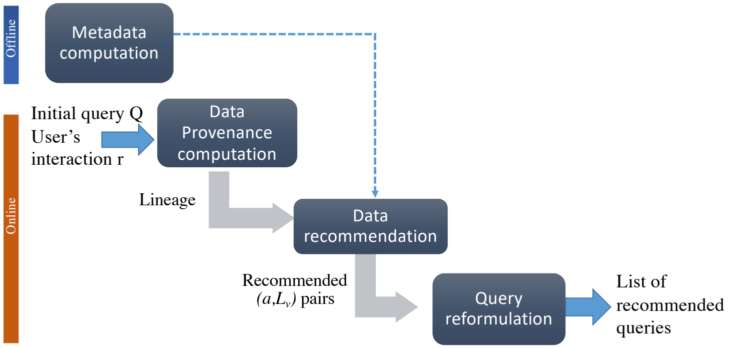


Figure 5.1.: Steps of the content-based query recommendation process

during the query reformulation phase, our content-based query recommendation approach leverages each returned (a, L_v) pair to derive recommended queries based on Q. All steps described so far are online. Hence, they are handled dynamically following user's interactions. We also incorporate an offline phase to our approach where we precompute or preconfigure information that is later accessed during the online computations. These information include the frequency for each attribute present in the explored dataset. Furthermore, we identify the set of functional dependencies present in the underlying dataset.

5.2.1. Data provenance computation

In the first step, we compute the data provenance of result r, denoted as $Lin(r)$. Based on a data region determined by a user interaction $r \in Q(D)$, we compute the data provenance using the Perm provenance management system [GA09]. This provenance $Lin(r)$ corresponds to all tuples in D that have contributed to producing r (i.e., why-provenance [CCT09]).

This lineage is input to the second phase of our content-based query recommendation approach.

Algorithm 1: Data recommendation algorithm

```
   Input: $Q$, $D$, $Lin(r)$, $\theta_L$, $\theta_{supp}$
   Output: $RMap$: data recommendations in form of a mapping
           associating attribute $a$ to value lists $L_V$
1  foreach $a \in schema(Lin(r))$ do
2      foreach $v \in adom(a)$ do
3          $f_{Lin} \leftarrow$ compute $f_{a,v}(Lin(r))$ using Equation 5.1;
4          if $f_{Lin} \geq \theta_L$ then
5              $f_D \leftarrow$ compute $f_{a,v}(D)$ using Equation 5.1;
6              $support \leftarrow supp_{a,v}(r)$ as defined by Equation 5.2 ;
7              if $support \geq \theta_{supp}$ then
8                  $RMap \leftarrow mapInsert(RMap, (a, v))$ ;
9  forall $a_i, a_j \in keys(RMap) \times keys(RMap), i \neq j$ do
10     if Functional dependency $a_i \rightarrow a_j$ holds then
11         $mapRemove(RMap, a_j)$;
12 Return $RMap$;
```

5.2.2. Data recommendation

Data recommendation is based on a novel provenance-based recommendation algorithm. It exploits data provenance to recommend attributes and values of these attributes that may raise user interest. Algorithm 1 shows its pseudocode.

The algorithm takes as input query Q, a data warehouse D, the lineage $Lin(r)$ of a sub-result r, and two threshold values θ_L and θ_{supp}. It outputs ultimately a set of pairs (a, L_v) where a is an attribute in the schema of the provenance $Lin(r)$ and $L_v = \{v_1, \ldots, v_n\}$ is a list of values available in the active domain of a denoted $adom(a)$.

Lines 1– 8 perform the provenance-based identification of relevant data by iterating over all attribute-value pairs (a, v) available in the lineage $Lin(r)$. Relevant attribute-value (a, v) are pairs that appear frequently enough in the provenance while this frequency significantly differs from the appearance frequency of (a, v) in the whole data warehouse D.

More formally, we compute the appearance frequency for each value v of each attribute a in $Lin(r)$ as

$$f_{a,v}(Lin(r)) = \frac{|\{t_i|t_i \in Lin(r) \land t_i.a = v\}|}{|Lin(r)|} \tag{5.1}$$

Subsequently, we consider only attributes that are frequent enough to have any significance w.r.t. user's initial selection, i.e., their appearance frequency $f_{a,v}$ should be greater or equal to a predefined threshold θ_L(see line 4). At this stage, we obtain already a list of interesting attribute-value pairs that we can use to compute recommended queries. Yet, this list of interesting candidates may contain attribute-value pairs that are mainly present in the explored dataset. These dominant attribute-value pairs will be always highly present in the lineage, regardless of the user's interaction. In turn, they will always be recommended regardless of the user's query. This breaches the rationale of recommendation that consists of recommending information strongly related to the user's focus. To cope with this problem, Equation 5.1 is further used in line 5 to compute the appearance frequency of the remaining candidates in the complete dataset (by replacing $Lin(r)$ by D). Using the two previous frequencies, we compute the support of each candidate in the provenance w.r.t. the whole explored data warehouse D as

$$supp_{a,v}(r) = \left| log_e \left(\frac{f_{a,v}(Lin(r))}{f_{a,v}(D)} \right) \right| \tag{5.2}$$

Intuitively, if the two considered frequencies significantly differ, the corresponding attribute-value pair is considered as potentially interesting. Thus, interesting attribute-value pairs are associated with a high support value and we only keep candidates with $supp_{a,v} \geq \theta_{supp}$ (line 7).

Overall, an attribute-value pair (a, v) is deemed interesting if (i) it is widely present in lineage $Lin(r)$ and more massively present in the explored data warehouse D, or (ii) it is widely present in lineage $Lin(r)$ but rarely present in the data warehouse D.

Accordingly, we define recommended attribute-value pairs as follows.

Definition 5.1 (Recommended attribute-value pairs) *Given a query Q on a data warehouse D, and a sub-result $r \in Q(D)$ of interest, an attribute-value pair (a, v) is recommended if $f_{a,v}(Lin(r)) \geq \theta_L$ and $supp_{a,v}(r) \geq \theta_{supp}$, where θ_L and θ_{supp} are predefined threshold values.*

Based on Definition 5.1, our proposed process of interesting attribute-value pair identification focuses mainly on the measure of candidates' frequencies discrepancy. Yet, it is easy to adapt Algorithm 1 to further distribution patterns or other models of interestingness.

The selected attribute-value pairs are added in line 8 to $RMap$ by calling $mapInsert(RMap, (a, v))$. Essentially, if a does not exist as a key in $RMap$, this function creates a new key-value pair that maps a to a set of values $L_V = \{v\}$. Otherwise, it adds v to the set of values readily existing for key a.

The attribute-value pairs grouped together by attribute in $RMap$ result in what we call data recommendations.

Definition 5.2 (Data recommendation) *Data recommendations are pairs (a, L_v) where a is an attribute, $L_v = \{v_1, \ldots, v_n\}$ is a list of values and it holds for all v_i that (a, v_i) is a recommended attribute-value pair.*

Example 11 (Data recommendation outcome example) *In our example (see Example 4, Page 41) that explores the set of delayed flights, assume the user has selected the highest bar that designates the delayed flights departing from the state California (see Example 6, Page 42). This triggers the computation of the why provenance of California. This lineage is in turn processed by the data recommendation step that outputs the following set of data recommendation pairs: $\{(airlineID, \{WN, UA, OO, AS\})$, $(airport,$ {San Diego International-Lindbergh, Metropolitan Oakland International, Los Angeles International, San Francisco International},(city,$\{San\ Francisco,\ San\ Diego,\ Oakland,$ $Los\ Angeles\})\}$.*

Using our data recommendation step described above, it is possible that two distinct entries in recommended pairs $RMap$, i.e., (a,v) and (a',v') yield redundant query reformulations later.

This occurs when functional dependencies exist between attributes present in $RMap$. Indeed, as we will show in the next section, our query reformulation relies mainly on recommended pairs to reformulate user's initial query Q. Accordingly, if a user inspects currently $Q(D)$, several data regions will be subsequently recommended such as $Q_1 : \sigma_{a=v}(Q(D))$ and $Q_2 : \sigma_{a'=v'}(Q(D))$ given recommended pairs $\{(a,v),(a',v')\} \subseteq RMap$. Assume in this case that there is a functional dependency between a and a' such that $a \rightarrow a'$. Then, we know that whenever a has value v, a' has a value v'. So there does not exist any tuple in the relation $R(a,a')$ such that $a = v$ and $a' \neq v'$. The recommended query using a, $Q_1 : \sigma_{a=v}(Q(D))$ returns then all tuples with values (v,v') in R. Conversely, it is possible for two distinct tuples in R with $a' = v'$ to have different values than v on attribute a. So tuples in the result of the recommended query using a', $Q_2 : \sigma_{a'=v'}(Q(D))$ includes both all (v,v') from before as well as additional tuples (if present). So $\sigma_{a=v}(Q(D)) \subseteq \sigma_{a'=v'}(Q(D))$. Consequently, we are recommending redundant data regions.

To avoid redundant recommendations, we employ data profiling algorithms to determine functional dependencies [AGN15] of the form $a \rightarrow a'$ and prune a'. More specifically, in lines 10 – 12 of Algorithm 1, we determine whether a functional dependency $a_i \rightarrow a_j$ between two attributes from the computed data recommendations holds. If so, we only retain the data recommendation involving a_i.

Example 12 (Functional dependency-based filtering) *Continuing Example 11, the data recommendation process reveals three interesting attribute-values pairs. The computation of functional dependencies reveals a dependency between attributes "city" and "airport" such that $airport \rightarrow city$. To this end, we filter out the attribute-values pair $(city, \{$San Francisco, San Diego, Oakland, Los Angeles$\})$. This leads to the following set of data recommendations $\{(airlineID, \{WN, UA, OO, AS\})$, $(airport, \{$San Diego International-Lindbergh, Metropolitan Oakland International, Los Angeles International,San*

Francisco International}) *that will subsequently be used to generate recommended queries.*

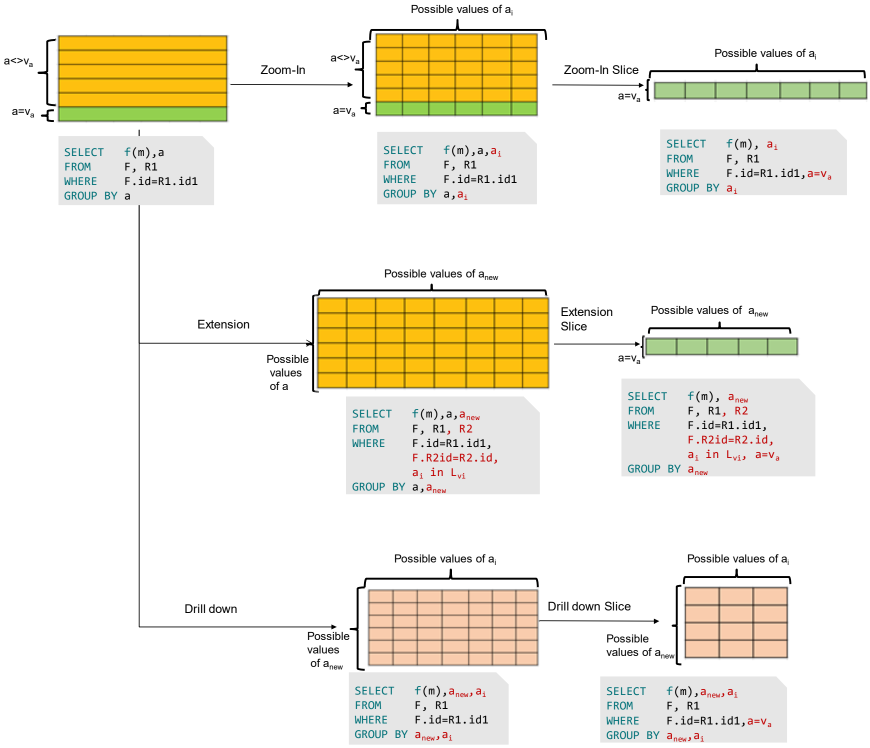
Overall, the complexity of Algorithm 1 that performs the data recommendation step is dominated by the search of interesting attribute-values. Accordingly, Algorithm 1 is performed in O($|schema(Lin(r))| \times ð$) with $|schema(Lin(r))|$ is the size of the lineage schema (number of columns present in the lineage) and $ð = \max_{i=1}^{|schema(Lin(r))|} |adom(i)|$ is the maximum active domain size of attributes present in the lineage.

Later, we study in Section 6.3, the impact of the lineage size and domain size of columns on the performance of our data exploration process. Overall, experiments (presented later in Section 6.3) show that domain size and lineage size impact the runtime and the final number of interesting attribute-values output by the data recommendation step. Yet, our evaluation results show that the content-based query recommendation is still fast enough for an interactive visual data exploration process.

5.2.3. Query reformulation

The data recommendations produced by the previous step are input to the query reformulation step where we compute, for each (a, L_v), a set of queries corresponding to variations of the initial user's query Q. Each variation reflects a particular operation typical when querying data. The variations that we propose are targeted towards covering the whole data space of the explored data warehouse D, allowing to reach and explore "unknown territory", i.e., data initially not related to $Q(D)$. At the same time, they follow well known data warehouse query patterns to facilitate users' tasks when exploring data.

More specifically, we consider the query types described below. Given an initial query Q (that respects the format given in Definition 3.1), its associated visualization V, and a user's interaction via V of a sub-result $r_i = (a, v_a) \subseteq Q(D)$, our query reformulation process uses an interesting attribute-values pair $\left(a_i, L_{v_i}\right)$ output by the data recommendation process, to



```
SELECT   f(m),a
FROM     F, R1
WHERE    F.id=R1.id1
GROUP BY a
```

```
SELECT   f(m),a,a_i
FROM     F, R1
WHERE    F.id=R1.id1
GROUP BY a,a_i
```

```
SELECT   f(m), a_i
FROM     F, R1
WHERE    F.id=R1.id1,a=v_a
GROUP BY a_i
```

```
SELECT   f(m),a,a_new
FROM     F, R1, R2
WHERE    F.id=R1.id1,
         F.R2id=R2.id,
         a_i in L_vi
GROUP BY a,a_new
```

```
SELECT   f(m), a_new
FROM     F, R1, R2
WHERE    F.id=R1.id1,
         F.R2id=R2.id,
         a_i in L_vi, a=v_a
GROUP BY a_new
```

```
SELECT   f(m),a_new,a_i
FROM     F, R1
WHERE    F.id=R1.id1
GROUP BY a_new,a_i
```

```
SELECT   f(m),a_new,a_i
FROM     F, R1
WHERE    F.id=R1.id1,a=v_a
GROUP BY a_new,a_i
```

Figure 5.2.: Templates of query derivations used for content-based query recommendation

derive and recommend queries whose templates are depicted in Figure 5.2. For ease of presentation but without loss of generality, we assume here that Q involves only two tables, i.e., joins the fact table F with one dimension table R_1, and aggregates results by groups defined by one attribute a. As we see on the left of Figure 5.2, this results in a one-dimensional “array” of aggregated values. This also means that any tuple in $Q(D)$ has exactly two attributes, in particular including user’s interaction r_i (attribute a with a value v_a). In Figure 5.2, we distinguish these by color from tuples where the value of attribute a is different from v_a.

Zoom-in / zoom-in slice. This type of query retains the query schema of the original query Q, i.e., the FROM clause remains unchanged. However, zoom-in adds the attribute a_i, that by definition is among the attributes in the schema of Q, as a second dimension to the aggregated values (by adding a_i to the SELECT and the GROUP BY clauses). This query type may be interesting for further exploration when an analyst wants to compare the contribution of specific a_i values to the the selected sub-region r_i to the values' contribution in the remaining dataset $Q(D) \setminus r_i$.

When zoom-in is combined with a slice, the focus is set to the selected region r_i, by adding the condition $a = v_a$ to the WHERE clause of Q.

Extension / extension slice. This query type extends the data presented to the analyst to another dimension of the data warehouse. That is, another dimension table $R_2 \in A$ currently not in the schema of Q is added to the FROM clause, and the corresponding join condition is added to the WHERE clause. Let a_{new} be the attribute of R_2 with coarsest granularity. This attribute defines the second dimension we consider for our aggregated values, i.e., a_{new} is added to the SELECT and GROUP BY CLAUSE. To reflect the focus on a recommended (a_i, L_i)-pair, we further add the condition $a_i\ IN\ L_i$ to the WHERE clause. An extension query is created for each dimension not included in the schema of Q. The rationale behind this query type is to support the analytical task of investigating how interesting attribute values of the current view on the data relate to yet unexplored data of another dimension.

As before, when combining extension with slice, we add the user's interaction $a = v_a$ to the WHERE clause of the extension query, allowing analysts to focus in detail on the extension wrt to the data region r_i they previously selected.

drill-down (roll-up) / drill-down (roll-up) slice. This query type navigates along the dimension of attribute a to either move to a more detailed granularity in the dimension hierarchy of a in the case of drill down, or to a coarser granularity in case of a roll up. Let a_{new} be the attribute in the dimension hierarchy of a with more detailed (coarser) granularity. To perform

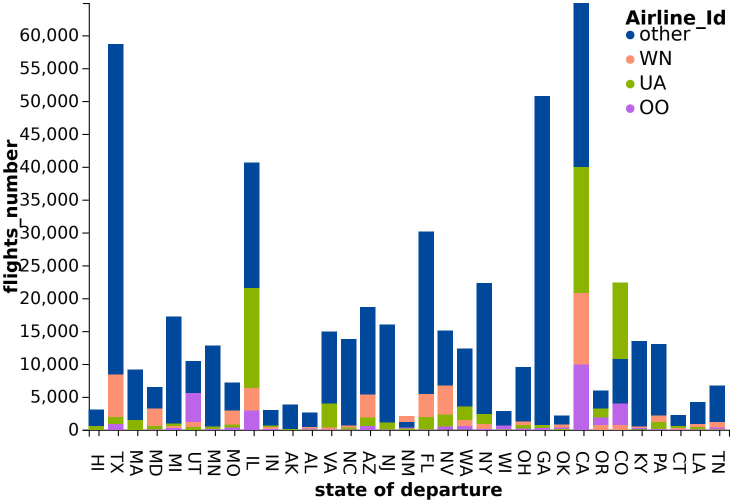


Figure 5.3.: Sample recommended query result

the drill-down (roll-up), a is replaced by a_{new} in Q. As the recommendation algorithm has identified potentially interesting values in the schema of R, we further group the data by a_i. This type of recommendation allows analysts to relate how attribute values of potential interest at the currently explored granularity impact higher (lower) levels of granularity.

When combining drill-down with slice, we further add the user's interaction $a = v_a$ to the WHERE clause. This allows to focus on the different values arising at higher granularities of the user-selected data region.

Example 13 (Sample recommended query) *For our running example (discussed in Examples 4, 6, and 12), the* content-based query recommender *generates a recommended query of type "Zoom-In" associated to the pair* $\{(airlineID, \{WN, UA, OO\})$*. This recommended query studies the distribution of delayed flights performed by different airline companies when departing from several states. This recommended query is defined as follows.*

SELECT count(*) , A.state , F.airlineID FROM Flights as F *,Airports* as A *WHERE A.iata=F.origin* AND $F.depdelay > 0$ *GROUP BY A.state , F.airlineID*

The visualization of this recommendation result, generated by the visual recommender*, is depicted in Figure 5.3. This visualization shows that airlines* $\{WN, UA, OO\}$ *have more delayed flights when departing from California than when departing from any other state.*

Based on the content-based query recommendation approach, we return several recommendations related to the user's current focus. Later, we explain how these recommended queries are ranked to offer users some help in exploring the data warehouses more efficiently.

5.2.4. Related work

We have already highlighted the set of existing visual data exploration systems providing query or visualization recommendations in Section 2.2.2. In this section, we pay particular attention to state-of-the-art visual data exploration solutions that provide content-based query recommendations. This concerns several existing work including for instance [ESC18; MSK18; SK16b; THY+17; VRM+15; WMA+16; WQM+17]. Overall, these existing visual data exploration approaches analyze initial user's query results, i.e., $Q(D)$, to find data items or sub-regions of data highly interesting to the context of the user. For instance, [ESC18; MSK18; SK16b; VRM+15] adopt deviation to recommend data regions present in the user's initial query's result $Q(D)$. Similarly, authors in [THY+17] analyze data cubes specified by users and search subspaces that encompass outliers and trends. Voyager [WMA+16; WQM+17] relies on the set of attributes initially selected by the user to recommend the set of exploration queries strongly related to the user's initial selection.

Table 5.1 provides an overview about existing visual data exploration work that supports content-based query recommendation. This tables encompasses also our solution EVLIN. For each approach, it describes (i) the supported types of input queries (e.g., select-project-join (SPJ) queries, project-aggregate (PA) queries, or cube queries corresponding to SPJA queries) and (ii) the type of recommended output queries. We observe that existing

System	Input query	Recommended query
Ziggy [SK16b]	SJ	SPJ
SeeDB [VRM+15]	cube	sub-cube
Muve [ESC18]	cube	sub-cube
Dive [MSK18]	cube	sub-cube
[THY+17]	cube	sub-cube
Voyager [WMA+16; WQM+17]	SPA	Changed SELECT-clause
EVLIN [BHBK18]	cube	OLAP queries

Table 5.1.: Expressiveness of visual data exploration solutions supporting content-based query recommendation

solutions such as [ESC18; MSK18; SK16b; THY+17; VRM+15; WMA+16; WQM+17] offer various types of recommended queries. Yet, these work have only very limited support for query recommendation expressiveness. Typically, there is a one to one mapping between the recommended query and a particular type of query e.g., SPJ (adopted in [SK16b]) or sub-cubes e.i, slices of cubes (adopted in [ESC18; MSK18; THY+17; VRM+15]). In comparison, our solution EVLIN, allows a more expressive data exploration by supporting the set of widely accepted OLAP queries including Zoom In, Drill, and Slice. Thus, the recommendations methodologies supported in our system cover the whole data space of the explored dataset, allowing to reach and explore “unknown territory”, i.e., data initially not related to the user query.

5.3. Quantification of recommendations interestingness

The query recommendation algorithm presented in the previous section returns a set of queries, which can in general become large enough to overwhelm users. To guide users choosing which recommendations to inspect next we rank recommendations based on their interestingness, as discussed in this section.

We introduce first in Section 5.3.1 the utility metric used to quantify recommendations. To make its computation fast enough for an interactive exploration process, we present our proposed optimizations in Section 5.3.2.

5.3.1. Quantification of recommended query interestingness

Our quantification approach for the interestingness of recommended queries leverages *the deviation metric* to compute the utility of recommended queries. This metric computes the divergence between two data regions. In our context, these two data regions are (i) the result of a recommended query $Q_{rec}(D)$, and (ii) the full explored data warehouse, denoted D. Accordingly, as the deviation between the two data regions is more significant, the recommended query is considered more interesting as it has a data distribution different from those of the reference data region.

This metric was first proposed in [VRM+15], where authors demonstrated that the deviation metric provides users with interesting recommendations. Subsequently, this metric was adopted in many work including for instance [ESC18; LDH+19; SK16a]. That is, our work follows up on previous work using deviation metrics for query recommendations.

Traditionally, the problem of quantifying deviation between distributions is transformed to a distance computation between histograms representative of data distributions. Similarly, we generate histograms associated with the recommended query $Q'(D)$ (under quantification) and the data warehouse D. Accordingly, we compute the distance between the two histograms (representative of data distributions of two regions) to measure the deviation between the two queries' results.

Puzicha et al [PBRT99] surveyed existing methods that could be employed to compute the distance between histograms. This results in a classification of distance functions in four main categories. These classes as well as their prominent implementations are summarized in Table 5.2.

The first class is called *heuristic-based distance*. It computes distance between two data distributions by performing a bin by bin comparison between their associated histograms. Examples of implementations following this class include the Manhattan, Euclidian, and Chybyshev Distances.

The second class comprises the *non parametric test statistics*. These measure the maximum discrepancy between cumulative distributions. Many distance functions (e.g., The Kramer von Mises and Chi square) fit in this class.

Information theoretic divergence is the third class. It measures how compact one distribution can be coded using the other distribution. The Kullback Leibler divergence and the Jeffrey distance are examples of implementations fitting in this class.

Finally, the *Ground distance measure* presents the fourth distance class. It incorporates the cross-bin comparison to compute the minimal cost needed to transform one distribution to the other. Examples of functions belonging to this class include Quadratic form and Earth Mover's Distance.

Later in Chapter 6, we study how a representative of each class performs when quantifying the interestingness of recommended queries generated using our content-based query recommendation approach. More specifically, we implement and evaluate the following functions: (i) Euclidean distance (L2) that belongs to the *heuristic-based distances* class; (ii) Chi square (CS) that is an example of the *non parametric test statistics* class; (iii) Kullback Leibler (KL) that presents an implementation of the *information theoretic divergence* class; and (iv) Earth Mover's Distance (EM) that belongs to the *ground distance measures* class.

5.3.2. Computation of the interestingness scores of recommended queries

So far, we have described the process of query recommendation. This latter is reinforced by a process of quantification that assigns a utility for each recommendation. Overall, the exploration step conduct within our system EVLIN is depicted in Figure 5.4. First, the user inspects using EVLIN at each

Category	**Examples of distance functions**
Heuristic-based distances	Manhattan L_1, Euclidian L_2, Minkwoski L_p, Chybyshev Distance
Non parametric test statistics	The Kolmogorov Smirnov distance, Kramer von Mises, Chi square
Information theoretic divergence	The Kullback Leibler divergence, The Jeffrey divergence
Ground distance measure	Quadratic form , Earth Mover's Distance

Table 5.2.: Classification of some existing dissimilarity/distance functions

exploration step an exploration query Q of the form

$\texttt{SELECT}\ a_i, f(m)\ \texttt{FROM}\ rel(D)\ \texttt{WHERE}\ cond\ \texttt{GROUP BY}\ a_i$

where $m \in M$, $a_i \in schema(A)$, $f \in F$, $rel(D)$ refers to one or more relations in D, and $cond$ is a (conjunction of) predicates.

This corresponds to the first step of the exploration step conduct depicted in Figure 5.4.

To trigger the computation of recommendation, users select a sub-result $r \in Q(D)$ of interest. This results in the identification of the set of interesting attribute-values pairs that are used to derive recommended queries from the current explored query Q. This corresponds to the second step depicted in Figure 5.4.

Consequently, quantifying the interestingness of a considerable number of recommended queries is the third step depicted in Figure 5.4. Recall that such quantification of the recommendation process needs to be re-computed after each user's interaction during the inspection of an exploration step. Indeed, both the number of recommended queries as well as the complexity of the interestingness score computation for each query is a complex and time consuming computational task.

More formally, assume that n is the number of relevant attribute-values for recommendations and k is a specific number of query types adopted by the query reformulation process (see Section. 5.2.3). Then, we have $k * n$ recommended queries.

Let $\beta = \max_{i=1}^{c} |dom(i)|$ be the maximum domain size of any column c in the set of columns referred to in any of the $k * n$ recommended queries. Then, the cost of quantifying the interestingness of all recommended queries is in

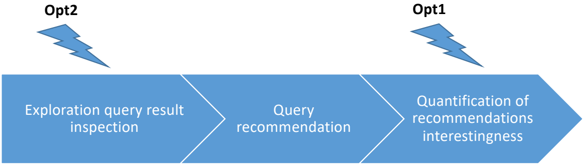

Opt2
Opt1
Exploration query result inspection
Query recommendation
Quantification of recommendations interestingness

Figure 5.4.: Conduct of an exploration step

$O(k * n * \beta^2)$.

Given the above problem, we propose the following two optimizations (depicted in Figure 5.4) meant to reduce the overall runtime in practical use cases.

Opt1: Minimum shared regions computation. Inspired by Agrawal et al [ACN00], we determine a minimum set of materialized views to precompute data regions that are frequently accessed when computing recommendation scores. Essentially, we construct in the third step of the exploration step (cf. Figure 5.4) two materialized views MV_1 and MV_2. These two materialized views cover all data regions of recommendations. They thereby reduce the number of transactions required to perform when computing the interestingness of each recommendation.

The two materialized views MV_1 and MV_2 are defined as follows.

MV_1 : `SELECT * FROM` $rel(D)$ `WHERE cond`

MV_2 : `SELECT * FROM` $rel(D)$ `WHERE cond AND` r

where $rel(D)$ refers to one or more relations present in the user's query Q, $cond$ is a (conjunction of) predicates present also in Q, and r is the sub-result of $Q(D)$ selected by the user.

In relation to EVLIN's recommendation templates specified in Figure 5.2, we construct MV_1 to compute later data regions corresponding to all recommendations of types “Zoom-In”, “Extension” and “Drill-down” while MV_2 is constructed to derive data regions corresponding to recommended queries of types “Zoom In Slice”, “Extension Slice” and “Drill-down Slice”.

Opt2: Eager computation. To compensate for the overhead incurred by the materialization of shared data regions, we propose to proactively compute the data region of interest that is used subsequently to quantify a wide range of recommended queries later during exploration. For instance, the definition of MV_1 described above shows that this view is independent of the user's interaction. Accordingly, we compute proactively this data region at the first step of Figure 5.4, before the user asks explicitly to get recommendations by clicking on such interesting sub-result.

The impact of these two optimizations on recommendations' quantification runtime is evaluated in Chapter 6. Overall, evaluation results show that our optimizations are effective and they lead to a significant improvement of runtime up to 80% of the recommendations quantification runtime. This makes the computation of recommendations' interestingness scores fast enough for an interactive visual data exploration process.

5.3.3. Visualization of quantified recommended queries

Once the interestingness scores of recommended queries are computed using our proposed methods, we need to communicate this information to help users in selecting interesting recommendations to study further.

We choose to adopt a matrix visualization to render the set of scored recommended queries output by EVLIN. We use the term "impact matrix" to refer to this visualization in what follows. Recall that we have already presented in relation to our running example a sample of impact matrix (cf. Example 7, Page 43).

Each line of the matrix corresponds to an (a, L_V)-pair identified in the content-query recommendation approach as relevant information of recommendation (cf. Example 7). Each column corresponds to a type of query variation (presented in Section 5.2.3). Finally, the cell colors encode the deviation value. As a reminder, as the deviation value increases, the recommendation is considered more interesting as it has a data distribution significantly different from the distribution present in the whole dataset. Accordingly, the interestingness scores are mapped to a sequential scale of the red color that contains various intensities where the bright red color maps to a deviation value close to 0 whereas the dark red color refers to an interesting recommendation having a deviation value close to 1.

From the example of impact matrix depicted in Figure 3.4, we see that several recommended queries stand out by having high interestingness scores, suggesting these as worth exploring next.

5.3.4. Existing interestingness metrics

Several visual data exploration work have proposed recommendation approaches to support users in their exploration tasks. These recommendation methods leverage diverse metrics to quantify the interestingness of candidates for recommendation. We distinguish two metric classes that are surveyed in [BVW03; GH06].

The first class concerns *objective* metrics. These are based only on data and they correspond to quantitative metrics that can be directly measured. Examples of objective metrics include the *relevance* of the recommendation with respect to the exploration or to user's exploration goals.

This metric is applied for example in Ziggy [SK16b] which relies on a meet-in-the-middle approach between dissimilarity (between recommended views and the database) and diversity (among the recommendations set). Zenvisage [SKL+16] also adopts diversity as well as the strength of trends to quantify the interestingness of recommendations.

The *Novelty (or deviation)* is another objective metric that has gained substantial attention in the quantification of recommendations in visual data exploration research. This term was defined in [KB16]. Indeed, it expresses the discrepancy between a recommendation candidate and a target (or a reference) data set. The deviation metric is one implementation of novelty. Hence, it was commonly adopted in works [ESC18; LDH+19; MSK18; VRM+15]. Furthermore, the capability of the deviation metric to reveal interesting recommendations was demonstrated in [VRM+15]. As we have seen, we also adopt the deviation metric to quantify the interestingness of query recommendations in our work. To incorporate successfully this metric, we provide an efficient computation method that is capable to return results in acceptable time and to ensure thereby an interactive visual data exploration process.

The second class of metrics used to quantify the interestingness of candidates for recommendation comprises of *subjective* metrics. It considers both the data and the user. Several subjective measures were proposed to quantify interestingness of recommendations. For instance, Dejdaini et al [DDL+19; DLMP17] employ two metrics to quantify the interestingness of queries susceptible to recommendation. The first metric is called *focus*. It indicates the degree of coherence between the candidate of recommendation and other queries already explored. The second metric is called *contribution*. It computes the potential knowledge (or benefit) expected to gain, when recommending the underlying query. Another subjective metric was proposed in [THY+17]. It adopts a significance measure that computes uncommonness of a recommended data cube. More specifically, this metric computes the uncommonness of the data cube distribution compared to an expected distribution. Similarly, Sarawagi et al [Sar00] adopt a surprise factor as a metric to estimate the interestingness of not visited data regions. While the adoption of subjective measures may improve the quantification process that we have adopted in our setting, they are left for future research.

5.4. Computing query recommendations based on collaborative-filtering

So far, the query recommendation approach leverages data provenance to make content-based recommendations. In this section, we now present how to further exploit our evolution provenance to incorporate collaborative-filtering based recommendations.

For that, we give in the Section 5.4.1 a brief overview about our proposed collaborative-filtering query recommendation. After that, we describe in Section 5.4.2 how we generate an evolution provenance graph that results from the aggregation of many evolution provenance records. Then, we discuss in Section 5.4.3 our proposed collaborative-filtering query recommendation process. Furthermore, we discuss related work in Section 5.4.4.

Finally, we point out that the content of this section is mainly based on methods and approaches described in [BH19b; BH20].

5.4.1. Overview of collaborative-filtering recommendation approach

In this section, we leverage the evolution provenance already collected and stored as exploration session graph in order to incorporate collaborative-filtering query recommendation into our system EVLIN. To this end, we propose in Section 5.4.2 a set of merge methods that generate a multi-user exploration graph (cf. Definition 4.5) that summarizes multiple exploration session graphs.

The algorithms merge an exploration session graph G_{XS} into the (typically larger) multi-user graph G_{MU} maintained by our system EVLIN.

Later, our collaborative-filtering recommendation approach, detailed in Section 5.4.3, considers G_{MU} to compute query recommendations potentially interesting for a given (current) exploration step $X = \{Q, V\}$. Note that the process of quantifying recommendations' interestingness (described in Section 5.3) also requires slight modifications to take into account the results of the collaborative-filtering recommendations. These minor changes are introduced (as discussed in Section 5.4.3) to compute an overall score for the recommendations displayed to the user.

5.4.2. Merging of evolution provenance graphs

We discuss in this section the process of evolution provenance merge to obtain a multi-user exploration graph. Essentially, the merge of evolution provenance graphs takes as input an exploration session graph G_{XS} and the current version of the multi-user graph G_{MU}, determines a one-to-one matching between the set of vertices (more precisely, the queries of exploration steps), and finally merges matching nodes. Note that for simplicity, we slightly abuse the notation and consider that nodes of both graphs represent queries only.

More formally, during the matching phase, given the set of exploration steps modelled by vertices $N \in G_{XS}$ and $N_{MU} \in G_{MU}$, we determine a one-to-one matching $\mathcal{M}$ between nodes $n_i \in N$ and $n_j \in N_{MU}$ such that for each $m = (n_i, n_j) \in \mathcal{M}$, $sim(n_i, n_j) \geq \theta_{sim}$ (with θ_{sim} is a similarity threshold). The merge phase then produces a multi-user graph G'_{MU} with $N'_{MU} = N_{MU} \cup_{\approx} N$ and $E'_{MU} = E_{MU} \cup_{\approx} E$. The symbol $\cup_{\approx}$ denotes that the usual equality considered by set union is extended to also consider similar objects as "equal".

We first discuss the matching function that we use in Section 5.4.2.1. Remaining paragraphs in this sub-section cover different variants of applying this measure to obtain a merged multi-user graph. These variants trade off runtime efficiency with the achieved merge rate.

5.4.2.1. Measuring the similarity of exploration steps

The similarity sim of two exploration steps with respective queries Q and Q' over the same dataset D quantifies the extensional overlap of $Q(D)$ and $Q'(D)$. This leaves open many possibilities of defining a similarity measure. Equation 5.3, that is based on the Jaccard coefficient $Jaccard(S,S') = \frac{|S \cap S'|}{|S \cup S'|}$, presents one possible implementation of a similarity measure that we adopt. The similarity measure in Equation 5.3 uses three functions to extract token sets of each SQL-clause of an exploration query Q (see Definition 3.1):
$getSelect(Q) = \{a_1, \ldots, a_n, f(m)\}$,
$getTables(Q) = \{t | t$ referred to in $rel(D)\}$, and
$getConditions(Q) = \{p_i | p_i$ a predicate in $cond\}$. Then:

$$\begin{aligned} sim(Q_1,Q_2) &= \alpha \times Jaccard(getSelect(Q_1), getSelect(Q_2)) \\ &+ \beta \times Jaccard(getTables(Q_1), getTables(Q_2)) \\ &+ \gamma \times Jaccard(getConditions(Q_1), getConditions(Q_2)) \end{aligned} \tag{5.3}$$

where α, β, γ are weights, $\alpha + \beta + \gamma = 1$. By default, we use equal weights for all clauses.

While other, possibly more sophisticated measures are conceivable, the optimizations presented in Section 5.4.3 for efficiently finding queries to be recommended (in an online, i.e., time-critical fashion) are specific to the similarity measure based on Jaccard coefficient.

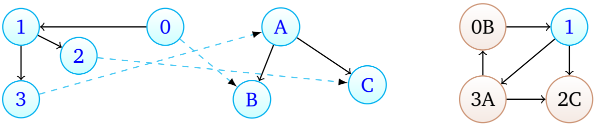


Figure 5.5.: Stable marriage application on exploration sessions [BH19b]

5.4.2.2. Obtaining a global matching of exploration steps

Conceptually, we can compute the similarity of all pairs of exploration steps in $N \times N_{MU}$ and retain only those pairs with a similarity above threshold θ_{sim} as candidate matches. Based on these, we can select a one-to-one matching $\mathcal{M}$ following several possible interpretations, e.g., stable marriage [Irv94]. However, it is easy to show that obtained matches may lead to a merged multi-user graph that includes cycles, which violates our definition of evolution provenance graph (cf. Definition 4.4, Page 52) and also the rationale of a recommended exploration step. For instance, Figure 5.5 shows, on the left, a possible matching (rendered as dashed edges) between two DAGs as determined by stable marriage [GS62]. The merged graph, resulting from merging nodes labeled 0 with B, 2 with C, and 3 with A (highlighted in red) is depicted on the right hand side. Observe that in the merged graph, we now have a cycle between nodes (0B), (1), and (3A).

Note that our merge problem also aims at generating a compact merged graph that encompasses less redundancy. The reduction of redundancy is important to avoid recommending similar explorations while the compactness of the merged graph is important to ensure a fast traversal of the graph when computing collaborative-filtering recommendations.

From these observation, we introduce the problem of merging evolution provenance graphs as follows:
Given two evolution provenance graphs $G_1 = (N_1, E_1)$, $G_2 = (N_2, E_2)$, a similarity function $sim()$, and a similarity threshold θ_{sim}, find the set of 1:1 matchings $\mathcal{M}$ such that

- for each $(n_i, n_j) \in \mathcal{M}$, $sim(n_i, n_j) \geq \theta_{sim}$ with $n_i \in N_1$ and $n_j \in N_2$

- the following equation is maximized

$$\sum_{i=0}^{|\mathcal{M}|} sim(\mathcal{M}_i.n_{i1}, \mathcal{M}_i.n_{i2}) \tag{5.4}$$

  with $n_{i1} \in N_1$ and $n_{i2} \in N_2$, $|\mathcal{M}|$ is the number of matching pairs and $sim(\mathcal{M}_i.n_1, \mathcal{M}_i.n_2)$ returns the similarity value of a matching pair $\mathcal{M}_i$.

- the merged graph $G_{merge} = (N_{merged}, E_{merged})$ is a directed acyclic graph that follows the Definition 4.5 (Page 56). More specifically, each matched pair of nodes are replaced by a merged node in the merged graph. Edges pointing to (and outgoing from) matched nodes are in turn added to the merged graph and rerouted to point at (stem from) merged nodes.

In relation to our problem, we review state of the art and we find the 1-1 homomorphism problem introduced in [FLM+10] where a graph $G_1 = (N_1, E_1)$ is said to be 1-1-homomorphism to $G_2 = (N_2, E_2)$ w.r.t. a similarity function sim() and a similarity threshold θ_{sim}, if there exists an injective mapping $\mathcal{M}$ from N_1 to N_2 such that for each node $n_i \in N_1$,

- if $\mathcal{M}(n_i) = n_j$ (with $n_j \in N_2$), then $sim(n_i, n_j) \geq \theta_{sim}$; and
- for any distinct nodes n_1, n_2 in G_1, $\mathcal{M}(n_1) \neq \mathcal{M}(n_2)$.
- for each edge (n_i, n_i') in E_1, there exists a nonempty path $n_j/.../n_j'$ in G_2 such that $\mathcal{M}(n_i) = n_j$ and $\mathcal{M}(n_i') = n_j'$ (with $n_j, n_j' \in N_2$), i.e., each edge from n_i is mapped to a path outgoing from n_j.

Overall, we are tackling a problem that is highly similar to the 1-1 homomorphism problem introduced in [FLM+10]. One main difference between the two problems concerns edge mapping. Indeed, our merge problem considers a more strict policy when mapping edges. Hence, if $(\mathcal{M}(v), \mathcal{M}(v'))$ is an edge in G_2, then (v, v') must be an edge in G_1; in contrast, 1-1 homomorphism only requires edges from G_1 to find a matched path in G_2. Finally, we point out that our merge problem considers the format of the merged graph resulting from G_1 and G_2. Indeed, our problem returns a DAG.

In contrast, authors in [FLM+10] have not discussed the DAG property of the graph resulting from matching the two DAGs G_1 and G_2 using the 1-1 homomorphism solution.

Accordingly, we provide below a proof that the 1-1 homomorphism solution produces a DAGs as a result.

Proof 1 ***Assumption***: *Suppose that $\mathcal{M}$ is a mapping solution of the 1-1 homomorphism problem when comparing two DAG graphs G_1 and G_2. More specifically, $\mathcal{M}$ finds two subgraphs $G_{sub_1} \subseteq G_1$ and $G_{sub_2} \subseteq G_2$ such that G_{sub_1} is homomorphic to G_{sub_2}. Accordingly, G_{sub_1} and G_{sub_2} are merged based on the 1-1 homomorphism solution $\mathcal{M}$. Assume that the merge of G_{sub_1} and G_{sub_2} produces a graph GM with a cycle c.*
$\Longrightarrow$ there exists a cycle c in GM such that we have two paths $p = y/.../x$ and $p' = x/.../y$ with $p, p' \in GM$.
$\Longrightarrow$ recall that G_{sub_1} and G_{sub_2} are DAGs. Then $p = y/.../x$ is not in G_{sub_1} when $p' = x/.../y$ is in G_{sub_1} with $x \in G_{sub_1}$ and $y \in G_{sub_1}$.
$\Longrightarrow$ there exists a path $p_2 = B/.../A$ in G_{sub_2} with $B \in G_{sub_2}$ and $A \in G_{sub_2}$ such that $\mathcal{M}(x) = A$ and $\mathcal{M}(y) = B$.
$\Longrightarrow$ recall that G_{sub_1} is homomorphic to G_{sub_2}, we have $\mathcal{M}(x) = A$, $\mathcal{M}(y) = B$ and we have a path $x/.../y$ in G_{sub_1}. Then based on the definition of 1-1 homomorphism solution, we conclude there is a nonempty path $A/.../B$ in G_{sub_2} such that each edge $e \in x/.../y$ has a mapping in the path $A/.../B$.
$\Longrightarrow$ We have two contradictory information. There is a nonempty path $A/.../B$ in G_{sub_2} (concluded in the third step of our proof) and there is a nonempty path $B/.../A$ in G_{sub_2} (see the previous step). This means that G_{sub_2} is not DAG. This breaches our assumption of DAG properties of G_1 and G_2.

Our proof ensures that a 1-1 homomorphism solution returns a DAG merged graph. This validates our observation that states that our studied problem is similar to the 1-1 homomorphism problem. Note that authors in [FLM+10] prove that the 1-1 homomorphism problem is NP-complete. Based on a proof analogous to the proof provided in [FLM+10] , we can show that our problem NP-complete.

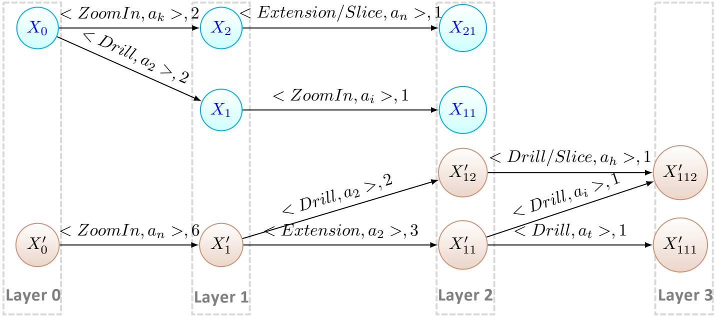


Figure 5.6.: Two exploration session graphs [BH19b]

Accordingly, we propose in what follows two approximate merge solutions. The two proposed strategies prune the matching space (where we compute candidate matches to begin with) such that we can guarantee that the merged graph remains a DAG. Depending on how constraining the pruning strategy is, the matching quality may vary, as experiments validate.

5.4.2.3. The root-layer-merge algorithm

The first algorithm is inspired by a match-merging algorithm that summarizes (through the merging of nodes) unordered trees [SZG+18]. The problem is transformed into a variant of the stable marriage problem [GS62] where sub-trees are compared and merged recursively. The match-merging strategy relies on the hierarchy of trees to identify sub-trees prone to merge, recursively descending through the tree. Given that exploration session graphs are DAGs, we can transform them into layered graphs [HN01], as illustrated in Figure 5.6.

Similarly to [SZG+18], we only match and merge nodes at layer i in N with nodes at layer i in N_{MU}.

The details of the main processing of our first algorithm, which we call *root-layer-merge (RLM)*, are provided in Algorithm 2. Given an exploration session graph, a multi-user graph, and a similarity threshold, it first determines all nodes of layer 0 (root nodes) to be matched between the two graphs (lines 1–

```
Algorithm 2: RLM($G_{XS}$,$G_{MU}$, $\theta_{sim}$)
  Input: $G_{XS}(N,E)$: a user graph, $\theta_{sim}$ : a similarity threshold,
         $G_{MU}(N_{MU},E_{MU})$: the multi-user graph
  Output: $G'_{MU}(N',E')$: merged multi-user graph
1 $R_l$ = set of candidate nodes in $N$ at layer $l$ ;
  /* initially filled with all root nodes at level 0 */
2 $R_{MU,l}$ = set of candidate nodes in $N_{MU}$ at layer $l$ ;
  /* initially filled with all root nodes at level 0 */
3 $N' = N \cup N_{MU}$ ;
4 $E' = \emptyset$;
5 $layerMatchMerge(R_l, R_{MU,l}, G'_{MU}(N',E'), \theta_{sim})$;
6 return $G'_{MU}$;
```

2). It further initializes the set of nodes N' and the set of edges E' of the merged graph result G'_{MU} (lines 3 and 4). It then calls $layerMatchMerge$ that recursively matches nodes and updates G' to merge matching nodes. Once $layerMatchMerge$ returns, the merged graph is returned.

As shown in Algorithm 3, $layerMatchMerge$ takes two sets of vertices N and N_{MU} as input. These are subsets of the matched exploration session graph and multi-user graph, selected to (i) be at the same layer and (ii) have matching ancestors in the merged graph $G'(N',E')$. The final input parameter is the similarity threshold to be used to determine candidate matches, which is done in line 1. Using stable matchings existing solutions, e.g., Gale Shapley algorithm [GS62], we determine a stable matching $\mathcal{M}$ in line 2. In lines 3 – 9, each match is processed as follows: First, the individual matching nodes are replaced in line 4 by a merged node in the node set of the merged graph, i.e., N'. This entails an update of edges pointing to replaced nodes, which are “rerouted” to point at the merged node. Since no merge has yet occurred at lower layers, we simply connect children of the matching nodes to the merged nodes by adding the corresponding edges to E'. Finally, we recursively proceed in lines 10– 11 with matching and merging children of merged nodes.

Overall, the complexity of Algorithm 3 is in O($|N_{layers}| \times width^2_{layer}$) with

Algorithm 3: layerMatchMerge(N,N_{MU},$G'(N',E')$, θ)

```
Input: $N$: nodes of the exploration session graph,
       $N_{MU}$: nodes of the multi-user graph,
       $\theta$: a similarity threshold,
       $G'(N',E')$: the merged graph, initially empty
$candidateMatches = computeSimilarPairs(N, N_{MU}, \theta)$;
$\mathcal{M} = stableMatching(candidateMatches)$;
foreach $m = (n, n_{MU}) \in \mathcal{M}$ do
    $N_{merged} = merge(m)$;
    $N' = (N' \setminus \{n, n_{MU}\}) \cup N_{merged}$;        /* replace individual
     nodes by merged one */
    foreach $(p, n, l) \in E'$ and $(p, n_{MU}, l) \in E'$ do
        $E' = (E' \setminus \{(p, n, l)\}) \cup \{(p, N_{merged}, updateLabel())\}$;
         /* update edges pointing to $n$ and $n_{MU}$  */
    foreach $(n, c, l) \in E$ and $(n_{MU}, c, l) \in E_{MU}$ do
        $E' = E' \cup \{(n, c, l)\}$;  /* add edges to children both $n$
         and $n_{MU}$ to $E'$ */
/* Recursive matching of children                          */
if $children(n_i) \neq \emptyset \wedge children(n_j) \neq \emptyset$ then
    $layerMatchMerge(children(n_i), children(n_j), G'_{MU}(N', E'), \theta))$;
```

$|N_{layers}| = \max(|G1.layers|, |G2.layers|)$ is the maximum number of layers of processed graphs and $width_{layer} = \max_{i=1}^{|G1.layers \cup G2.layers|} |\text{width–layer}(i)|$ is the layer of processed graph having the maximum number of nodes. We postpone the evaluation of the performance of Algorithm 3 to Section 6.5.

It is worth stressing also that our Algorithm 3 does not introduce cycles and preserves the DAG property. We prove this in what follows by contradiction:

Proof 2 ***Assumption***: *The application of RLM on two DAG graphs G_1 and G_2 leads to the generation of matching set $\mathcal{M}$. Accordingly, we use $\mathcal{M}$ to merge G_1 and G_2. This results in a merged graph GM with a cycle c.*
$\Longrightarrow$ *there exists an edge e from node y to x in GM such that e is not in G_1 and $x \in G_1$ and $y \in G_1$ and $layer(x) < layer(y)$ with $layer(n)$ is a function that returns the layer number associated to an input node n.*

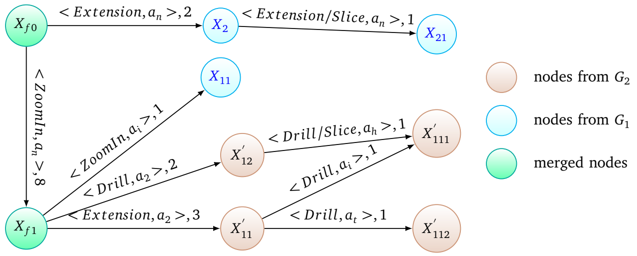


Figure 5.7.: Fused mutli-user exploration graph [BH19b]

$\Longrightarrow$ *there exists an edge e' from node B to A in G_2 with $B \in G_2$ and $A \in G_2$ such that $\mathcal{M}(x) = A$ and $\mathcal{M}(y) = B$. In this case, we have $layer(B) < layer(A)$.*

$\Longrightarrow$ *we have matching $M(x) = A$ and $M(y) = B$. Following our algorithm RLM, this means that $layer(B) = layer(y)$ and $layer(A) = layer(x)$. Recall that $layer(x) < layer(y)$ this means that $layer(A) < layer(B)$. This is in contradiction with the conclusion made in the previous line.*

Figure 5.7 depicts the merged multi-user graph resulting from the application of Algorithm 2 on two layered exploration graphs shown in Figure 5.6. Using this algorithm, assume that both root nodes are matched. This entails a merge of both nodes into the node X_{f0} (as depicted in Figure 5.7). Then, the algorithm proceeds to layer 1, where the exploration steps X_1 and X'_1 having consecutively queries Q_1 and Q'_1 are matched. This entails a merge of both these nodes at layer 1. As a consequence, at layer 2, further possible matches among pairs in $\{X_{11}\} \times \{X'_{11}, X'_{12}\}$ are searched. Assuming no further matches are found, the algorithm stops at this layer, resulting in the merged multi-user graph depicted in Figure 5.7.

5.4.2.4. The match-layer-merge algorithm

While we have shown in the previous section that the RLM algorithm does not introduce cycles in the merged graph, it is quite restrictive in terms of possible matches as it assumes that all exploration sessions begin with a similar query at layer 0 and once matching paths diverge, they cannot

Algorithm 4: MLM(G_{XS},G_{MU}, θ_{sim})

```
Input: $G_{XS}$: the user exploration session graph,
       $G_{MU}$: the multi-user graph,   $\theta_{sim}$: a threshold
Output: $mergeCand$: pairs of nodes (and their similarities) to be
        fused
$n = G_{XS}.nextNode()$;
Mark node $n$ as visited;
$N_{MU,candidates} = getCandidateNodes(G_{MU})$;
$mergeCand$ = set of $(n_i, n_j, s)$ triples of matching node pairs with
  similarity $s$, initially empty;
$candidateMatches = computeSimilarPairs(\{n\}, N_{MU,candidates}, \theta_{sim})$;
if $candidateMatches \neq \emptyset$ then
    $bestMatch$ = match in $candidateMatches$ with the highest
      similarity score ;
    /* unify checks if 1:1 matching is violated among
       all merge candidates                                   */
    $mergeCand = mergeCand.unify(bestMatch, G_{XS})$;
    if $bestMatch \neq NULL$ then
        $mergeCand = mergeCand \cup bestMatch$;
        Mark all nodes in $G_{XSm}$ and $G_{MUm}$ as visited and matched;
if $G_{XS}.hasNext()$ then
    $mergeCand = mergeCand \cup MLM(G_{XS}, G_{MU}, \theta_{sim})$;
foreach $merge \in mergeCand$ do
    $layerMatchMerge(children(merge.m1), children(merge.m2), G_{MU}, \theta_{sim})$;
```

converge at later layers. This assumption does not necessarily mimic reality where exploration sessions can start with completely different tasks and overlap later. Accordingly, this assumption prevents us from finding a solution that generates a compact merged graph which is one of the main goals of our studied problem. Thus, we propose the *match-layer-merge(MLM)* algorithm, shown in Algorithm 4, which is a meet-in-the-middle approach between a full stable marriage approach (that introduces cycles) and the layered variant (that produces few matches).

The MLM algorithm takes the same input as the RLM algorithm, however, it relies on an iterator $nextNode()$ that traverses a graph in breadth-first-order of nodes that have not been marked as visited (which happens in lines 2 and 11 of Algorithm 4). While we get in line 1 the next node n of the exploration session graph, line 3 retrieves in the multi-user graph, the candidates of merge with node n, using the $getCandidateNodes$ method. The set $mergeCand$ includes triples (n_i, n_j, s) where $n_i \in G_{XS}$, $n_j \in G_{MU}$, and $s = sim(n_i, n_j)$. At every stage of processing, it includes a valid 1:1 association of nodes to be merged among the two graphs. In line 7 , we identify candidate matches between node n and the selected subset of nodes in G_{MU}. The best match, if it exists, is unified in line 8 with the existing matching $mergeCand$. The unification checks if $mergeCand \cup \{bestMatch\}$ is still a 1:1 matching. If it is not, it resolves the issue. We will discuss two variants of this method later. Given a non-NULL (valid) $bestMatch = (n, n_j, s)$, we mark in line 11 all nodes in the subgraphs of G_{XS} and G_{MU} starting at n and n_j respectively as matched (and obviously as visited) nodes. This prevents them from being selected as next node. Assuming no further non-matched or non-visited nodes exist in G_{XS}, MLM returns $mergeCand$ that corresponds to $\mathcal{M}$. Thereafter, we call in line 15 Algorithm 3 to possibly find more matches in sub-graphs starting at each pair present in $\mathcal{M}$.

Overall, the complexity of our MLM algorithm is in $O(|G_1| \times |G_2|)$ with $|G_1|$ and $|G_2|$ are the size of the two merged graphs G_1 and G_2. The number of recursive calls to RLM algorithm is in $O(\mathfrak{J})$ with $\mathfrak{J}$ is the depth of the smallest graph between G_1 or G_2. In other words, $\mathfrak{J}$ is the number of layers in the smallest graph under merge. We postpone the evaluation of the performance of MLM algorithm to Section 6.5.

Figure 5.8 depicts the multi-user graph obtained when applying Algorithm 4 on the two exploration graphs shown in Figure 5.6. Indeed, Algorithm 4 discovers initially the best matching candidate that is (X_0, X'_1, s). Accordingly, it merges nodes present in best matching (X_0, X'_1, s) and calls RLM algorithm to go over their sub-graphs. This leads to two subsequent merges as depicted in Figure 5.8. Overall, Algorithm 4 generates a more compact fused-exploration graph whose size is less than the fused graph

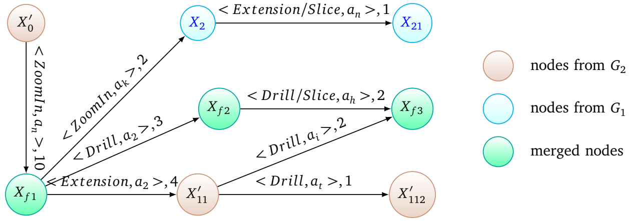


Figure 5.8.: Fused mutli-user graph using MLM [BH20]

obtained in Figure 5.7.

As mentioned previously, we consider different variants of $unify$ (line 8). These actually go hand in hand and deal with intended matching that may violate the 1:1 matching property. This property may be violated when a best match $m = (n, n_j, s)$ (obtained in line 7) includes a node $n_j \in G_{MU}$ that is already part of the matching, i.e., $\exists(n_i, n_j, s') \in mergeCand$. We say that such a match $m' = (n_i, n_j, s')$ is conflicting with m. To ensure a final 1:1 matching $\mathcal{M}$, we consider two strategies to incrementally build the matching.

The first one avoids the appearance of conflicting matches, while the second strategy resolves these. The conflict avoiding strategy CA simply retains the first matching found to a node n_j of the multi-user-graph, i.e., it will always retain m' and reject m. This can be implemented efficiently by ensuring that any previously matched node is no longer a candidate to which matches are searched for. That is $getCandidateNodes$ always returns all nodes of G_{MU} that have not been marked as matched. As a consequence, $unify$ can simply perform the union $mergeCand \cup \{bestMatch\}$, because $bestMatch$ is guaranteed to match a non-matched node of G_{XS} with a non-matched node of G_{MU}.

The second strategy, denoted as CR, allows conflicts to arise by always returning all nodes of G_{XS} when calling $getCandidateNodes$. Then, two cases are possible. In the first case, the similarity of the more recent match m is lower than the score of m', i.e., $s \leq s'$. In this case, m is ignored, which updates $bestMatch$ to $NULL$. Otherwise, m' is revoked, and $bestMatch$ remains m. To revoke m', we remove both m' and all matches to its descendant nodes from $mergeCand$. The nodes involved in the removed matches are unmarked, i.e., switch back from matched to unmatched nodes.

We evaluate the two strategies in Chapter 6. Overall, evaluation results show that the second strategy CR is the best suited when merging evolution provenance graphs.

5.4.3. Collaborative-filtering query recommendation

Using, the multi-user graph obtained as described in the previous section, we describe now our proposed collaborative-filtering query recommendation approach.

To do that, we first present a baseline collaborative-filtering query recommendation approach that, given a current exploration step $X = \{Q, V\}$, searches top-k similar exploration steps in G_{MU} and considers children of those as interesting next exploration steps to be recommended. We further discuss optimizations that aim at improving the runtime of computing recommendations.

5.4.3.1. Baseline recommender

Algorithm 5 summarizes our baseline recommendation algorithm. It takes as input a multi-user graph G_{MU}, a distance threshold θ_{dist}, and the current exploration step $X = \{Q, V\}$. We use a priority queue $Queue_{cand}$ to store the top-k similar exploration steps in descending order of their distance to X_{curr}. This latter is populated in line 5 by $findKSimilar$ function (discussed further below). Then, in lines 3– 5 of Algorithm 5, we determine the children of each top-k similar node in G_{MU} and add them to the set of recommended queries S_{collab}. Thereby, we get ultimately the full set of collaborative-filtering recommendations.

Example 14 (Collaborative-filtering query recommendation example) *Figure 5.9 illustrates an example of collaborative-filtering query recommendation. Given a user's current exploration step and a multi-user graph, our*

```
Algorithm 5: Baseline recommendation algorithm
  Input: $G_{MU}$ : multi-user graph,
  $\theta_{dist}$: distance threshold,
  $X = \{Q, V\}$: current exploration step,
  $S_{content}$: set of recommendation seed scores from our previous
  approach
$Queue_{cand} \leftarrow findKSimilar(G_{MU}, \theta_{dist}, Q, Queue_{cand})$;
$S_{collab} \leftarrow \emptyset$;
foreach $\langle n, d \rangle \in Queue_{cand}$ do
    foreach $e \in n.getOutEdges()$ do
        $S_{collab} \leftarrow rec \cup \{e\}$ ;
$S_{hybrid} \leftarrow S_{content}$ ;            /* Initialization of $S_{hybrid}$ */
foreach $elt \in S_{hybrid} \cap S_{collab}$ do
    $S_{hybrid}[elt] \leftarrow combineScores(S_{hybrid}[elt], S_{collab}[elt])$;
```

Algorithm 5 searches first the set of exploration steps available in the multi-user graph that are highly similar to the current exploration step investigated by the user. Assume that our Algorithm 5 searches top-1 similar exploration steps. In this case, it found X_{f1} (depicted in green color) that is the most similar exploration step (available in the multi-user graph) to the current exploration step investigated by the user. Consequently, we search direct descendants of X_{f1}. This results in the set of exploration steps REC_1, REC_2, and REC_3 (highlighted in red color). Our Algorithm 5 takes also into account scores present in the edges pointing to REC_1, REC_2, and REC_3. Ultimately, pairs of exploration steps (REC_1, REC_2, and REC_3) and their scores are returned as the final result of the collaborative-filtering recommendation process.

Actually, the recommendation engine (see Figure 3.1, Page 36) is executed as a whole block when a user solicits recommendations. Hence, the recommendation engine implemented in EVLIN computes first the set of content-based recommendations $S_{content}$, assigns interestingness scores to each recommendation present in $S_{content}$ using our recommendation quantification approach (discussed in Section 5.3) and finally invokes our

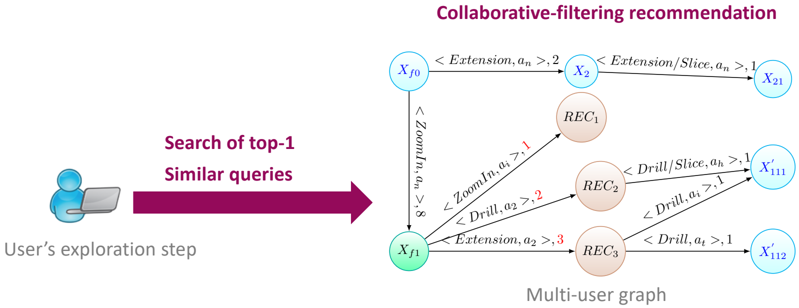


Figure 5.9.: Example of a collaborative-filtering recommendation computation

collaborative-filtering recommendation approach described in Algorithm 5. Accordingly, we use the set of collaborative-filtering recommendations computed in lines 6–8 of Algorithm 5, to update scores of content-based recommendations $S_{content}$ already computed as described in Section 5.3.

Recall that each collaborative-filtering recommendation is a direct descendant of an exploration step in the multi-user graph that has a high similarity to the current exploration step performed by the current user. Accordingly, we leverage information of edges that describe navigations from the highly similar exploration steps stored in $Queue_{cand}$ to the collaborative-filtering recommendations. More specifically, each edge is labeled by $\langle const, s \rangle$ as explained in Definition 4.5 where scores s reflect (i) the commonality (frequency) of the underlying navigation and; (ii) its exploration benefit (the number of subsequent explorations steps investigated by performing this navigation). That is, we combine in lines 6–8 of Algorithm 5, scores derived from labels of edges pointing to S_{collab}, with the (previously computed) scores $S_{content}$ from content-based recommendations. In our implementation, we simply implement $combineScores$ as the average of both normalized query-recommendation scores. Investigating further combination functions is left for future work.

In our baseline algorithm, $findKSimilar$ (see Algorithm 6), simply tra-

```
Algorithm 6: $findKSimilar(G_{MU}, \theta_{dist}, Q, Queue_{candidates})$
Input: $G_{MU}$ : multi-user graph,   $\theta_{dist}$: distance threshold,
       $Q$: query of current exploration step,   $Queue_{candidates}$: top-k similar queries
$d_{max} \leftarrow \infty$ ;
foreach $n \in G_{MU}$ holding $X_{MU} = \{Q_{MU}, D, V_{MU}\}$ do
    $d \leftarrow distance(Q, Q_{MU})$;
    if $d < \theta_{dist}$ then
        if $|Queue_{candidates}| \leq k$ then
            $Queue_{candidates}.add(\langle n, d \rangle)$;
        else
            $d_{max} \leftarrow Queue_{candidates}.last().getDistance()$;
            if $d \leq d_{max}$ then
                $Queue_{candidates}.removeLast()$;
                $Queue_{candidates}.add(\langle n, d \rangle)$;
```

verses the whole multi-user graph, computing a distance between Q and each query of G_{MU} and maintaining $Queue_{cand}$ such that it ultimately contains the k most similar (equivalent to k least distant) queries to Q. The complexity of $findKSimilar$ function is in $O(N_{MU})$. Thus, a big multi-user graph incurs a costly recommendation search operation. This is a critical issue for interactive visual exploration. Indeed, we show in the evaluation (see Section 6.6) that our baseline collaborative-filtering recommendation approach is an expensive process in terms of runtime. Therefore, we propose in what follows two pruning techniques that reduce the number of distance computations.

5.4.3.2. Triangle inequality based pruning

The triangle inequality has been previously used in many works, e.g., [RHS14; ZQYC12] to optimize the search of sub-graphs. As our adopted distance is computed as $distance(Q, Q') = 1 - sim(Q, Q')$ (see Equation 5.3), we can also exploit the triangle inequality that holds for the Jaccard coefficient, i.e., $Jaccard(A,B) \leq Jaccard(A,C) + Jaccard(C,B)$.

Lemma 1 (Lower bound distance) *Given queries Q_1 and Q_2 to compare with a query Q, it is true that $distance(Q_1,Q) \geq distance(Q_2,Q)$ if $distance(Q_1,Q_2) \rightsquigarrow 0$.*

Proof 3 *Based on the aforementioned triangle inequality for the Jaccard coefficient and $distance(Q,Q') = 1 - Jaccard(Q,Q')$, we get for queries Q_1, Q_2, and Q:*

$$\begin{aligned}
Jaccard(Q,Q_1) &\leq Jaccard(Q,Q_2) + Jaccard(Q_2,Q_1) \\
\Rightarrow Jaccard(Q,Q_1) &\leq Jaccard(Q,Q_2) \text{ if } Jaccard(Q_2,Q_1) \rightsquigarrow 1 \\
\Rightarrow distance(Q,Q_1) &\geq distance(Q_2,Q) \text{ if } distance(Q_2,Q_1) \rightsquigarrow 0
\end{aligned}$$

Based on this lemma, we reduce the number of distance computations in some cases. More specifically, assume that we have compared the current user exploration step $X = \{Q,V\}$ to an exploration step $X_{MU} = \{Q_{MU},V_{MU}\} \in G_{MU}$ and we found that $distance(Q,Q_{MU}) \geq \theta_{dist}$. Supposing that there exists another exploration step $X'_{MU} = \{Q'_{MU},V'_{MU}\} \in G_{MU}$ such that $distance(Q_{MU},Q'_{MU}) \rightsquigarrow 0$, we can conclude that $distance(Q,Q'_{MU}) \geq \theta_{dist}$.

To leverage this pruning technique, asynchronously to the actual recommendation computation (i.e., in an offline fashion), we cluster highly similar nodes of G_{MU} such that the distance of queries of any two nodes in a cluster tends to 0, as determined by a parameter ϵ close to 0. During online processing, whenever we find that the distance between the current exploration step query Q and a query Q_{MU} exceeds the acceptable distance threshold θ_{dist}, we lookup and remove all queries in the same cluster as Q_{MU} from the set of queries to which Q needs to be compared to.

Given that the multi-user graph results from merging similar nodes during evolution provenance aggregation, we expect that pruning solely based on the triangle inequality reduces moderately the number of nodes to iterate over during the recommendation phase.

5.4.3.3. Parenthood relationship based pruning

According to our definition of an exploration session graph G_{XS} (see Definition 4.4, Page 52), an edge represents the transition from an exploration step with query Q to another exploration step with query Q'. By construction, Q' is derived from Q as described in Figure 5.2 (Page 68). Given these derivation rules, it is easy to verify that $Q.getSelect() \cap Q'.getSelect() \neq \emptyset$, $Q.getConditions() \subseteq Q'.getConditions()$, and $Q.getTables() \subseteq Q'.getTables()$. We only elaborate our next pruning strategy for the select clause in what follows. The other clauses can be treated analogously. Consequently, we suppose that $distance(Q,Q') = 1 - Jaccard(getSelect(Q), getSelect(Q'))$.

Lemma 2 (Parent as a lower bound) *Let $X_{parent} = \{Q_{parent}, D, V_{parent}\}$ and $X_{child} = \{Q_{child}, D, V_{child}\}$ be two exploration steps in G_{MU} connected by an edge $e = (X_{parent}, X_{child}, L)$. The distance between Q_{parent} and the query Q of the current exploration step can be used to estimate the lower bound of the distance between Q_{child} and Q as*

$$distance(Q_{child}, Q)) \geq distance(Q_{parent}, Q) - \beta \tag{5.5}$$

$$with\ \beta = \frac{\big(getSelect(Q_{child}) \setminus getSelect(Q_{parent})\big) \cap getSelect(Q)}{getSelect(Q_{parent}) \cup getSelect(Q)}$$

Proof 4 *This proof applies for edge e on exploration steps X_{parent} to X_{child} having "Zoom IN" or "Extension" as label for the operation type. The proofs for the remaining operations are available in Appendix B.*

Let $getSelect(Q) = \{a, f(m)\}$, $getSelect(Q_{parent}) = \{a_p, f_p(m_p)\}$ and $Q_{child} = \{a_p, a_c, f_p(m_p)\}$. Note that the select clauses of Q_{parent} and Q_{child} are guaranteed to overlap (i.e., in a_p and $f_p(m_p)$) based on our derivation rules (see Figure 5.2).

$$Jaccard(Q_{child}, Q) = \frac{\{a, f(m)\} \cap \{a_p, a_c, f_p(m_p)\}}{\{a, f(m), a_p, a_c, f_p(m_p)\}}$$

By removing a_c from the denominator,

$$Jaccard(Q_{child}, Q) \leq \frac{\{a, f(m)\} \cap \{a_p, a_c, f_p(m_p)\}}{\{a, f(m), a_p, f_p(m_p)\}}$$

This is equivalent to: $Jaccard(Q_{child}, Q) \leq \frac{\{a, f(m)\} \cap \left(\{a_p, f_p(m_p)\} \cup \{a_c\}\right)}{\{a, f(m), a_p, f_p(m_p)\}}$

We can simplify it further as follows

$$Jaccard(Q_{child}, Q) \leq \frac{\left(\{a, f(m)\} \cap \{a_p, f_p(m_p)\}\right) \cup \left(\{a, f(m)\} \cap \{a_c\}\right)}{\{a, f(m), a_p, f_p(m_p)\}}$$

$$\Rightarrow Jaccard(Q_{child}, Q) \leq Jaccard(Q_{parent}, Q) + \frac{\{a, f(m)\} \cap \{a_c\}}{\{a, f(m), a_p, f_p(m_p)\}}$$

$$\Rightarrow Jaccard(Q_{child}, Q) \leq Jaccard(Q_{parent}, Q) + \frac{\left(getSelect(Q_{child}) \setminus getSelect(Q_{parent})\right) \cap getSelect(Q)}{getSelect(Q_{parent}) \cup getSelect(Q)}$$

$$\Rightarrow distance(Q_{child}, Q)) \geq distance(Q_{parent}, Q) - \frac{\left(getSelect(Q_{child}) \setminus getSelect(Q_{parent})\right) \cap getSelect(Q)}{getSelect(Q_{parent}) \cup getSelect(Q)}$$

Note that when $getSelect(Q) = \{a, f(m)\}$, $getSelect(Q_{parent}) = \{a_p, f_p(m_p)\}$ and $Q_{child} = \{a_p, a_c, f_p(m_p)\}$, $\beta = \frac{\{a, f(m)\} \cap \{a_c\}}{\{a, f(m), a_p, f_p(m_p)\}}$. Thus, the lower bound of distance between Q_{child} and Q is equal to $distance(Q_{parent}, Q)$ when $\{a, f(m)\} \cap \{a_c\} = \emptyset$. In this case, if $distance(Q_{parent}, Q)) \geq \theta_{dist}$, we can directly conclude that $LowerBound(distance(Q_{child}, Q)) \geq \theta_{dist}$. This means that we can directly prune Q_{child} from the set of queries to search without any distance computation. For the second case, (where $\{a, f(m)\} \cap \{a_c\} \neq \emptyset$), $arg\,max(\beta)$ is $\frac{1}{3}$ when $a_c = a$ and $f(m) = f_p(m_p)$. Thus, if the difference between $distance(Q_{parent}, Q)$ and θ_{dist} called Δ is bigger than $\frac{1}{3}$, we can prune Q_{child}.

Implementing the above pruning technique leverages a data structure that asynchronously (offline mode) computes parent-child information and the exact extensions done to derive a Q_{child} from a Q_{parent}. At each iteration, we then check if the lower bound distance can be estimated based on a previous distance computation using Equation 5.5, rather than being directly computed. If so, the lower bound is used to verify if the distance can exceed θ_{dist} without computing the more expensive similarity score.

5.4.4. Related work

In the following, we review the most relevant research related to our contributions discussed in this section.

Graph merging. Our merge approach fits in the scope of inexact matching [YYL+16] where there are structural differences between processed graphs. Traditionally, inexact matching is presented as an assignment problem where a cost function is introduced to compute the optimal assignment. In this case, the problem consists of finding an assignment that minimizes such cost. This leads to two formulations of the graph matching problem. The first formulation called Linear Sum Assignment Problem, considers only information about the vertices. Several Hungarian-type algorithms [JV87; Kuh55] exist to solve these kind of problems. However, applying these solutions to our directed exploration session graphs may lead to cycles as shown in Section 5.4.2.2.

The second formulation, called quadratic assignment problem (QAP) [KB55] leverages vertices and edges information. Indeed, it aims at minimizing the number of adjacency disagreements between the two matched graphs. Yet, it was proven in [BDM09] that QAP is NP-Hard. Accordingly, several relaxation algorithms, e.g., [LHS09; LQ14] have been proposed to find an approximate solution. Essentially, these approximate solutions transform the graph matching problem into a sequence of convex problems, such that a given initial solution is improved iteratively by decreasing the cost function up to a fixed point. In our work, we adopt a similar approach where we find initially most interesting matchings to perform. Those seed matchings are used to trigger the discovery of further matchings to perform. Nonetheless, our work differs from existing approximate solutions such as [LHS09; LQ14] in the initialization of the seed matching solution that will be subsequently improved. Indeed, our work is more oriented to graphs integration rather than matching. Thus, we aim at maximizing the integration of a small graph representative of a user exploration on a bigger graph corresponding to the multi-users graph. This “one-way matching” relieves the complexity of the problem of finding a seed matching solution to expand later. Hence, we match initially the user’s exploration graph nodes to the global graph in order to identify seed pair vertices whose children are later matched recursively.

Graphs Summary. Our merge approach that maintains periodically the multi-user graph also relates to the graph summarization. The main goal of graph summarization is to generate a concise graph summary beneficial for several purposes, e.g., efficient storage and query processing. In this context, we mention [LSDK18] as a recent survey on the topic of graph summary. This survey categorizes state-of-the-art graph summary approaches by the type of graphs taken as input as well as by the core methodology employed.

While these work focus mainly on compressing all similar nodes inside the same graph or between homogeneous graphs, we focus on performing a 1:1 merge of two similar nodes belonging to different graphs. Indeed, our goal is to increase the compactness of the multi-user graph while preserving the DAG aspect of the merged graph rather than compressing aggressively the multi-user graph. Yet, graph summary techniques could be an interesting research avenue to further minimize our multi-user graph.

Graph search. Our collaborative-filtering recommendation approach supported in EVLIN analyzes previous users' explorations to match the current exploration. A straightforward solution would be to use top-k similar graphs techniques to find previous exploration sessions having excerpts highly similar to the current exploration. In this context, we find several works proposed to find efficiently the top-k similar graphs given an input graph. This includes for instance [ZQYC12], [RHS14], and [GGY+14]. Yet, these techniques require the traversal of all individual previous exploration session graphs that may encompass redundancy. This is costly and does not fit our interactive visual data exploration context. Furthermore, this strategy may mitigate the efficiency of collaborative-filtering recommendation since a highly similar exploration session graph does not contain necessarily a node, highly similar to the current user's exploration. Accordingly, we adopt a top-k similar items search strategy to provide recommendations. This strategy is applied to our compact multi-user graph that is periodically maintained. Furthermore, we showed in Section 5.4.3.3 that the high connectivity incurred by the incremental merge of the multi-user graph is beneficial to optimize the top-k similar items search process.

Collaborative-filtering query recommendation. Several existing data exploration tools support users in investigating data by providing facilities to express exploration queries. In general, these work employ either content-based or collaborative-filtering recommendation techniques. As we have covered in Section 5.2.4 the set of existing work employing content-based recommendations to improve data exploration, we dedicate the rest of this paragraph to discuss existing work employing collaborative-filtering recommendation policy.

In the collaborative-filtering recommendation approach, existing data exploration work (e.g., [EAPS14; KKBS10]) rely on a set of previous explorations made by multiple users to identify prior exploration queries worth recommending to the current user. Yet, this range of systems provides limited visual interaction facilities when assisting SQL query formulation. Therefore, these works rely on a basic form of history traditionally presented as a serial trace, whereas we focus on interactive visual exploration whose history contains various users' navigations and forms thereby a direct graph. Accordingly, we benefit from our rich history model to infer ratings of exploration steps (via edges' scores), used in our collaborative-filtering recommendation approach. Recently, Milo et al proposed REACT [MS16; MS18], a visual data exploration tool that provides collaborative recommendations based on previous queries made by users exploring various data sets. While REACT leverages the history of users' explorations over different datasets to mitigate the problem of cold-start, e.g., absence of history about the explored data set, our system EVLIN remedies the cold-start situation by providing alongside the collaborative recommendations, other recommendations.

5.5. Visualization recommendation

We have introduced in Sections 5.2, and 5.4 our query recommendation approaches meant to assist users in exploring interesting data regions. Yet, users, especially those lacking visualization skills, still face difficulties in

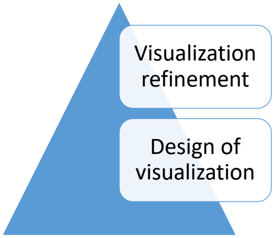


Figure 5.10.: Overview about our visualization recommendation approach

the course of the visual data exploration process especially when visualizing queries' results. Indeed, these users need support to design adequate visualizations that render appropriately the data. Furthermore, the data visualization could be a non-trivial task even for users having sufficient visualization knowledge since choosing the best visual/data mapping can be challenging given the large number of visualization techniques candidates that could be applied to render data.

In what follows, we tackle the problem of recommending 2-d visualizations in the context of visual exploration of data warehouses. Accordingly, we give first in Section 5.5.1 an overview about our proposed visualization recommendation approach. After that, we introduce in Section 5.5.2 metrics that we take into account when recommending visualizations. Later, we describe in Section 5.5.3 in details our visualization recommendation approach, implemented in EVLIN.

5.5.1. A high-level overview of the visualization recommendation process

We propose in this section, our novel visualization recommendation approach meant to render appropriately the exploration steps inspected by users.

Figure 5.10 depicts the general process of our visualization recommendation approach, which comprises two main steps: (i) design of visualization, and (ii) visualization refinement.

In the first step of the process depicted in Figure 5.10, our visualization recommendation approach resorts to widely used existing techniques to choose appropriately the visualization technique suitable to render exploration results. The selected visualization technique is instantiated at this stage of the visualization recommendation process. Next, our visualization recommendation approach adopts in the second step of the visualization process (cf. Figure 5.10) a set of metrics (explained later) to specify the visual encodings of the current instance of visualization to recommend. Finally, our approach outputs the recommended visualization.

In the next section, we discuss the set of metrics adopted in second step of our proposed visualization recommendation process. Later, we describe thoroughly in Section 5.5.3 how our proposed visualization recommendation approach performs the two steps of the process depicted in Figure 5.10 to assist users when inspecting exploration results.

5.5.2. Metrics of visualization recommendation

Towards recommending 2-d visualizations in the course of visual exploration session, we leverage the following metrics:

Expressiveness and effectiveness. The effectiveness and expressiveness metric were discussed in [CM84; Mac86] where authors propose several rules to prune and rank visualization candidates. As discussed in [Won18], the expressiveness metric verifies whether a visualization expresses all the facts available in the visualized dataset. The effectiveness metric determines whether a visualization effectively conveys the information in a way more readily perceived than other visualizations. These metrics are commonly adopted in visualization recommendation systems including Polaris [STH02], Tableau Show me [MHS07], and Voyager [WMA+16].

While we also adopt the effectiveness and expressiveness metrics in our work, we argue that these are not sufficient over the course of an exploration session that navigates between exploration steps. Generally, these visualizations generated within the same exploration session are related to one another as they describe related topics. Thus, these visualizations may lead to confusion if they are generated independently, e.g., using different visual encodings to render the same information. This disturbs users and

Data types	Mark types
(Ordinal or nominal) × (Ordinal or nominal)	point>text
Quantitative × nominal	bar>point>text
Quantitative × (temporal or nominal)	line>bar>point>text
Quantitative × Quantitative	point>text

Table 5.3.: Ranked mark types based on the data types of 2-d visualizations [WMA+16]

can mislead conclusions surfaced from visualizations. To cope with this inconsistency problem, we propose our proper metric for visualization recommendation.

Consistency. Apart from the expressiveness and effectiveness metrics, we adopt the consistency as an additional metric when recommending visualization in the course of the visual data exploration process. The consistency metric consists of presenting the same information (or concept) in the same way. In the same spirit, different information should be presented differently using the consistency metric.

In the following section, we describe how our visualization recommendation approach implements these three metrics. A special attention will be given to the consistency, our proper metric that we adopt when recommending visualization in EVLIN.

5.5.3. Implementation of visualization recommendation

Our visualization recommendation approach leverages the following metrics: (i) expressiveness/effectiveness and (ii) consistency to render 2-d visualizations.

While the expressiveness/effectiveness metrics are implemented as a rule-based approach (as shown in Table 5.3), we leverage the evolution provenance graph G_{XS} discussed in Chapter 4, to implement the consistency metric.

Our visualization recommendation approach follows the pipeline depicted in Figure 5.11.

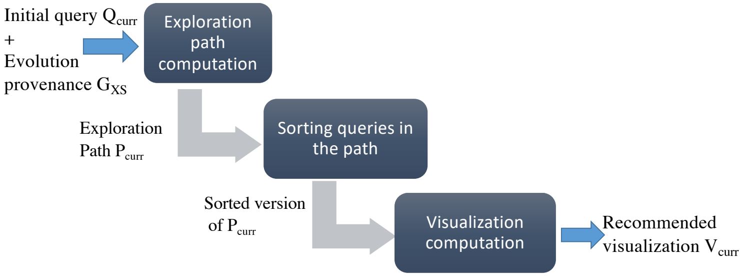


Figure 5.11.: Steps of the visualization recommendation process

It comprises three steps that are: (i) *Exploration path computation*, (ii) *Sorting queries in the path*, and (iii) *Visualization computation*. It takes as input a current exploration query investigated by the user Q_{curr} and evolution provenance graph G_{XS} that records all user's manipulations done within the exploration session. The visualization recommendation approach returns a recommended visualization V_{curr} that renders results of Q_{curr}.

The first step of the visualization recommendation process that is *Exploration path computation* consists in computing the exploration path (see Definition 4.3) that leads to Q_{curr}. This path contains all exploration queries involved to reach the query Q_{curr}. This path undergoes a second step *Sorting queries in the path* where exploration queries are sorted in descending order of their similarity with the current exploration query Q_{curr}. Finally, we compute the recommended visualization in the third step *Visualization computation* where we fetch iteratively the visualization exploration resource (see Definition 4.2) associated with previous exploration queries present in the sorted exploration path. Those visualization resources are used to construct the recommended visualization V_{curr} associated with Q_{curr}.

After this brief overview of main steps depicted in Figure 5.11, we discuss Algorithm 7 that shows the pseudocode of the visualization recommendation approach.

Essentially, it takes as input the current exploration query Q_{curr} to inspect and the evolution provenance graph G_{XS} recording the set of exploration

Algorithm 7: Visualization recommendation algorithm

```
   Input: $Q_{curr}$, $G_{XS}$
   Output: $V_{curr}$: Recommended visualization associated to $Q_{curr}$
1  $AttsType \leftarrow$ ExtractTypes($S_{curr}$);
2  $VisType \leftarrow$ DecideVisTechniques($AttsType$);
3  $V_{curr} \leftarrow$ AllocateMarks($VisType$);
4  $isComplete \leftarrow$ false ;
5  $P_{curr} \leftarrow$ ComputeExplorationPath($Q_{curr}, G_{XS}$);
6  $Queue \leftarrow$ SortExplorationSteps($P_{curr}, Q_{curr}$) ;
   /* we use a queue of $\langle q,s \rangle$ elements where $q$ is a query
      and $s$ is a similarity value                                     */
7  while $\neg isComplete$ AND $\neg Queue.isEmpty$ do
8      $X_i = (Q_i, V_i) \leftarrow Queue.poll()$;
9      $ReuseVis(V_{curr}, V_i)$;
10     $isComplete \leftarrow completeCheck(V_{curr})$;
11 if $\neg isComplete$ then
12     $AutoComplete(Vis_{curr})$;
13 Return $V_{curr}$;
```

steps performed so far by the user. This algorithm returns a recommended visualization V_{curr} associated to the query Q_{curr}.

At line 1, we identify the types of attributes queried in Q_{curr} (e.g., ordinal, nominal or temporal data). Given the identified data types, we resort in line 2 to Table 5.3 encompassing expressiveness/effectiveness rules to determine the suitable visualization technique to render the result of Q_{curr}. Based on this information (visualization technique), we initialize at line 3 the visualization V_{curr}. To do that, we instantiate permitted graphical marks associated to the adopted visualization technique as described in Table 5.3. Consequently, we get the preliminary skeleton for the recommended visualization of V_{curr}. This latter should be populated appropriately with visual encodings resources to recommend a suitable visualization V_{curr}.

To this end, we employ the *consistency* metric that relies mainly on previous visualizations seen during the current exploration session to fill missing visual encodings resources of V_{curr}.

Consequently, we leverage at lines 5–6 the evolution provenance graph G_{XS} to extract the exploration path P_{curr} from the initial query Q to the node representing the query Q_{curr}, including all meta-data associated to graph nodes. Note that we leverage Dijkstra's Shortest Path First algorithm [Dij59] to compute the exploration path P_{curr}. Yet, it is easy to to adopt other solutions solving the problem of exploration path computation.

Using the computed exploration path P_{curr}, our visualization recommendation approach aims at maximizing the visual similarity of the recommended visualization V_{curr} with those seen and interacted with previously for similar queries (intuitively, such that users easily recognize the same information as seen previously and thus understand the meaning of visualizations faster).

This first requires determining similar queries among those in P_{curr}. This is done in line 6 in Algorithm 7 by the function SortExplorationSteps described in Algorithm 8.

Algorithm 8: SortExplorationSteps (Q_{curr}, P_{curr})

```
Input: $Q_{curr}$: the current exploration query, $P_{curr}$: the exploration
       path containing explorations sorted in descending order of
       their recentness
Output: $queue$: A queue of exploration steps sorted in a descending
        order of their similarity scores
$queue \leftarrow \emptyset$ ;
/* The queue storing queries associated with their
   similarity scores                                    */
$S_{recentness} \leftarrow 0$;
forall $X_i = (Q_i, V_i) \in P_{curr}$ do
    $w_{Q_i}(Q_{curr}) \leftarrow |P_{curr}| - S_{recentness}$;
    $S_{similarity} \leftarrow sim(Q_i, Q_{curr})$;
    $score \leftarrow S_{similarity} \times w_{Q_i}(Q_{curr})$;
    $queue.add(Q_i, score)$ ;
    $S_{recentness} \leftarrow S_{recentness} + 1$ ;
$queue.sort(score, descending)$;
/* Sort the queue in descending order of their
   similarity scores                                    */
Return $queue$;
```

This latter takes as input the current exploration query Q_{curr} and the exploration path P_{curr} leading to the current user's exploration.

Essentially, the SortExplorationSteps function quantifies in lines 3 – 8 the similarity $S_{similarity}$ between Q_{curr} and an exploration query $Q_i \in P_{curr}$ using a similarity function sim. More precisely, we determine in line 6 for each retrieved previous exploration query, the semantic overlap of the data returned by these queries, compared to the data returned by Q_{curr}. Intuitively, the higher this semantic overlap, the more similar the queries and hence we want to reuse same or similar visual encodings if possible. We denote the function used to compute the semantic overlap between two queries Q_1 an Q_2 as $sim(Q_1, Q_2)$.

Equation 5.3 presents one possible implementation of a the function sim, based on the Jaccard coefficient $Jaccard(S, S') = \frac{|S \cap S'|}{|S \cup S'|}$.

To also account for the fact that visualizations encountered more recently during an exploration session are more present in user's memory than those seen longer ago, we introduce in line 4 of Algorithm 8 a weight $w_{Q_2}(Q_1)$ for a query Q_1 that stores the number of exploration steps present in the exploration path P_{curr}, between a query Q_1 and a query Q_2. For any query $Q_i \in P_{curr}$ on the exploration path yielding Q_i, we compute a score $sim(Q_{curr}, Q_i) \times w_{Q_{curr}}(Q_i)$. We store in line 9 of Algorithm 8 the set of exploration queries of P_{curr} with a similarity to Q_i in a queue that is sorted in descending order of similarity.

This latter is used in the while loop (lines 7–10) of Algorithm 7 to construct the visualization V_{curr}. More precisely, we iterate the filled queue in descending order of similarity. At each iteration we extract a previous exploration step $X_i = (Q_i, V_i)$ and we fetch its associated visualization exploration resource V_i (stored in relational way as described in Section 4.3.1). The set of visual encodings of V_i that comply with the graphical marks of V_{curr} are re-used in the current visualization V_{curr}. Note that we provide details of our ReuseVis function in the Appendix A. The process is repeated iteratively as long as $Queue$ contains similar previous exploration queries and V_{curr} still lacks some visual resources.

In case no prior visualizations can be used or in the initial exploration step (where evolution provenance is empty), we resort in line 12 of Algorithm 7 to a set of predefined visual encodings that we use to complete the recommended visualization V_{curr}.

Overall, the complexity of Algorithm 7 that performs the visualization recommendation approach, is in $O((N+E) \times log(N) + Path_Q \times log(Path_Q) + c \times Path_Q$) where E and N refer respectively to the edges and nodes of the evolution provenance graph, $Path_Q$ is the size of the exploration path and c is the number of visual encodings to fill for the recommended visualization.

Recall that our visualization recommendation comprises three steps that are shown in Figure 5.11. The complexity of the first step that is exploration path computation is in $O((N+E) \times log(N))$. The second step that consists of sorting the exploration path is performed in $O(Path_Q \times log(Path_Q))$ while the third step recommending visualization is in $O(c \times Path_Q)$. We postpone the discussion of the evaluation results of each step present in the visualization recommendation process to Section 6.8.

Finally, we give an example that shows the functionality of the visualization recommendation.

Example 15 *In our running example (see Example 4, Page 41), we have discussed a sample of a visual exploration session using EVLIN where the user investigated initially the set of delayed flights that are grouped by their state of departure. Figure 3.3 (Page 42) displays the recommended bar-chart visualization for the initial user's query Q. For this initial rendering, the evolution provenance graph $G_{XS} = \emptyset$, thus, the recommendation is solely based on the effectiveness/expressiveness metrics.*

As discussed in Example 6 (Page 42), the user has later selected the highest bar that designates the delayed flights departing from the state California. The selected region is highlighted in a different color. At this point, G_{XS} is updated to include the initial exploration step $X = \{Q, V\}$ including its query Q as well as visual encoding parameters of the displayed chart V.

The user's interaction triggers the recommendation of a set of exploration queries (discussed in Example 7, Page 43). Subsequently, the user has requested to investigate the recommended query discussed in Example 13 (Page 70) that *returns the set of delayed flights departing from several states in USA and clustered by airlines companies (e.g. OO, UA, etc ...).*

Recommending the visualization of Figure 5.3 (Page 70) relies on the updated G_{XS} to generate a similar visualization of same features, e.g., same axis scale for the y-axis, same order of states on the x-axis, or choice of a stacked bar chart to maintain same heights as seen previously (e.g., in the bar-chart of Figure 3.3, Page 42).

Unlike existing work discussed in Section 2.2.2, that provide either visual or query recommendation, we show how our system EVLIN integrates seamlessly both query and visualization recommendation for a streamlined, interactive visual exploration user-experience. This is performed by the virtue of our novel recommendation strategies that leverage provenance to recommend queries and interactive visualizations related to each other. Later, we study in Section 6.9 users' experiences when exploring visually data using EVLIN.

5.5.4. Existing visualization recommendation work

Several prior work (e.g., [MHS07; MVT16; STH02; WMA+16; WQM+17]) have focused on recommending visualizations given a query result to investigate. Many of these work leverage effectiveness and expressiveness metrics. The expressiveness metric enables to design a visualization that expresses all the facts present in the visualized dataset. The effectiveness metric consists in choosing the visualization, provided that information conveyed by this particular visualization is more readily perceived than the information in the other visualization.

Examples of work that leverage expressiveness and effectiveness metrics include Show Me [MHS07], VizRec [MVT16] and Voyager [WMA+16]. However, these work are limited to question answer processes where users specify the input data and get as a result the suitable visualization to inspect results. This does not fit the context of visual data exploration where users are exposed throughout an exploration session to many overlapping visualizations that describe related topics. To this end, we adopt these metrics (expressiveness and effectiveness) as well as a third metric that is the *consistency*.

In a similar approach, VizRec [MVT16] mixed metrics proposed in Mackinlay's APT system [Mac86] with a collaborative-filtering recommendation strategy to recommend visualizations. This approach differs from our approach in two main points: (i) they use tags to describe the content of visualizations while we leverage the evolution provenance to get visualizations content; (ii) authors in [MVT16] leverage previous users' ratings over visualizations to find previous users with similar preferences to the active user while we focus only on prior visualizations of the active user. While the above method may complement our visualization recommendation approach, it is left for future research.

5.6. Conclusion

In this chapter, we presented our scientific contributions proposed to implement the *recommendation engine* module of our visual data exploration framework ProvVDE.

In particular, we discussed our content-based query recommendation meant to assist users in exploring interesting data regions. Furthermore, we described our approach proposed to quantify the interestingness of recommended queries

Second, we investigated our merge methods meant to fuse evolution provenance graphs, our collaborative-filtering query recommendation method and how this new recommendation approach is employed to reinforce the process of quantifying the interestingness of recommendations.

Finally, we discussed our visualization recommendation showing thereby how our framework ProvVDE integrates seamlessly both query and visualization recommendation for an interactive visual exploration user-experience.

Finally, we point out that these techniques are tailored to the exploration of data warehouses, leveraging specificities of this application for specific optimizations necessary for good quality results with interactivity requirement. Therefore, all these techniques are implemented in our system EVLIN, which is the basis for the evaluation discussed next.

Chapter 6

Evaluation

6.1. Introduction

We dedicate this chapter to the evaluation of the algorithms discussed in Chapter 5 that integrates an implementation of our framework (see Chapter 3) and leverages the evolution provenance model presented in Chapter 4.

We start by providing a brief description of the experimental setup in Section 6.2. It includes implementation details of our framework implementation EVLIN. In Section 6.3, we evaluate our proposed content-based query recommendation approach. Then, we present in Section 6.4 the evaluation results of various functions implemented to quantify the interestingness of recommended exploration queries. Thereafter, we evaluate in Section 6.5 and Section 6.6 the performance of our merge and collaborative-filtering recommendation methods, respectively, the usability of our collaborative recommendation when visually exploring data warehouses is evaluated in Section 6.7. Later, we evaluate in Section 6.8 the performance of our visualization recommendation approach. Finally, Section 6.9 presents a comparative evaluation of our different recommendation approaches and other state-of-the art systems based on a user's study.

Note that this chapter is significantly based on the evaluation result, published in [BH17; BH19b; BH20]. Further experiments results are discussed here to provide a more thorough evaluation.

6.2. Experimental setup

We describe in this section how our visual data exploration system EVLIN is implemented. Furthermore, we give a glimpse of the set of data warehouses used through the experiments. Finally, we discuss the methodology adopted when performing experiments.

6.2.1. Implementation

We have implemented our visual data exploration system EVLIN as a web application. Our implementation follows the model-view-controller design pattern. The view layer is implemented using JSP, Bootstrap 3, and mainly Vega 1[1] for interactive visualizations. The control layer is based on the Apache Struts framework and it implements the visual data exploration process using Java 8. The model layer leverages the Hibernate framework that offers the necessary facilities to interact with the database. Finally, EVLIN leverages Perm [GA09] as a backend database. Note that Perm [GA09] is an extended version of PostgreSQL that supports provenance information management.

6.2.2. Datasets

Our evaluation relies on three real-world data sets from different domains. The domains are chosen so that some basic knowledge about database schemas and attributes can be assumed.

Formula One. The first data warehouse[2] describes more than 23,000 formula one races made between 1950 and 2017. It contains three dimensions:

[1] https://vega.github.io/vega/

[2] https://www.kaggle.com/cjgdev/formula-1-race-data-19502017

a dimension describing race locations including more than 70 formula one circuits, a car constructors dimension that encompasses information about teams participating in the formula one and a third dimension that concerns drivers' information. It stores information about more than 800 drivers. The facts recorded for each formula one race include various measures such as elapsed time, final position, stops number, etc ...

Soccer. The second data warehouse is the European soccer league database[1]. It contains detailed information (possession, corner, cross, fouls, etc ...) about more than 25,000 fixtures and 10,000 players in 11 European championships for seasons between 2008 and 2016.

Flights. The third data warehouse describes US domestic flights[2]. It is the biggest data warehouse we consider. Its fact table contains around 1 million flights. An overview of this data warehouse is already given in Example 1 (see Page 28). As a reminder, the US domestic data warehouse contains information about one million flights done by more than 1500 airline companies between 2007 and 2008. It includes further information about 3300 airports and almost 4500 plane types used for the covered flights. The facts recorded for each flight include various numerical attributes such as delays, cancellation, arrival and departure time, etc ...

The main features of each data warehouse are summarized in Table 6.1.

6.2.3. Methodology

All experiments described in this chapter were conducted on a single machine with a 2.2 GHz quad-core Intel processor and 16 GB RAM. These experiments

Data warehouse	**#attributes**	**size of the fact table (#tuples)**	**number of dimension tables**	**#distinct values**
US flights	46	1M	4	26316
Soccer	60	25k	26	122292
Formula one	46	23k	3	13936

Table 6.1.: Information about data warehouses available in EVLIN

[1] https://www.kaggle.com/hugomathien/soccer

[2] https://stat-computing.org/dataexpo/2009/

were applied to the following types of data.

Data warehouses. We resort to one or many real world data warehouses among those described in Section 6.2.2 to evaluate the performance of our proposed approaches.

Real exploration sessions. We collected real visual data exploration sessions. To this end, we collaborated first with researchers in our department. The rationale behind that is to collect a large real history of explorations that could be harnessed subsequently to provide collaborative-filtering recommendations. Our collaborators were also involved in other activities such as rating the interestingness of recommendations and in labeling the similarity between exploration steps that belong to different exploration sessions.

As we will show in Section 6.9, we performed also several user studies where graduate students explored visually the flights data warehouse using our system EVLIN. For each study, we collected the evolution provenance that tracks all exploration steps performed within an exploration session.

Synthetic data. To more systemically evaluate the performance of our approaches, we also used synthetic data in some experiments. For instance, we implemented an exploration session generator that takes a set of real exploration sessions and generates variants for each seed.

6.3. Content-based query recommendation evaluation

As described in Section 5.2, our content-based recommendation approach comprises three steps that are: provenance computation, data recommendation, and query reformulation. In what follows, we conduct a series of experiments to evaluate the three steps in terms of parameter sensitivity, accuracy, and performance.

To do that, we defined nine exploration queries that conform to Definition 3.1. The proposed queries vary in the underlying data warehouse,

Data warehouse	Query	Definition
Flights	Q1	SELECT count(*),C.code FROM Carriers AS C,Tail AS T,Flights AS F WHERE F.uniquecarrier=C.code AND F.tailnum=T.tailnum AND T.manufacturer='BOEING' AND $distance \geq 1500$ GROUP BY C.code
Flights	Q2	SELECT sum(carrierdelay),code FROM Carriers AS C, Flights AS F, Airports AS A WHERE A.iata=F.origin AND F.uniquecarrier=C.code AND A.state='CA' GROUP BY code
Flights	Q3	SELECT avg(distance), A.state FROM Airports AS A, Flights AS F WHERE A.iata=F.origin AND deptime BETWEEN 2000 AND 2200 GROUP BY A.state
Formula 1	Q4	SELECT count(*),C.constructor_nationality FROM result AS R,constructor AS C WHERE C.constructor_id=R.constructorfk_id GROUP BY C.constructor_nationality
Formula 1	Q5	SELECT count(*),R1.race_name FROM result AS R, race AS R1 WHERE R1.race_id=R.racefk_id GROUP BY R1.race_name
Formula 1	Q6	SELECT sum(R.points),D.driver_nationality FROM result AS R, driver AS D WHERE D.driver_id=R.driverfk_id AND $rank > 0$ GROUP BY D.driver_nationality
Soccer	Q7	SELECT avg(goal),L.league_name FROM match AS M, league AS L WHERE M.league_id=L.league_id GROUP BY L.league_name
Soccer	Q8	SELECT count(*),country_id FROM match WHERE result='home_win' GROUP BY country_id
Soccer	Q9	SELECT count(*),country_id FROM match WHERE result='away_win' GROUP BY country_id

Table 6.2.: Set of exploration queries used in the evaluation

the used aggregation functions, the complexity of conditions, and the join combinations. We summarize the used queries in Table 6.2.

Thresholds settings. Our first experiment focuses on a particular step of the content-based query recommendation process that is the *data recommendation* step. This particular step is invoked to compute the set of interesting attribute-values, used later to construct recommended queries.

As shown in Algorithm 1, the *data recommendation* step depends mainly on the lineage threshold θ_L and the database threshold θ_{supp} to filter out uninteresting attribute-values recommendation candidates. Yet, assigning values to θ_L and θ_{supp} is not a trivial task. On the one hand, setting high values to these two thresholds (θ_L and θ_{supp}) will lead to a small number of recommendations and increases thereby the likelihood of discarding interesting attribute-values pairs worth recommending. On the other hand, assigning low thresholds values may lead to a high number of recommended attribute-values pairs that mainly include uninteresting information. Furthermore, used thresholds values of θ_L and θ_{supp} need to be set generically, independently from the explored data warehouses.

To identify suited parameter settings, we perform a parameter sensitivity study by varying the thresholds and studying the effect of this variation on the effectiveness of recommendations.

Recall that following our recommendation approach described in Algorithm 1 (Page 63), we need initially to compute frequencies of all attribute-value pairs present in the lineage. Later, θ_L is used to filter out attribute-value candidates having mediocre frequency values in the lineage. Hence, we need to find a suitable value of frequency that we can use it to filter out irrelevant attribute-value candidates. One possible solution to find a suitable frequency value consists of computing the standard deviation SD of attribute-value' frequencies in the lineage. In this case, the standard deviation measures the dispersion of attribute-value' frequencies from the mean of frequencies in the lineage. Hence, attribute-value candidates whose frequencies are above the standard deviation value may likely belong to the set of relevant candidates that need to be further processed in the subsequent steps of our content-based query recommendation approach. Accordingly, we use standard deviations of attribute-value' frequencies in the lineage to identify interesting attribute-values candidates present in the lineage.

Similarly, based on Definition 5.1 (Page 65), we use θ_{supp} to measure the significant change of an attribute-value pair's frequency in the lineage compared with the frequency of the same pair in the whole database. Accordingly, we use $\mu \in \{\times 1.5, \times 2, \times 3\}$ to express the strength of the frequency's change of an attribute-value candidate in the lineage compared to the whole database. In other words, an attribute-value candidate (a, v) having a frequency in the lineage $f_{a,v}(L)$ and a frequency in the database $f_{a,v}(D)$ is considered interesting either if $f_{a,v}(L) = \mu \times f_{a,v}(D)$ or if $f_{a,v}(D) = \mu \times f_{a,v}(L)$. Consequently, $\theta_{supp} \in \{log_e(1.5), log_e(2), log_e(3)\}$ following Definition 5.2.

Based on these rationales, we construct several settings of threshold values that are the result of the cross product of possible values assigned to θ_L and θ_{supp}. The set of investigated settings are described in Table 6.3.

Configuration	θ_L	θ_{supp}
C_1	SD_L	$log_e(1.5)$
C_2	SD_L	$log_e(2)$
C_3	SD_L	$log_e(3)$
C_4	$2 \times SD_L$	$log_e(1.5)$
C_5	$2 \times SD_L$	$log_e(2)$
C_6	$2 \times SD_L$	$log_e(3)$
C_7	$3 \times SD_L$	$log_e(1.5)$
C_8	$3 \times SD_L$	$log_e(2)$
C_9	$3 \times SD_L$	$log_e(3)$
C_{10}	$4 \times SD_L$	$log_e(1.5)$
C_{11}	$4 \times SD_L$	$log_e(2)$
C_{12}	$4 \times SD_L$	$log_e(3)$

Table 6.3.: Set of thresholds settings

Note that the *data recommendation* step is the crux of our content-based query recommendation approach. Indeed, the set of interesting attribute-value pairs are identified in this particular step. Later these attribute-values are used to derive recommended queries. Accordingly, we study in what follows the impact of threshold configuration on the set of interesting attribute-value pairs output by the *data recommendation* step.

More specifically, we prepare initially a ground truth data set about the interestingness of candidates computed by our *data recommendation* step. To this end, we presented three data analysis experts with the full set of recommended attribute-values generated with low values of lineage and database thresholds values (θ_L set to the fifth of the standard deviations of attributes' frequencies in the lineage and θ_{supp} set to 0.1) when exploring the queries depicted in Table 6.2. Later, we ask the experts to classify each recommended attribute-values as interesting or not interesting with respect to the investigated exploration query. Overall, the three experts investigated around 320 attribute-values pairs recommended when exploring all the exploration queries depicted in Table 6.2. Among these recommendations, they found around 50 interesting attribute-values pairs worth recommending to users. We use in what follows P_{valid} to refer to this particular set of interesting attribute-values pairs found by experts.

Subsequently, we compute the set of recommended attribute-values gen-

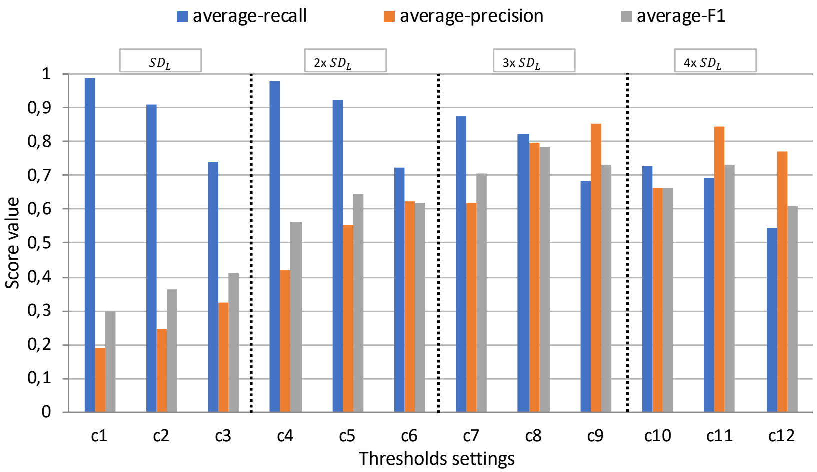


Figure 6.1.: Accuracy study of various data recommendation configurations

erated by our *data recommendation* using various configurations of thresholds that are depicted in Table 6.3. Accordingly, we compare between the ground truth set P_{valid} and P_{rec}: the recommendation sets obtained for each setting. More specifically, we compute for each setting: (i) the recall($= \frac{|P_{rec} \cap P_{valid}|}{|P_{valid}|}$) that counts the ratio of interesting attribute-values pairs retrieved to the total number of interesting attribute-values found by experts; (ii) the precision($= \frac{|P_{rec} \cap P_{valid}|}{|P_{rec}|}$) that counts the ratio of interesting attribute-values pairs retrieved to the total number of attribute-values pairs output using this setting and (iii) F1-score($= \frac{2 \times recall \times precision\}}{recall + precision}$) which is the trade-off score between the precision and recall.

Finally, we average the scores computed for each setting applied to each query present in Table 6.2.

Figure 6.1 depicts the average of recall, precision, and F1-score for the various settings. As expected, we observe a constant drop in recall performance as we adopt moderate to high thresholds settings. Yet, this range of strict settings (except C_{12}) succeeds to maintain acceptable performance of recall (that is always above 0.7).

Contrary to the previous observation, we see that precision performance is improved as we use moderate to high thresholds settings. For instance settings C_8, C_9 and, C_{11} have high precision scores that are around 0.8. Overall, we see in Figure 6.1 that setting C_8 succeeds to maintain a good precision performance (around 0.8) as well as a high recall rate (around 0.8). Accordingly, it has the best F1 score.

Given the above results, we adopt the configuration C_8 ($3 \times SD_L$ and $log_e(2)$) in all subsequent experiments as it was demonstrated to be the best configuration in terms of F1 score.

Evolution of recommendation candidates number. Our second experiment considers the evolution of the number of candidate recommendations through the *data recommendation* step (cf. Section 5.2.2, Page 63).

More specifically, we report in Figure 6.2 three measurements for each query: (i) $Lineage-(a,v)$ refers to the number of candidate attribute-value pairs that are candidates after considering the provenance of r (i.e., candidates remaining at line 4 of Algorithm 1, Page 63), (ii) $DB-(a,v)$ is the number of recommended attribute-value pairs (cf. Definition 5.1, Page 65)

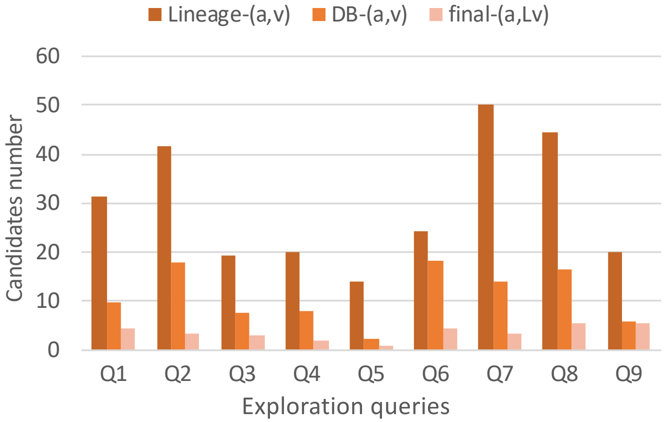


Figure 6.2.: Evolution of candidates number

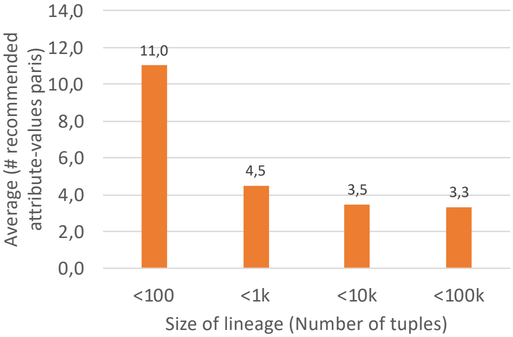


Figure 6.3.: The impact of lineage's size on candidates number

processed at line 7 of Algorithm 1 (Page 63), and (iii) $final-(a, Lv)$ reflects the final number of data recommendations output ultimately by the content-based query recommendation approach. These three measurements are computed for the nine exploration queries written in Table 6.2.

The results depicted in Figure 6.2, indicate that each candidate pruning step is effective and allows to reduce the number of recommendations to be processed by the user to a manageable number. Also, the returned recommendations are less redundant and more concise, potentially having a positive influence on users' satisfaction as we will show later in Section 6.9.

Lineage size impact on recommendation. The inspection of results of the previous experiment reported in Figure 6.2 shows that the final number of recommended attribute-values pairs vary significantly from an exploration query to another. For instance, the final number of recommended attribute-values pairs (a, L_v) in the previous experiment ranges between 1 (case of $Q5$) and 6 (e.g. $Q8$ and $Q9$).

To this end, we study in this experiment a factor that may impact the number of recommendations output using our approach. More specifically, we study now the impact of the lineage size on the number of candidates generated by the *data recommendation* step.

For that, we have computed the average of lineage sizes generated when performing exploration using the seed exploration queries depicted in Table 6.2. Subsequently, we have classified these lineages based on their sizes in four categories: (i) the first range corresponds to explorations where the computed lineage size is less than 100 tuples, (ii) the second range reflects exploration queries whose lineage size is between 100 tuples and 1000 tuples, (iii) the third range reflects explorations queries whose lineage size is between 1000 tuples and 10000 tuples, (iv) the fourth range corresponds to big lineages encompassing a number of tuples ranging between 10000 and 100000.

For each range, we compute the average number of recommended attribute-values output by the *data recommendation* step. Figure 6.3 depicts the variation of the number of the recommended attribute-values depending on the size of lineage. We observe that small lineage leads to a bigger number of recommendations. We explain that as follows: as the lineage size decreases, the data distribution of any attribute present in the lineage becomes more specific and increases thereby the likelihood of getting attribute-value pairs whose frequencies change significantly in the lineage compared with their frequencies in the whole database.

In practice, this means that our content-based recommendation strategy provides a larger set of recommendations when a (seed) query explores specific sub-cubes / sub-regions whereas a very general query (where most of the data in the database is selected as data region to focus on) results in fewer recommendations. This fits the interaction paradigm of our framework, where users select a region of interest.

Runtime measure. Our next experiment examines the runtime of the three main steps of the content-based query recommendation that are *provenance computation*, *data recommendation*, and *query reformulation* for the same nine exploration queries used previously. Each exploration query, presented in Table 6.2, is used as a seed query, first triggering visual recommendation. A user then randomly selects $r \in Q(D)$, an interaction that triggers the content-based query recommendation.

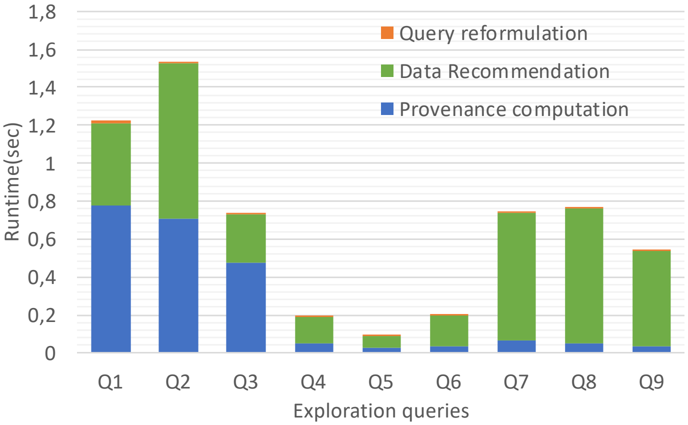


Figure 6.4.: Runtime comparison of recommendation steps

For each exploration query, we tried three different interactions triggering three different recommendation sets. Note that the query reformulation is lazily executed. Indeed, EVLIN reformulates on demand a recommended query once it is selected explicitly by the user over the impact matrix discussed in Section 5.3.3. However, we adjust EVLIN during experiments to implement a batch query reformulation of all pair attributes-values $final-(a, Lv)$ output by the data recommendation step.

Figure 6.4 shows the results, averaging runtimes of each stage over 27 runs (three exploration scenarios for each query) of the experiment. We first observe that the overall runtime of the content-based query recommendation is acceptable (between 0.1s and 1.6s). Next, we see that the time needed for the query reformulation runtime is negligible compared to the other two steps despite that we take into account the worst-case scenario where we reformulate at once all recommended queries. Additionally, we see that the computation times of provenance vary significantly. Hence, it takes between 45 and 65% of the overall runtime for $Q1$, $Q2$, and $Q3$, whereas it is always below 25 % of the runtime for the remaining experiments. This is explained by the fact that the flights data warehouse has a larger size compared to the two other data warehouses. Accordingly, the user has more chance to query large regions of data using the flights data warehouse. This is the case of $Q1$, $Q2$, and $Q3$ queries that target large regions of data. Accordingly, the lineage size is expected to be relatively large compared with lineage sets generated in scenarios exploring the Formula one and the Soccer data warehouses. In general, a large lineage set requires a longer provenance computation time.

Similarly to the provenance computation runtime, we observe that the data recommendation runtime varies from an exploration query to another. For instance, some exploration queries such as $Q1$, $Q2$, $Q3$, $Q7$, $Q8$, and $Q9$ have high data recommendation runtimes while data recommendation is fast for other exploration queries (e.g., $Q4$, $Q5$, and $Q6$). From this result, we observe that provenance computation time impacts in many cases the data recommendation runtime. For instance, $Q4$, $Q5$, and $Q6$ have rapid provenance computation runtime. This means that these queries are associated with small lineage sets. Thereby, the data recommendation will be performed rapidly given the small search space of recommendation candidates. In the same spirit, the exploration queries $Q1$, $Q2$, and $Q3$ have high data recommendation runtimes due to the important size of lineage output by the provenance computation. Indeed, having large lineages (provenance sets) implies a larger search space of candidates of recommendations. In tight comparison between data recommendation runtimes of $Q1$ and $Q2$, we observe that the data recommendation runtime of the exploration query $Q2$ is higher than this of exploration query $Q1$ despite that this latter has the worst provenance computation runtime. This is explained by the larger list of attributes-values candidates of $Q2$ compared to those in $Q1$ as shown in Figure 6.2. Indeed, the discovery of a large list of candidates $Lineage-(a,v)$ in $Q2$ incurs an overhead in the subsequent steps of the data recommendation process (see Algorithm 1, Page 63) where more filtering operations of candidates are performed in comparison with $Q1$. Finally, we observe also that the exploration queries $Q7$, $Q8$, and $Q9$ proposed to explore the Soccer data warehouse, have high data recommendation runtimes despite that they have not high provenance computation runtime. This is explained by the high dispersion of data distribution in the Soccer data warehouse. Indeed, as described in Table 6.1, this particular data warehouse encompasses a huge number of distinct values compared to the other data warehouses. Consequently, we have in this data warehouse a huge number of attribute-value candidates that need to be processed in the course of the data recommendation process.

To summarize, we study our proposed content-based query recommendation approach through a set of experiments discussed in this section. We first investigate suitable threshold settings used to filter out non-interesting candidates during recommendation. Our results reveal that setting the lineage threshold θ_L to $3 \times SD_L$ and the support lineage θ_{supp} to $log_e(2)$, leads to the best performance in terms of F1 score. Hence, this particular setting maximizes the generation of interesting recommendations while minimizing the likelihood of obtaining non-interesting recommendations.

We investigate also the evolution of recommendation candidates' numbers in the course of our content-based query recommendation approach. Results of experiments show that our approach filters out efficiently a considerable number of non-interesting candidates of recommendations. Our experiments show also that the final number of recommendations output to the user varies depending on the lineage size. As the lineage size decreases, the final number of recommendations increases.

Finally, we measure the runtime of the three main steps invoked in the course of our content-based query recommendation approach. Overall, results show that the total average runtime of the three steps is acceptable and it ranges between 0.1s and 1.6s. The analysis of runtime results reveals also a couple of interesting observations. Accordingly, we observe a high runtime of provenance computation when exploring data warehouses having large sizes (e.g., the flights data warehouse). We reveal also that some features of the explored data warehouse (such as the number of unique values) impact the data recommendation runtime. Finally, we observe that a high provenance computation time is always linked with an important data recommendation runtime.

6.4. Quantification of recommendation evaluation

After the computation of the content-based query recommendation, our proposed process of recommendations quantification (cf. Section 5.3) is invoked to assign an interestingness score to each recommendation to help users select the next exploration step to analyze.

In this section, we conduct several experiments to study the effectiveness and the performance of our proposed approach quantifying recommendations' interestingness. Similarly to the previous section, we use the three data warehouses introduced in Section 6.2.2 to evaluate the performance of all proposed approaches. We resort also to the usage of collected real visual data exploration sessions. To this end, we collaborated first with researchers in our department where they were also involved in rating the interestingness of recommendations proposed in the course of exploration.

6.4.1. Effectiveness of quantification methods

In Section 5.3, we have presented our deviation metric adopted for the quantification of recommendation interestingness. It measures the dissimilarity between data distributions of the recommended query's data region and the whole database.

Accordingly, we have discussed possible implementations that could be used to compute the deviation metric. More specifically, we have implemented as discussed in Section 5.3 the following four quantification functions: (i) Euclidean distance (EU); (ii) Chi square (CS); (iii) Kullback Leibler (KL); and (iv) Earth Mover's Distance (EM).

In what follows, we study the performance of these four implementations in terms of theeffectiveness of scores assigned to each recommendation.

To perform this study, we have first tracked 12 exploration sessions made by researchers in our department when exploring the flights, formula one, and the soccer data warehouses. In each exploration session, we have invited our collaborators to select randomly five recommendations to investigate next. Later, we asked them to rate the interestingness of the five recommended queries using a 5-point scale from 1 (not interesting) to 5 (highly interesting). To avoid influencing researchers during the investigation of recommendations, we have used during this experiment a specific version of EVLIN that does not provide quantification of recommendations feature. In other words, participants received at each exploration step a set of not quantified recommendations. This enables participants to choose freely any recommendation that might be interesting or not interesting.

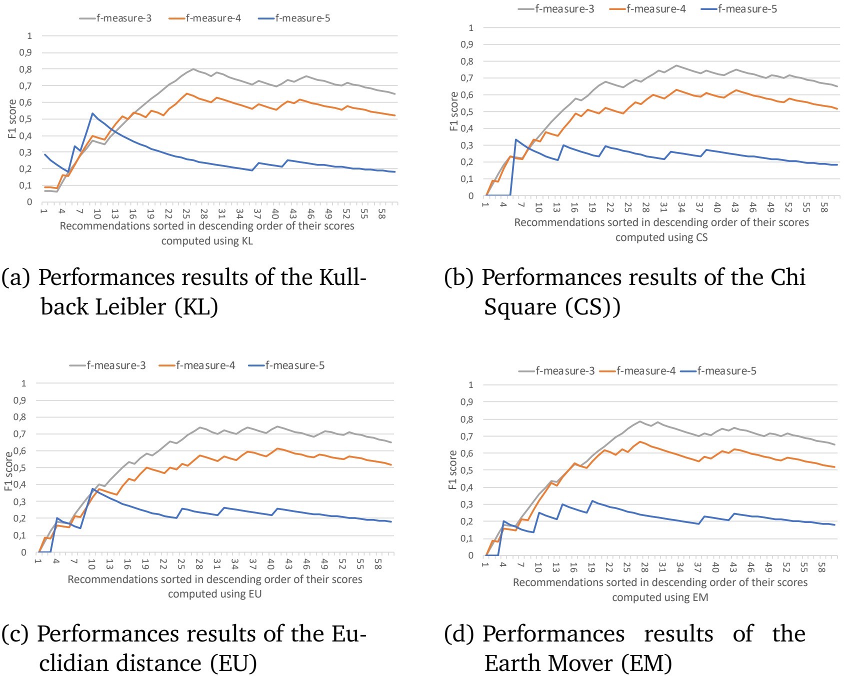
This results in a data set containing 60 rated recommended queries. Table 6.4 encompasses the number of recommendations falling in each rating score value. In what follows, we leverage our collaborators' ratings to study the performance of our four studied quantification functions in terms of information retrieval metrics. To do that, we apply the following process for each studied quantification function.

We sort initially the 60 recommendations in descending order of their interestingness scores computed using each studied quantification function. We then compute the F1 score associated with different sizes of the top-k recommendations. We set initially k to one and we compute F1 score. Afterward, k is iteratively incremented (until reaching 60) and the F1 score is re-computed.

To compute the F1 score at each iteration, we need to identify initially the set of really interesting recommendations to be able to construct later the true positive TP, the false positive FP, the true negative TN, and the false negative FN sets. For that, we define the ground truth by splitting labeled recommendations into two sets using a cut-off value c: (i) the set of interesting recommendations whose ratings values are bigger or equal to c, and (ii) the set of non-interesting recommendations whose ratings values are less than c. Note that we use in the course of this experiment moderate

Rating value	**not rated**	**1**	**2**	**3**	**4**	**5**
Number of rated recommendations	4	15	11	8	15	7

Table 6.4.: Rated explorations



(a) Performances results of the Kullback Leibler (KL)

(b) Performances results of the Chi Square (CS))

(c) Performances results of the Euclidian distance (EU)

(d) Performances results of the Earth Mover (EM)

Figure 6.5.: Comparison of performances of functions implementing the quantification of recommendations' interestingness

to high values of users' ratings ([3,4,5]) as cut-off values to construct the ground truth.

Figure 6.5 depicts the performance of our four implemented quantification functions, computed in terms of F1 score when using various values of users' ratings ([3,4,5]) as cut off c between interesting and non-interesting recommendations. More specifically, each line chart depicted in Figure 6.5 shows the evolution of the following three scores *f-measure-c* with $c \in [3, 4, 5]$). Each score *f-measure-c* corresponds to F1 score computed when considering the recommendations whose user's rating is larger or equal to the value of c as interesting recommendations. It is worth stressing that computing F1 score of a quantification function f for a particular cut off setting c at ranking position k relies on recall and precision metrics computed as follows:

- Recall is computed as $= \frac{TP_{ck}}{TP_c}$ with TP_c is the total number of recommendations considered as interesting following users' ratings (user's rating is bigger or equal to c) and TP_{ck} is the number of interesting recommendations obtained when considering the top-k recommendations sorted in descending order of their interestingness scores computed using the quantification function f.
- Precision is computed as $= \frac{TP_{ck}}{k}$ with TP_{ck} is the number of interesting recommendations obtained when considering the top-k recommendations sorted in descending order of their interestingness scores computed using the quantification function f. The number k is the ranking position. It refers to the top-k recommendations considered at this stage of computation.

Note that we also computed F1 scores also when considering lower users' ratings values as cut-off (e.g., 1, 2). For these particular settings, F1 scores performances are similar for the four studied quantification functions. Hence, we can not conclude which quantification function outperforms others. Accordingly, we decide to omit these results for the sake of figures' simplicity.

For moderate values of cut off (e.g., 3, 4), we observe in Figure 6.5 that the best score values of the F1 score are obtained using Kullback Leibler (KL) and Earth Mover (EM). More specifically, the best f-measure-4 value reaches 0.65 using these two functions (KL and EM) when setting the cut off to 4 while the Kullback Leibler (KL) slightly outperforms the Earth Mover (EM) when setting the cut-off to 3. Indeed, the highest f-measure-3 associated with the latter is 0.78 while the highest f-measure-3 associated with the former is 0.8. Furthermore, we observe that the highest F1 score values recorded for the Kullback Leibler (KL) and Earth Mover (EM) quantification functions are obtained in a similar order when setting the cut off to 3 or 4. For instance, the best F1 score obtained when setting the cut off to 3, is obtained when considering the top-26 recommendations output using the Kullback Leibler (KL) while the best F1 score for Earth Mover (EM) function is obtained when investigating the top-27 recommendations.

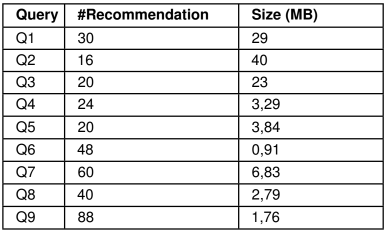
Finally, for high values of cut off (e.g., 5), the Kullback Leibler (KL) outperforms significantly the remaining quantification functions including namely the Earth Mover (EM). Indeed, the highest F1 score value is associated with the Kullback Leibler (KL) when considering the top-10 recommendations sorted in descending order of their values.

To conclude, our experiments show that the Kullback Leibler (KL) and the Earth mover distance (EM) have the best F1 performances compared with the Chi Square (CS) and the Euclidian distance (EU). More specifically, the Kullback Leibler (KL) and the Earth mover distance (EM) have comparable F1 performances when adopting cut off values set to 3 and 4.

Yet, we observe that the Kullback Leibler (KL) outperforms clearly all remaining studied quantification functions (including the Earth mover distance (EM)) when setting the cut off to 5. This indicates that the Kullback Leibler (KL) function provides more robust performance when quantifying the interestingness of recommendations. Accordingly, we adopt in what follows the Kullback Leibler (KL) function to compute the interestingness scores of recommendations output by our system EVLIN.

6.4.2. Performance study

In chapter 5.3.2, we discussed our two optimizations proposed to speed up the computation of interestingness scores (implemented using Kullback Leibler (KL) function) associated with each recommended query output by our system EVLIN.



Query	#Recommendation	Size (MB)
Q1	30	29
Q2	16	40
Q3	20	23
Q4	24	3,29
Q5	20	3,84
Q6	48	0,91
Q7	60	6,83
Q8	40	2,79
Q9	88	1,76

Table 6.5.: Set of exploration queries used in the evaluation

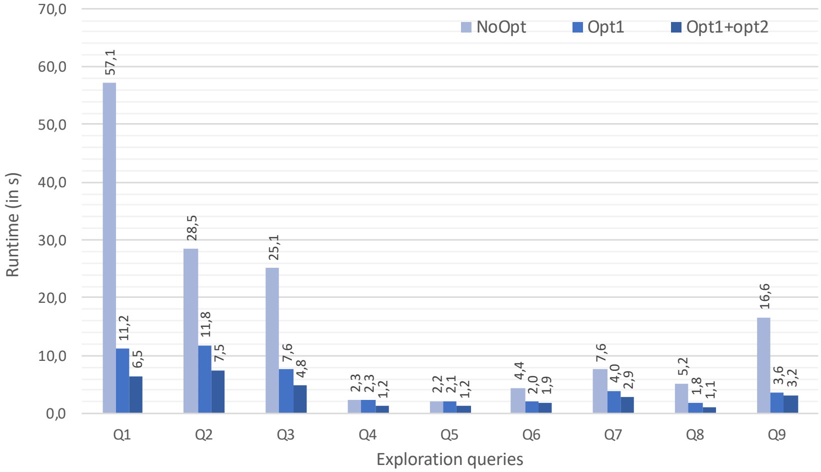


Figure 6.6.: Runtime results of implementations quantifying interestingness of recommendations

As a reminder, $Opt1$ computes two materialized views (once) to then reuse this intermediate result to quantify multiple query recommendations. This is done once a user has selected a data region of interest (see the third step in Figure 5.4, Page 75). $Opt2$ introduces different timing by eagerly computing one of the materialized views prior to the interaction, as it is independent of the selected data region.

Accordingly, we study in this section the impact of our proposed optimizations on quantification runtime when exploring data warehouses. As representative examples, we show runtime results for the nine exploration queries described in Table 6.2. These queries are used to launch nine exploration sessions over the three data warehouses presented in Section 6.2.

Furthermore, we provide Table 6.5 that gives additional information about each exploration query. Indeed, it encompasses the number of recommended queries associated with each exploration query present in Table 6.2 and the size of the initially accessed data region.

Figure 6.6 shows runtime results when quantifying recommendations with and without optimizations. Overall, we observe that the proposed optimizations are effective in reducing the runtime. Indeed, we get runtimes improvement of 80% (e.g., $Q1$, $Q2$, and $Q3$) compared to the runtime of the baseline quantification recommendation method. We observe also that optimization techniques were less effective for $Q4$, $Q5$, and $Q6$. This is explained by the small size of the explored regions and the small number of recommendations of these queries as described in Table 6.5. Recall that the complexity of the recommendation quantification depends on these two factors as discussed in Section 5.3.2. Hence low values of these two factors simplify the task of recommendation quantification that can be computed in acceptable time without optimization. This observation is further confirmed when observing runtimes when quantifying recommendations for $Q7$, $Q8$, and $Q9$. Indeed, these queries have small sizes of their associated explored data regions. Yet, they have a large number of recommendations as indicated in Table 6.5. In this case, our optimizations were effective in reducing the runtimes when quantifying the recommendations of these queries.

We observe that $Opt1$ performs well for most of the depicted exploration queries while $Opt2$ is more effective for exploration queries $Q1$, $Q2$, and $Q3$. Note that this particular range of exploration queries is meant to explore large size of the explored regions as indicated in Table 6.5. This impacts the size of shared views constructed using $Opt1$ that are in their turn large. This leads to a considerable overhead incurred by the expensive construction operation. $Opt2$ succeeds in this case to proactively compute one large materialized view before that user asks explicitly for recommendations. This alleviates later the overhead incurred by the materialization of shared data regions during the quantification of recommendations computation.

Overall, results of experiments discussed in this section show that recommendation's quantification process implemented using Kullback leibler (KL) function succeeds to provide robust performances in terms of accuracy. Furthermore, our experiments show an effective improvement of the efficiency of the recommendation's quantification phase in EVLIN when employing our proposed optimizations. Accordingly, we succeed to speed up the computation of the recommendations' interestingness scores which is crucial for interactive visual exploration.

Finally, we point out that the runtime of the content-based query recommendation (cf. Section 6.3) and the runtime of the interestingness scores quantification add up for the overall visual exploration process supported in EVLIN. Indeed, based on the runtime results reported in Figures 6.4 and 6.6, the overall runtime between interacting with a visualization and obtaining a quantified list of query recommendations takes between 1,5 seconds and 9 seconds which is acceptable (but there is room for improvement).

6.5. Merge of evolution provenance evaluation

We discussed in Section 5.4 our collaborative-filtering query recommendation approach. This latter relies on a multi-user graph obtained by merging periodically the evolution provenance graphs of users' exploration sessions. In this section, we evaluate the performance of our merge approaches (proposed in Section 5.4.2) in terms of runtime, quality, and conciseness.

To do that, we use the US flight data warehouse to evaluate the performance of all proposed merge approaches. This is explained by the fact that the evaluation of proposed merge methods focuses mainly on the exploration session graphs' aspects (e.g., size, connectivity, etc...) rather than the content of the data warehouse. Accordingly, we used the set of exploration sessions recorded initially when exploring the US flight data warehouse to understand the behavior of our proposed merge methods.

Furthermore, we implemented an exploration session generator that takes as input a set of real exploration sessions and generates variants for each seed. This generator is used later to evaluate the performance of our proposed merge methods.

Our first set of experiments studies the performance of three merging techniques described in Section 5.4.3. Table 6.6 summarizes the different merging techniques. For varying similarity threshold θ_{sim} (set to 0.5, 0.7, and 0.9), we study the merge algorithms' runtimes and their fusion rates ($\frac{size\_multi-user\_graph}{\sum size\_users\_exploration\_sessions}$) on 5 exploration session workloads. Each

Name	Description
RLM	The root-layer-merge algorithm described in Algorithm 2
MLM-CA	The match-layer-merge algorithm (Algorithm 4) that avoids conflicts in the incrementally created matching
MLM-CR	The match-layer-merge algorithm that resolves conflicts, taking the matching with the highest similarity score

Table 6.6.: Merge-algorithms used in the provenance aggregator evaluation

workload contains a set of exploration sessions that comprise around 1000 exploration steps. Individual exploration sessions have sizes between 10 and 20 exploration steps. All algorithms iterate over the exploration sessions of each workload, incrementally merging each session into the multi-user graph. Figure 6.7 reports results as averages over the five randomly generated workloads.

Considering the runtime results (secondary y-axis) presented as dashed lines in Figure 6.7, we first observe that RLM is the fastest algorithm. For $\theta_{sim} = 0.5$, both the runtime and merge rate significantly increase compared to more strict thresholds. This is explained by the relaxed threshold similarity value that matches most of the queries to one another. The runtime of both MLM variants is significantly higher than for RLM, due to the larger search space when looking for matches. Yet, this benefits the merge rate, which indicates that the MLM algorithms are more effective in aggregating exploration sessions when operating on moderate to high similarity thresholds.

Comparing MLM-CA and MLM-CR, we observe that MLM-CR takes slightly

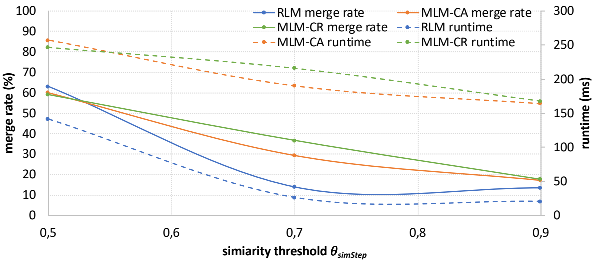


Figure 6.7.: θ_{sim} vs. merge rate (solid lines) and runtime (dashed lines)

more time, which is explained by a low rate of revoked matches. Consistently taking the best match as a starting point for a layered traversal also increases chances of finding matches during this traversal in cases where not a significant number of nodes matches overall (i.e., for thresholds 0.7 and 0.9).

Overall, this experiment shows that RLM outperforms the other studied merge methods in terms of merge rate for $\theta_{sim} = 0.5$, while the MLM algorithms are more effective when adopting high values of θ_{sim}. Thereby, the assignment of the similarity threshold value impacts significantly the choice of the merge algorithm to use. Yet, setting the similarity threshold value is not an obvious task as it may impact the merge quality.

Accordingly, we study the impact of the threshold similarity values (used in the previous experiment) on the merge quality. This is done by studying the evaluation metrics recall and precision.

To do that, we prepared first, eight pairs of exploration session graphs input to merge. These eight pairs of graphs are synthetic exploration sessions generated using our exploration session generator based on real sessions collected when exploring the flights data warehouse. The set of exploration graphs present in these eight pairs vary significantly in size to avoid the impact of this metric on the performance of our studied merge approaches. For each graphs pair (G_1,G_2), we constructed $\mathcal{M}_{complete}$ the full list of possible one-to-one matchings between nodes $n_1 \in G_1$ and $n_2 \in G_2$. Subsequently, we investigated manually the set of pair nodes available in $\mathcal{M}_{complete}$ to construct $\mathcal{M}_{valid}$, the subset of matching candidates worth performing. Note that $\mathcal{M}_{valid}$ may contain overlapping matchings e.g., $\{(n_1, n_2), (n_1, n_2')\} \subseteq \mathcal{M}_{valid}$. This signifies that we have several alternatives to merge the node n_1. Recall that our merge algorithms RLM, MLM-CA and MLM-CR perform 1:1 matching when merging two input graphs. Accordingly, we cluster $\mathcal{M}_{valid}$, e.g., $\{(n_1, [n_2, n_2'])\} \in \mathcal{M}_{valid}$. Consequently, we consider that such merge $\mathcal{M} = (n_1, n_2)$ (output by one of our merge method) is sufficient to consider $\{(n_1, [n_2, n_2'])\} \in \mathcal{M}_{valid}$ as a fulfilled valid matching.

Afterward, we merged each graphs pair using our three merge algorithms. For each merge approach, we tried various similarity threshold θ_{sim} values

Similarity threshold θ_{sim}	**Metric**	**RLM**	**MLM-CA**	**MLM-CR**
0.5	recall	0.45	0.6	0.66
	precision	0.37	0.5	0.53
	F1	0.41	0.55	0.59
0.6	recall	0.45	0.6	0.64
	precision	0.4	0.52	0.54
	F1	0.42	0.55	0.58
0.7	recall	0.43	0.64	0.73
	precision	0.4	0.61	0.66
	F1	0.42	0.62	0.7
0.8	recall	0.32	0.57	0.68
	precision	0.38	0.71	0.81
	F1	0.35	0.63	0.74
0.85	recall	0.32	0.59	0.68
	precision	0.44	0.94	0.94
	F1	0.37	0.73	0.79
0.9	recall	0.32	0.59	0.68
	precision	0.44	0.92	0.97
	F1	0.37	0.72	0.8
0.95	recall	0.29	0.46	0.56
	precision	0.5	1	1
	F1	0.36	0.63	0.71
1	recall	0.29	0.44	0.53
	precision	0.5	1	1
	F1	0.36	0.61	0.69

Table 6.7.: Study of merge quality with varying thresholds and varying merge approaches

(set to 0.5, 0.6, 0.7, 0.8, 0.85, 0.9, 0.95 and 1). At each merge experiment, we collected the set of performed matchings $\mathcal{M}_{performed}$. Those latter are compared to the $\mathcal{M}_{valid}$ set in order to compute recall and precision metrics. More specifically, recall is equal to $\frac{|\mathcal{M}_{performed} \cap \mathcal{M}_{valid}|}{|\mathcal{M}_{valid}|}$ while precision is equal to $\frac{|\mathcal{M}_{performed} \cap \mathcal{M}_{valid}|}{|\mathcal{M}_{performed}|}$.

Table 6.7 reports for different values of θ_{sim}, the recall, precision and F1 results as averages over the eight graph pairs merged using different merge approaches.

Over results available in Table 6.7, we observe first that RLM has always mediocre performances in terms of recall, precision, and F1 scores regardless of the value of θ_{sim}. More specifically, performance scores associated with RLM never reach a value above 0.5.

Contrarily, we see that MLM algorithms have better results. Thus, the MLM variants have highly similar performances with slightly better performance for MLM-CR. Overall, MLM merge family methods outperform RLM in the recall, precision, and, F1 scores regardless of the value of θ_{sim}. Consequently,

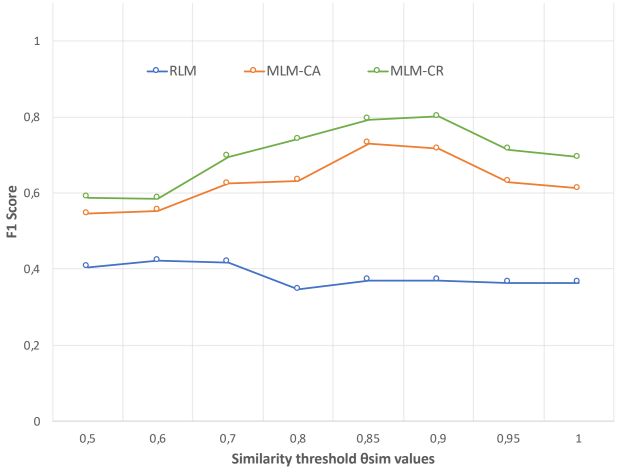


Figure 6.8.: Study of threshold values impact on quality merge methods

the use of MLM algorithms is more appropriate to obtain a good quality of merge.

Our final observation is that the MLM-CR method outperforms consistently the MLM-CA merge method in all the studied quality metrics regardless of the value of θ_{sim}. This is explained by the conflicting matching candidates' policy adopted by the two methods. Hence, MLM-CA prioritizes the first discovered match, even if its distance is higher than the alternative matching candidate. On the other side, MLM-CR performs all matching proposals encompassing the lowest distance. This increases the likelihood of subsequent application of recursive merge between two highly similar sub-graphs.

Overall, Table 6.7 shows that MLM-CR outperforms the other studied merge methods in terms of quality regardless of the similarity threshold value. This is clear when visualizing Figure 6.8 that summarizes the F1 score's value evolution of the three studied merge methods for varying values of the similarity threshold.

Another intriguing observation from Figure 6.8 is related to setting the similarity threshold. Indeed, this figure shows that the range of interest

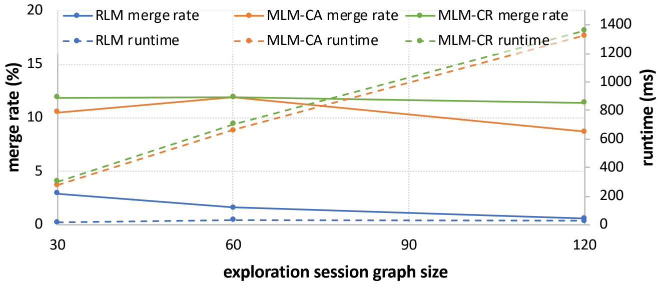


Figure 6.9.: Exploration session size vs. merge rate (solid lines) and runtime (dashed lines)

related to the setting of the similarity threshold is between 0.7 and 0.95 values (where we got good performance of MLM merge family methods). For this particular setting of experiment (where we explore only the flights data warehouse) we see in Figure 6.8 that setting θ_{sim} to 0.9 leads to the best performance of MLM-CR in terms of F1 score in comparison with other results obtained when adopting other similarity threshold values. As a consequence, we set the similarity threshold θ_{sim} to 0.9 in all subsequent described experiments.

Our next experiment studies the impact of exploration sessions' size on the merge rates by varying the number of exploration steps per session. We report results for 30, 60, and 120 explorations steps per session. As before, we generate five random workloads of exploration sessions. Each workload comprises around 1000 exploration steps distributed between exploration sessions of sizes equal to 30, 60, or 120. We study the average performance in terms of runtime and merge rate of our merge methods. All results are presented in Figure 6.9, when setting $\theta_{sim} = 0.9$.

First, we observe that the runtime of RLM remains constant over all exploration session sizes. Associated with this runtime, we observe a constant drop of RLM merge performance. This can be explained by the fact that the root nodes are consistently insufficiently matched, which entails that no matches can be found at deeper layers. As the exploration session's size increases, the number of layers increases as well, so the number of missed matching opportunities increases.

In contrast to the RLM performances, we see that the MLM algorithms' runtimes increase roughly linearly despite their stable merge rates that are roughly constant over all exploration session sizes. This linear behavior of MLM 's runtime is related to the exploration sessions' sizes. Indeed, as the exploration session's size increases, the number of layers increases as well as and their sizes. Hence, this rise of the exploration session's size increases also the likelihood of obtaining large layers. This slows down the runtime of MLM methods as we need to search possible matching in deep layers usually having large sizes.

Our final observation is that the merge rate of MLM-CA method fluctuates, as it first increases to then significantly drop for exploration sessions of size 120. To understand this behavior, recall that in the case of conflicting matching candidates, MLM-CA prioritizes the first discovered match, even if its distance is higher than the alternative matching candidate. Nonetheless, sub-graphs have most likely a big size. Thus, choosing a proposal matching with the lowest distance (that is done in MLM-CR) will lead to a more efficient application of recursive merge between two big and highly similar sub-graphs.

To summarize, our evaluation of merge methods shows that we need to trade off efficiency with merge rate and merge quality. Higher merge rates and qualities are desirable to make more meaningful recommendations in the next step. Also, given that evolution provenance aggregation is performed offline, the runtime is not the main bottleneck. Given these results, we conclude that the MLM-CR approach is the best suited in general as it has comparable runtime to MLM-CA while having more effective merge rates and better merges' quality.

Query	Definition	User's interaction
Q_1	SELECT sum(lateaircraftdelay), T.manufacturer FROM Tail AS T, Flights AS F WHERE F.tailnum=T.tailnum GROUP BY T.manufacturer	$manufacturer$ =Boeing
Q_2	SELECT count(*),A.airport FROM Airports AS A,Flights AS F WHERE A.iata=F.origin AND $deptime < 200$ AND A.state='NV' GROUP BY A.airport	$airport$ =McCarran airport
Q_3	SELECT count(*),A.airport FROM Airports AS A,Flights AS F WHERE A.iata=F.origin AND $depdelay > 0$ AND month=12 GROUP BY A.airport	$airport$ =William Hartsfield-Atlanta Intl
Q_4	SELECT sum(lateaircraftdelay),T.year_t FROM Tail AS T, Flights AS F WHERE F.tailnum=T.tailnum AND T.manufacturer='DOUGLAS' GROUP BY T.year_t	$T.year\_t = 1986$
Q_5	SELECT count(*),A.state FROM Airports AS A,Flights AS F WHERE A.iata=F.origin AND $deptime < crsdeptime$ GROUP BY A.state	$state$ =CA

Table 6.8.: Exploration queries used in the evaluation

6.6. Collaborative-filtering query recommendation performance evaluation

In this section, we conduct a series of experiments to study the performance of our collaborative-filtering recommendation approach (cf. Section 5.4) in terms of parameter sensitivity, and runtime. To do that, we use the flights data warehouse. This is explained by the fact that we need to collect initially a history of explorations targeting the same data warehouse to be able later to perform our collaborative-filtering recommendation approach. Accordingly, we have recorded initially a set of exploration sessions targeting the flights data warehouse.

6.6.1. Collaborative-filtering recommendation setting

As described in Section 5.4, our collaborative-filtering query recommendation approach computes given a current user's exploration within an exploration session G_{XS}, the top-k similar exploration steps present in the multi-user graph G_{MU} with respect to a distance threshold θ_{dist}. Subsequently, our collaborative-filtering approach recommends explorations succeeding those similar exploration steps.

To ensure an efficient collaborative-filtering recommendation, we need to ensure that the set of similar exploration steps identified in the multi-user graph really match to the current user's exploration. Accordingly, we investigate in what follows possible settings of collaborative-filtering recommendation parameters (θ_{dist} and top-k) and their impact on providing an effective collaborative-filtering recommendation. More specifically, we compute the recall, precision, and F1 score for various settings of collaborative-filtering recommendation parameters. To do that, we define first an initial collaborative setting S_0 that sets the distance threshold θ_{dist} to 0.9 and top-k to top-50. This relaxed setting enables us to inspect all possible similar previous explorations in G_{MU}, that can be used to output collaborative-filtering recommendations. For each exploration query Q_{curr} described in Table 6.8, we compute its collaborative-filtering recommendations when setting our baseline collaborative recommender (described in Section 5.4.3.1) following S_0 and based on a real multi-user graph G_{MU} having roughly 950 nodes. This graph results from applying MLM-CR on many individual exploration sessions made by researchers in our department, using a threshold θ_{sim} set to 0.9. Later, we investigated manually all similar exploration steps found using our setting S_0 to label them as interesting or non-interesting items for the collaborative-filtering recommendation. Overall, we investigated the top-50 similar explorations for each exploration query described in Table 6.8. This leads to the construction of the set SIM_{valid} that comprises 11 previous exploration steps in the multi-user graph G_{MU} that are highly similar to the five exploration queries present in Table 6.8. Afterward, we compute again collaborative-filtering recommendations using our baseline collaborative recommender (described in Section 5.4.3.1) applied on all queries present in Table 6.8 with more selective and practically relevant values of the distance threshold $\theta_{dist} \in [0.1, 0.2, 0.3, 0.4, 0.5]$ as well as with various values of $k \in [1, 3, 5, 7]$. At each experiment, we compare the set of similar exploration steps SIM_{rec} output by our collaborative recommendation approach with the explorations steps present in SIM_{valid} in order to compute the following metrics: (i) the recall($= \frac{|SIM_{rec} \cap SIM_{valid}|}{|SIM_{valid}|}$) counts the ratio of similar exploration steps retrieved by our collaborative-filtering recommendation approach to the total number of similar exploration steps found by experts; (ii) the precision($= \frac{|SIM_{rec} \cap SIM_{valid}|}{|SIM_{rec}|}$) counts the ratio of similar exploration steps retrieved by experts as well as by our collaborative-filtering

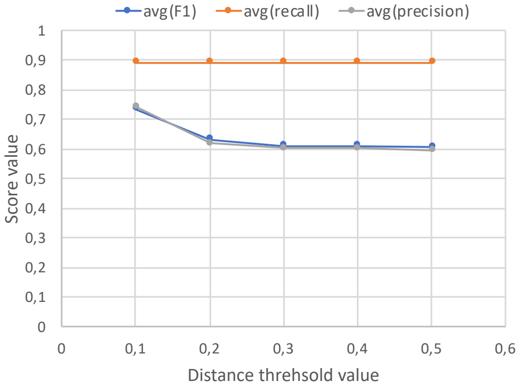


Figure 6.10.: Study of the distance threshold parameter setting in our collaborative-filtering recommendation

recommendation approach to the total number of exploration steps output by our recommendation approach; and (iii) F1-score($= \frac{2 \times recall \times precision\}}{recall + precision}$) is the trade-off score between the precision and recall.

Figure 6.10 reports the results for varying values of the distance threshold θ_{dist}. For each distance threshold, in this graph, we average recall, precision, and F1 scores obtained with various $k \in [1, 3, 5, 7]$ when computing collaborative recommendations of queries described in Table 6.8. We observe first in Figure 6.10 that the recall performance is almost stable regardless of the distance threshold value. This is explained by the fact that our collaborative-filtering recommendation approach stores similar exploration steps found in the multi-user graph G_{MU} in a queue that is sorted in descending order of their similarity distance. Thereby the same order of recommendations is guaranteed regardless of the distance threshold value. Furthermore, we notice that explorations present in the manually labeled set SIM_{valid} have mostly a similarity distance less than 0.1. This means that this set of interesting exploration steps found when computing collaborative-filtering is unchanged mostly for all distance threshold values in [0.1,0.2,0.3,0.4,0.5].

In contrast to the recall performance, we see a significant drop in precision performance when adopting higher distance threshold values. Accordingly, these results impact the F1 score. Indeed, setting a high value of distance threshold leads to acceptable F1 performances whereas setting θ_{dist} to 0.1 leads to the obtention of best F1 performance.

Being aware of θ_{dist}'s impact on the quality of collaborative-filtering recommendation, we investigate now the top-k variation's impact on the accuracy of our approach. We set this time the θ_{dist} to 0.1 given its performance discussed previously. For each top-k $\in [1, 3, 5, 7]$, we average the recall, precision and F1 scores obtained when computing collaborative-filtering recommendations of queries present in Table 6.8 using our baseline recommender approach applied on the same multi-user graph of size 950 nodes with θ_{dist} to 0.1. Similarly to the previous experiment, we compare the set of similar exploration steps SIM_{rec} output by our collaborative recommendation approach with the explorations steps present in SIM_{valid} in order to compute the recall, the precision, and F1 scores (recall= $\frac{|SIM_{rec} \cap SIM_{valid}|}{|SIM_{valid}|}$, precision= $\frac{|SIM_{rec} \cap SIM_{valid}|}{|SIM_{rec}|}$, and F1-score= $\frac{2 \times recall \times precision\}}{recall + precision}$).

Figure 6.11 depicts the evolution of recall, precision, and F1 scores for various settings of k. First, we observe that the recall score increases significantly at the beginning when changing the k from 1 to 3 while this increase of recall score is less significant when passing from 3 to larger values of k. The two observations are explained by the fact that the size of manually identified similar explorations is eleven. This signifies that each query present in Table 6.8 has at most three highly similar exploration steps. Consequently, top-5 and top-7 do not lead to discovering further interesting similar items as our multi-user graph is constructed by the iterative merge of evolution provenance graphs. Accordingly, the rate of similarity in this kind of graph is expected to be lower.

We see that the precision performance drops constantly when increasing k beyond 3. Yet, the precision rate remains acceptable (above 0.7) for all settings.

Based on recall and precision performances, we see that the setting top-3 holds the best F1 score. Indeed, it succeeds to maintain good performance for both recall and precision.

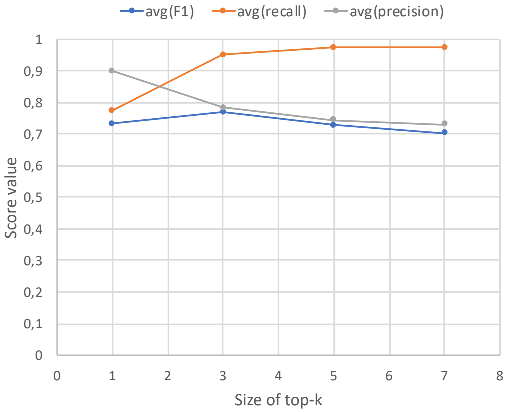


Figure 6.11.: Study of top-k parameter setting in our collaborative-filtering recommendation

To summarize, our evaluation of collaborative-filtering recommendation settings shows that setting the distance threshold θ_{dist} to 0.1 and when applying top-3 similarity search leads to the obtention of a good quality of recommendations that match with users' current explorations. Note that experiments made in this section are only based on exploration sessions and evolution provenance tracked when exploring the flights data warehouse. Hence, we intend in the future to explore other data warehouses and to provide a more thorough quantification mechanism to study extensively the appropriate parameters required to set our collaborative-filtering recommendation approach.

6.6.2. Runtime and scalability of the collaborative-filtering recommendation

In this section, we evaluate four collaborative-filtering recommendation implementations discussed in Section 5.4.3 which are: (i) REC-simple referring to the baseline recommender; (ii) REC-triangle implementing the triangle inequality discussed in Section 5.4.3.2; (iii) REC-parent implementing the

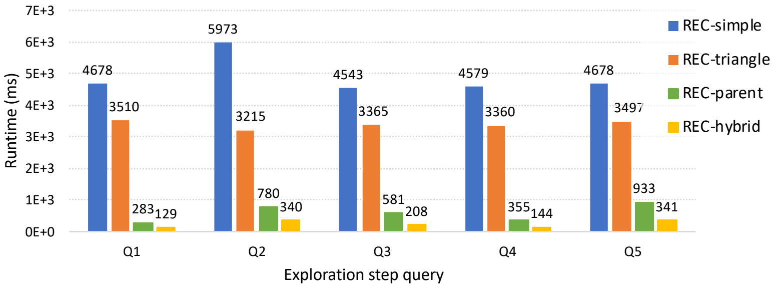


Figure 6.12.: Collaborative-filtering recommendation runtime

parenthood based pruning discussed in Section 5.4.3.3; and (iv) REC-hybrid that combines both triangle inequality and parenthood pruning.

As the recommendation algorithms rely essentially on a current exploration step within an exploration session G_{XS}, we evaluate them using G_{MU} and randomly generated exploration queries that simulate the current exploration step. Table 6.8 shows five representative queries, for which we provide detailed results. Table 6.8 also indicates which interaction triggers the recommendation process for which we compute top-3 recommendations using our four recommendation algorithms and using a distance threshold θ_{dist} set to 0.1. We apply these methods on a multi-user graph G_{MU} having roughly 950 nodes. This graph results from applying MLM-CR on many individual exploration sessions whose sum size is around 1000, using a threshold θ_{sim} set to 0.9.

Figure 6.12 presents the runtime of each recommendation algorithm. As expected, REC-simple has the highest runtime (≈5s) that remains constant over all queries as it always traverses the full graph G_{MU}. REC-triangle moderately improves on this runtime, with again a stable runtime for all queries. This performance is explained by the small number and size of clusters and the fact that the pruning performance is determined by the number of pre-computed clusters of highly similar exploration steps in G_{MU} that does not vary for different exploration step queries. For REC-parent, we observe that it improves on the runtime of REC-simple by up to an order of

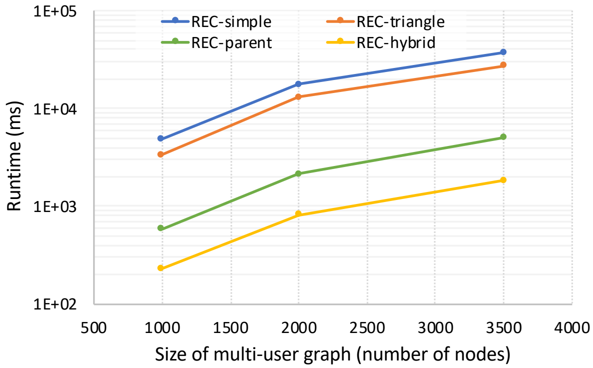


Figure 6.13.: Collaborative-filtering recommendation runtime for varying G_{MU} sizes

magnitude, clearly showing the effectiveness of parenthood-based pruning. This performance can be slightly improved when combining both pruning methods, as shown by the bars for REC-hybrid.

We also study the scalability of the recommendation algorithms for increasing sizes of G_{MU}. To this end, we generate multi-user graphs with 1000, 2000, and 3500 exploration steps, respectively. For each new multi-user graph, we have computed the average runtime made by each studied recommendation method executed for the same exploration queries described in Table 6.8. Figure 6.13 reports the average runtime over the five exploration queries for each algorithm for the different sizes of G_{MU}.

As expected, the runtime of REC-simple increases with the size of G_{MU}. For REC-triangle, we see a runtime evolution that is comparable to those depicted for REC-simple. However, as the size of G_{MU} increases, the gap between REC-simple and REC-triangle becomes more significant. Looking at the number and the size of clusters obtained for the triangle inequality pruning technique, we notice that as the multi-user graph size increases, the number of clusters increases giving thereby more opportunities to apply the triangle inequality pruning technique. Similarly, the parenthood-based pruning used both by REC-parent and REC-hybrid becomes more effective at larger graph sizes. Increasing the graph size also increases the average path length in the graph, giving the parenthood-based pruning more opportunities to prune comparisons, compared to smaller graphs.

Overall, the results of experiments discussed in this section show an effective improvement of the efficiency of the collaborative-filtering recommendation computation when employing our proposed optimizations (the triangle inequality pruning and the parenthood-based pruning). Accordingly, we succeed to speed up the computation of the collaborative-filtering recommendations which is crucial for interactive visual exploration.

6.7. Collaborative-filtering query recommendation usability

While the previous section focused on a quantitative performance analysis of our proposed collaborative recommendation methods, this section evaluates the impact of introducing our collaborative-filtering recommendation method in the conduct of the visual data exploration. We first validate in this section that the collaborative-filtering technique is effective in diversifying and improving the recommendations generated by our system EVLIN. Subsequently, we assess in Section 6.9 users' satisfaction when visually exploring data warehouses based on a user study.

Similarly to the previous section, we use in this experiment the US flight data warehouse to evaluate the effectiveness of our collaborative-filtering query recommendation technique in diversifying and improving the recommendations generated by EVLIN. This is explained by the fact that we need to collect initially a history of explorations targeting the same data warehouse to be able to perform our collaborative-filtering recommendation approach. Accordingly, we have recorded a set of exploration sessions targeting the US flight data warehouse.

Towards studying the impact of collaborative-filtering in diversifying recommendations, we study first the distribution of recommendation scores with (and without) collaborative recommendations.

To this end, we performed first a content-based recommendation for exploration steps specified in Table 6.8. These recommendations are quantified subsequently using our *recommendation quantifier* component (see Section 5.3). Recall that this latter implements Kullback Leibler (KL) to compute the deviation metric between recommendation's data region and the whole database as a utility score.

Afterward, we compute the top-3 collaborative-filtering recommendations for the same exploration steps defined in Table 6.8 using our REC-hybrid approach. Recommendations generated by this approach are used to update interestingness scores computed initially using the Kullback Leibler (KL).

Interestingness scores computed with (or without) collaborative-filtering recommendation approach are then split into four ranges (i) [0-0.25] corresponds to scores that are less or equal to 0.25, (ii)]0.25-0.5] corresponds to scores that are bigger than 0.25 and less or equal to 0.5; (iii)]0.5-0.75] represents scores ranging between 0.5 and 0.75; and (iv)]0.75-1] represent high interestingness scores larger than 0.75. Accordingly, we cluster interestingness scores computed with (or without) a collaborative-filtering recommendation approach by their corresponding ranges. This leads to the generation of the horizontal mirror histogram depicted in Figure 6.14. This visualization compares the average distribution of the interestingness scores ranges of recommendations computed for the exploration steps defined in

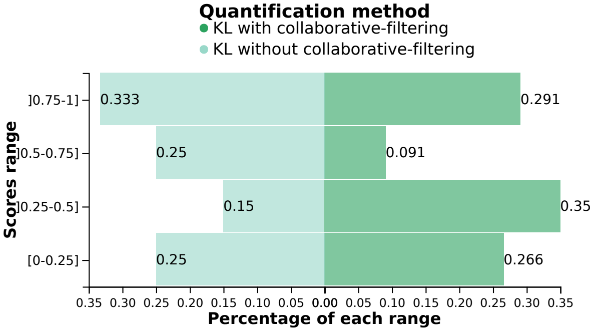


Figure 6.14.: Accuracy study of various data recommendation configurations

Table 6.8 using the Kullback Leibler (KL), with (or without) collaborative-filtering recommendation approach.

The analysis of the left side of this figure shows that all histograms associated with the quantification method, without collaborative-filtering encompass high density of interestingness scores whose values belong to the two ranges]0.5-0.75] and]0.75-1]. This indicates that adopting only the deviation metric (e.g., Kullback Leibler) to quantify recommendation is not sufficient as the user is left likely with a large number of highly interesting recommendations.

On the right hand side of Figure 6.14, we observe that the blending deviation metric (computed using the Kullback Leibler) with the collaborative-filtering recommendations' information contributes to the decrease of interestingness scores whose values belong to the ranges]0.5-0.75] and]0.75-1]. This indicates that the collaborative-filtering technique succeeds to improve the distinction between recommendations having high scores and those having low scores.

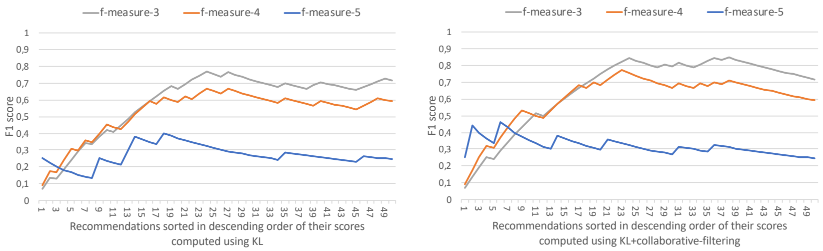
We show so far that the introduction of our collaborative-filtering recommendations approach leads to a significant change of interestingness scores distribution. In what follows, we study the impact of this change on the accuracy of quantifying recommendations.

To do that, we compare now the F1 score performance of (i) the quantification method implemented using Kullback Leibler (KL) and (ii) the mixed quantification that blends the interestingness scores computed using the Kullback Leibler function with those output by our collaborative-filtering recommendation approach.

Similarly to the experiment described in Section 6.4.1, we first tracked 10 exploration sessions made by researchers in our department when exploring the flights data warehouse. In each exploration session, we invited our collaborators to rate the interestingness of five recommended queries using a 5-point scale from 1 (not interesting) to 5 (highly interesting). Note that we didn't use already rated exploration sessions discussed in Section 6.4.1. This is due to the fact that sessions labeled in Section 6.4.1 concern the three data warehouses presented in Section 6.2.2 whereas the employment of collaborative-filtering in our current experiment requires only the flights data warehouse for it we maintain a multi-user graph. To avoid influencing researchers, we used during this experiment a specific version of EVLIN that does not support the quantification of recommendations to avoid influencing participants during this experiment.

This results in a data set containing 50 rated recommendations. Afterward, we compute interestingness scores of these 50 recommendations using our two quantification approaches: (i) the first quantification method is based only on the Kullback Leibler (KL), and (ii) the second quantification method blends the interestingness scores computed using the Kullback Leibler function with the collaborative-filtering scores. For the second approach, we compute top-3 collaborative recommendations using the REC-hybrid implementation applied on a multi-user graph containing 3300 nodes. Subsequently, we leverage our collaborators' ratings to evaluate our two studied approaches of recommendations' quantification in terms of F1 score.

To do that, we apply for each studied quantification approach the same process described in Section 6.4.1. More specifically, we sort initially the 50 recommendations in descending order of their interestingness scores computed using each studied recommendations' quantification approach. Next, we compute the F1 score (using the same formulas described in Section 6.4.1) associated with different sizes of the top-k recommendations. We set initially k to one and we compute F1 score. In the course of the experiment, k is incremented iteratively (until reaching 50) and the F1 score is re-computed at each step. To obtain the F1 score at each iteration, the F1 score of a quantification approach f, for a particular cut off setting c ($\in [3, 4, 5]$), at ranking position r, we compute recall and precision metrics using the same formulas specified in Section 6.4.1. In a similar way to the experiment described in Section 6.4.1, we use in the course of this experiment three possible values of users' ratings ([3,4,5]) as possible cut-off values to define the set of recommendations deemed as interesting. Accordingly, we repeat the process of F1 value computation for varying sizes of top-k recommendations and for each possible value of users' ratings ([3,4,5]). This leads to the computation of f-measure-3, f-measure-4, and f-measure-5 that



(a) Quantification process without the collaborative-filtering recommendation

(b) Quantification process with the collaborative-filtering recommendation

Figure 6.15.: Comparison of performances between the two quantification methods

correspond to F1 scores computed when setting the cut off value respectively to 3, 4, and 5.

Figure 6.15 depicts the evolution of the F1 scores' values of our two studied recommendations quantification processes for various values of the cut-off. We observe in this figure that the best score values of the F1 score are always obtained using the mixed quantification approach regardless of the value of the cut off. More specifically, the best F1 score value reaches 0.84, 0.77, and 0.46 using the mixed quantification approach when setting the cut off to 3, 4 and 5 respectively while best values of F1 scores associated with the quantification approach based only on Kullback Leibler (KL) reach 0.75, 0.65 and 0.4 when setting the cut off to 3, 4 and 5 respectively. Note also that the best F1 scores recorded for the mixed quantification approach are obtained earlier compared with the quantification approach implementing only the Kullback Leibler (KL). For instance, when setting the cut off to 4, the best F1 score of the mixed quantification approach is obtained when considering the top-23 recommendations while we need to investigate the top-28 recommendations sorted in descending order using scores output by the quantification approach implementing only Kullback Leibler (KL). This same observation holds when setting the cut off to 5. Indeed, we need to investigate only the top-6 recommendations of the mixed quantification approach to reach the best F1 score performance whereas the best F1 score associated with the Kullback Leibler quantification approach is obtained when reaching the top-18 recommendations.

Overall, we conclude from this experiment that taking into account the collaborative-filtering recommendations when quantifying recommendations contributes to an improvement of the F1 score. This leads thereby to the improvement of accuracy of quantification of recommendation process and indicates that we have now a better distinction between interesting and non-interesting recommendations. Based on this observation, we expect a more effective visual data exploration experience when introducing the collaborative-filtering recommendation in our system EVLIN. To this end, we investigate the impact of extending our system EVLIN with the collaborative-filtering recommendation in the users' experience when exploring visually data warehouses.

6.8. Evaluation of visualization recommendation

In what follows, we evaluate quantitatively the performance of the visualization recommendation computation in terms of runtime and consistency of recommended visualizations.

Similarly to Sections 6.3, and 6.4, we use the three data warehouses presented in Section 6.2.2 to evaluate the performance of our proposed visualization recommendation approach.

Runtime. We study first the runtime of the three main steps (depicted in Figure 5.11, Page 105) that are invoked when recommending visualizations. To do that, we resort to the exploration queries described in Table 6.2 (Page 117). These exploration queries are used in the course of our experiment by our collaborators to launch various exploration sessions.

As shown in Algorithm 7 (Page 106), our visualization recommendation approach relies mainly on the exploration path available in the evolution provenance that tracks user's previous explorations, to recommend consistent visualizations. Consequently, we compute the runtimes of visualization recommendations for varying sizes of exploration paths (ranging from an empty set to an exploration path containing four exploration steps). We choose to stop at the exploration path of size equal to four as our user study discussed in the next section shows that the real-world exploration sessions give rise to shallow and wide graphs with a larger number of short paths rather than a small number of long paths. This is explained by the fact that users typically use the *recover* interaction (see Automaton 3.2, Page 40) to get back to a previously seen exploration step and proceed from there.

Using the various sizes of exploration paths, we measure the runtime of the three steps of the visualization recommendation process (cf. Figure 5.11, Page 105).

Figure 6.16 reports the average runtime of these three steps for exploration sessions launched using queries described in Table 6.2 (Page 117). The first step is called *PathComp*. It corresponds to the computation of the exploration path leading to the current exploration query. The second step *SortPath* maps to the phase of sorting queries present in the path in descending order of their similarity with the current inspected query. Finally, we compute the runtime of *ComputeViz* that corresponds to constructing the recommended visualization using visual encodings of visualizations previously seen.

The results depicted in Figure 6.16 show first that overall, the computation of the visualization recommendation is fast. Indeed, its runtime ranges between 1 and 20 ms. It is thus negligible compared to the previously

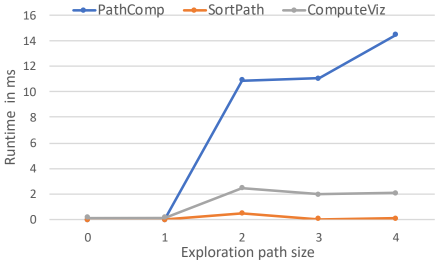


Figure 6.16.: Runtime of main steps of the visualization recommendation process

discussed query recommendation time.

We observe that the runtime of the overall process of visualization recommendation is almost equal to zero when the exploration path is empty. This is due to the fact that the evolution provenance is empty and there are no paths to explore at the beginning. Accordingly, the runtimes of computation of the exploration path and its sorting are negligible in this case. Similarly, we have an empty history of previous user visualization resources. So, *ComputeViz* uses a predefined template to design the recommended visualization. This leads thereby to a negligible runtime of the *ComputeViz* step.

We observe also in Figure 6.16 that runtimes of *SortPath* are overall negligible at all stages of the experiment. This is due to the small size of exploration paths generated alongside users' explorations (between 0 and 4). This small size of exploration paths is explained by the fact that, users typically use the *recover* interaction (see Automaton 3.2, Page 40) to get back to a previously seen exploration step and proceed from there.

We see also that the step *PathComp* is done in acceptable times. This runtime increases significantly starting from exploration path of size equal to 1. This is explained by the increase of the evolution provenance graph's size. As the user inspects more queries' results through the exploration session, the size of the evolution provenance graph increases also. Accordingly, the time of searching the exploration path that starts from the initial user's query to reach ultimately the current exploration, is more important.

Regarding the step *ComputeViz*'s runtime, we observe in Figure 6.16 that the runtime of this step is almost stable alongside the visual data exploration process (except for the first step of exploration). Recall that the runtime of the *ComputeViz* step is impacted by the size of the exploration path. Indeed, the *ComputeViz* step accesses iteratively to the visualizations associated with queries present in the exploration path to complete visual encodings not yet specified in the current visualization to recommend. Once the specification of the current visualization to recommend is complete, the *ComputeViz* skips remaining elements present in the exploration path and returns the recommended visualization. In this context, our subsequent experiments (see Experiment 6.8) show that the $top-2$ visuallizations present in the sorted exploration path are usually sufficient to construct the recommended visualization. This explains the almost stable runtime of step *ComputeViz* where we use almost two or three previous visualizations regardless of the size of the exploration path to design the recommended visualization.

Consistency rate. As described in Section 5.5, our novel visualization recommendation approach leverages a new metric that is consistency to render a set of coherent visualizations alongside the exploration session. Accordingly, we have studied the consistency between recommended visualizations rendered within the same exploration session. Recall that in the previous experiment, we have generated several exploration sessions using exploration queries described in Table 6.2 (Page 117). In the course of these exploration sessions, we have tracked the set of previous visualizations that our proposed recommendation approach used to generate a current recommended visualization with varying sizes of the exploration path.

Figure 6.17 shows the average number of previous exploration steps involved when designing recommended visualizations for varying sizes of the exploration path. Overall, results show that the consistency rate increases with the size of the exploration path. Indeed, as we advance in the visual data exploration session, the evolution provenance graph increases giving thereby the visualization recommendation more opportunities to re-use more previous visualizations and thereby more existing visual encodings.

As our recommendation process may re-use more than one visualization

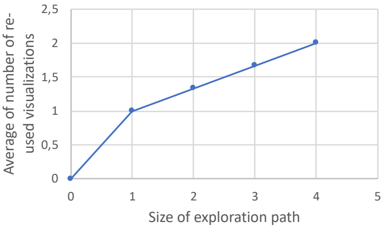


Figure 6.17.: Number of previous visualizations re-used when recommending visualizations

component from a previous visualization to render the current investigated exploration step, we have also tracked in the previous experiment the rate of re-used axes and visual encodings channels. We explain our choice of these two components (cf. Figure 4.1, Page 50) as follows: (i) axes provide a reference for reading the visual/data mapping, and (ii) the visual encoding channel is the fine-grained component of a visualization. It describes the visual mappings of each concept (or information). Recall that consistency metric (as defined in Figure 5.5.2, Page 103) consists of presenting the same information (or concept) in the same way. Thereby, tracking axes and visual encodings channels is useful to estimate the coherence of representing same concepts in the course of an exploration session and to quantify thereby the consistency metric. For each possible value of the exploration path size $p_{val} \in [0, 1, 2, 3, 4]$, we record a snapshot $snap_{viz}$ of the visualization resources relations recorded in the evolution provenance graph (see Figure 4.1, Page 50). $snap_{viz}(p_{val})$ contains all visualization constructed by our visualization recommender until reaching an exploration size equal to p_{val}. For each possible value of the exploration path size p_{val}, we access its snapshot $snap_{viz}(p_{val})$ to compute: (i) reused axes rate($= \frac{\#used\_axis - \#constructed\_axis}{\#used - axis}$) that compares between the number of used axis (available in the relation *UsageAxis* in $snap_{viz}(p_{val})$) and the fictive number of constructed axis (available in the relation *Axis* in $snap_{viz}(p_{val})$), and (ii) reused visual encoding channels rate($= \frac{\#used\_channels - \#constructed\_channels}{\#used - channels}$) compares between the number of used encoding channels (available in the relation *UsageChannel* in $snap_{viz}(p_{val})$) and the fictive number of constructed encoding channels (available in the relation *Usage* in $snap_{viz}(p_{val})$).

Figure 6.18 reports results of our two metrics for varying sizes of exploration paths. This figure shows that the rate of re-using existing axes and visual encoding channels increases with the size of the exploration path. Indeed, the rates of re-using axes and visual encoding channels reach 38% and 30% consecutively when the exploration path contains four exploration steps.

Overall, the evaluation study in this section shows that our visualization recommendation approach designs visualizations in acceptable time that

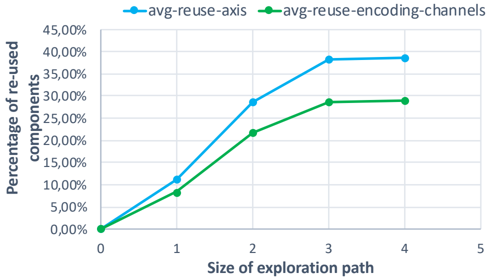


Figure 6.18.: Evolution of consistency rate when recommending visualizations

ranges between 1 and 20 ms. We demonstrate also that our visualization recommendation takes into account previous rendered visualizations to render explorations results in a consistent way.

6.9. User study

This section discusses a comparative evaluation of (i) EVLIN, supporting all contributions discussed in Chapter 5 except of the collaborative-filtering query recommendation and; (ii) EVLIN++, supporting all contributions discussed in Chapter 5 including the collaborative-filtering query recommendation, and (iii) Voyager [WQM+17], a state-of-the-art visual data exploration tool that offers query recommendations with associated visualization recommendations, which is the most similar experience to our system that we identified in our literature review. Voyager consists of a dashboard that initially introduces the explored dataset (e.g., the structure of the dataset including its measures and dimensions and also some generic visualizations summarizing the data). Users can select interactively the set of attributes of interest to explore. Based on the selected attributes, Voyager recommends a set of exploration queries strongly related to the user's initial selection. Accordingly, it generates a large number of recommended visualizations (that render recommended queries' results) and organizes them by relevance. Users then analyze this large set of simultaneously rendered visualizations, can bookmark interesting ones, and can then continue to refine or adapt their selection of attributes to get further recommendations.

We compare the three approaches based on a user study. We recruited 14 graduate students who are familiar with data analysis practices to explore the flights data warehouse. They were equally assigned to two groups. The first group used EVLIN only. The second group used both EVLIN++ and Voyager (except one participant who only used EVLIN++). To study the effect of collaborative-filtering, prior to the user study, we collected 180 real exploration sessions made by researchers in our department when exploring the flights data warehouse. We then applied MLM-CR to merge these sessions with $\theta_{sim} = 0.9$. This entails a multi-user graph having 2900 exploration steps to be used by the collaborative recommender.

Study protocol. To familarize participants with the different systems, we first let them use the systems they shall focus on (EVLIN for one group, EVLIN++ and Voyager for the other) on a different data warehouse containing Formula one data (cf. Section 6.2). After acquiring a general understanding of their respective systems, the participants of the first group explored the flights data warehouse for 25 minutes using EVLIN and rated investigated recommendations using a 5-point scale from 1 (not interesting) to 5 (highly interesting). User of the second group performed the same task using EVLIN++. In addition, 6 participants of this group also used Voyager for 25 minutes to explore the same data warehouse. Voyager does not support rating visualizations, we thus use its bookmarking feature to let participants label recommended queries with their visualizations as interesting. Finally, users evaluating EVLIN++ and Voyager filled out an exit survey meant to understand users' preferences for the two tools. Note that for all experiments, we did not ask participants to investigate a specific topic, giving them full freedom for their visual data exploration session.

For each exploration session, we capture which exploration steps were performed (as the evolution provenance in EVLIN and EVLIN++ and via logging of Voyager) and which step is associated to which rating or a bookmark. We also track how many interactions users perform on rendered

Metric	**EVLIN**	**EVLIN++**	**Voyager**
#exploration steps	132	107	n.a.
#rendered visualizations	198	197	253
#interesting findings for different ratings (3/4/5)	85/60/22	103/81/48	57
average visualizations per session	**28.3**	**28.1**	**42.2**
average rating per session	**3.24**	**3.95**	n.a.
average ratio of findings to #exploration steps per session (3/4/5)	**0.59 / 0.41 / 0.178**	**0.76 / 0.53 / 0.355**	n.a.
average ratio of findings over visualizations per session (3/4/5)	**0.41/ 0.28 / 0.12**	**0.77 / 0.4 / 0.24**	**0.22**

Table 6.9.: Comparison between EVLIN, EVLIN++, and Voyager

visualizations.

Study results. We summarize results obtained in the user study in Table 6.9. The first two rows report the total number of exploration steps across all exploration sessions and the number of rendered visualizations (higher than the number of exploration steps as more than one visualization can be recommended at each step). In subsequent lines, when we write (3/4/5), we report numbers relating to interesting findings for EVLIN and EVLIN++ that consider the three possible cut-offs for interesting recommendations (i.e., rating ≥ 3, rating ≥ 4 or rating ≥ 5). As no ratings are available in Voyager, we consider all bookmarked results of Voyager as interesting. In particular, for different rating thresholds to split interesting findings from uninteresting visualizations, we report the overall number of interesting findings, the average ratio of $\frac{\#findings}{\#explorationsteps}$ per session, and the average ration of $\frac{\#findings}{\#visualizations}$ per session. As the interactions of Voyager do not follow individual exploration steps, no corresponding number is reported for Voyager. We further show the average rating per session obtained by both EVLIN and EVLIN++. No equivalent can be reported for Voyager, as it only allows to bookmark a visualization as interesting.

We further discuss the results highlighted in blue in Table 6.9. Our observations are all verified to be statistically significant using ANOVA.

EVLIN vs EVLIN++ Firstly, we notice that the average number of visualizations per session shown to users is comparable for both EVLIN and EVLIN++. This is expected as these are two versions of the same system with the same interfaces, differences concern only the recommendation methods in the backend. Among the recommended and visualized queries, we see that the quality, quantified by the average rating per session, is significantly higher (approximately 20%) when using EVLIN++ than when using EVLIN. This corroborates our previous conclusion that integrating collaborative-filtering recommendations improves the quality of the overall recommendations.

Next, we study the average ratio of $\frac{\#findings}{\#explorationsteps}$ per session. This corresponds to the probability that an exploration step encountered in a session is interesting. Independently of whether we consider as interesting exploration steps rated with at least 3, 4, or 5 , we find that EVLIN++ significantly increases the likelihood of obtaining an interesting recommendation in a user session. For instance, considering only the most highly rated queries as interesting, an average of 35% of queries encountered in a session are interesting findings when using EVLIN++, as opposed to 18% for EVLIN.

Overall, comparing EVLIN and EVLIN++, we conclude that EVLIN++, which blends collaborative-filtering and content-based recommendations significantly improves the quality of recommendations for interactive visual exploration compared to EVLIN.

EVLIN++ vs Voyager Let us now focus on how EVLIN++ compares to Voyager. Our first observation is that users are exposed to more visualizations per session using Voyager (42.2) than using EVLIN++ (28.1). This is due to the "brute-force" approach adopted by Voyager that essentially recommends all possible extensions of the user's initial query. In contrast, EVLIN++ adopts a focused recommendation approach where only queries deemed interesting are recommended.

Next, we analyze the number of interesting findings obtained using EVLIN++ and Voyager. We see that despite users of Voyager are exposed to more visualizations per session, the number of interesting findings output by Voyager is not significantly higher. When looking at the average $\frac{\#findings}{\#visualizations}$ per session, we see that even when considering only visualizations associated to queries rated with 5, EVLIN++ increases the likelihood of surfacing important findings per visualizations that reaches 0.24, compared to Voyager (0.22). Accordingly, these results indicate that the participants spent less effort (number of inspected visualizations) using EVLIN++ to surface important information.

We have also analyzed surveys filled out by participants. We find firstly equitable results of the two systems in terms of visualization quality (50% for each). In terms of usability, 44.4% of participants prefer EVLIN++ while 55.6% opt for Voyager. Thus, many participants liked the drag and drop interaction supported by Voyager to specify attributes to explore. Other participants liked the exploration process ensured by EVLIN++ where users can navigate smoothly between exploration steps via simple interactions. In terms of recommendation interestingness, 57.14% of participants believed that EVLIN++ generates more interesting recommendations while 42.86% opt for Voyager. Indeed, most of the users opting for EVLIN++ agreed on the fact that the underlying system provides more focused recommendations that comply with users interactions. Finally, we asked participants about the global experience of visual data exploration including learning and discovering important insights. Our survey analysis shows that 62.5% of participants prefer EVLIN++ while 37.5% favor Voyager. This result is possibly related to the sheer number of recommendations generated by Voyager that may overwhelm users. Indeed, we find in the survey users' replies confirming that, e.g., "...Voyager had a cluttered interface" and "...too many visualizations lead to a clutter". On the other hand, many participants appreciate the capability of EVLIN++ to dig into details. This is not the case for Voyager where users need to adjust manually the visual and the data panels to pursue such interesting exploration. Thus, we find comments related to this statement e.g., "EVLIN is very useful to go in depth on your data" and "EVLIN is better to get details and to construct a full exploration story". Additionally, some participants opting for EVLIN++ deemed interestingness scores for recommendations, quite helpful to surface important insights. Thus, one participant noted that "The interestingness matrix was helpful as it points out highly interesting recommendations". Overall, this study shows a general satisfaction among participants regarding EVLIN++.

6.10. Conclusion

In this chapter, we performed a series of evaluations to validate our contributions implementing the *provenance engine* and the *recommendation engine* modules of our framework ProvVDE (cf. Figure 3.1) in our system EVLIN. More specifically, we evaluated first our content-based query recommendation approach and our methods quantifying recommended queries. Second, we investigated our merge methods meant to fuse evolution provenance graphs, our collaborative-filtering query recommendation method and its impact on improving the quantification of recommendation. Finally, we evaluated our visualization recommendation approach.

Overall, these contributions implemented in our system EVLIN, were evaluated both quantitatively using performance measurements and qualitatively with a user study on both synthetic and real data. Quantitative experiments show the feasibility and the efficiency of our proposed solutions for visual and interactive data exploration while qualitative evaluations show a general satisfaction among users when visually exploring data warehouses data using our system EVLIN.

While our contributions evaluated in this chapter focus mainly on using provenance for visual data exploration, we noticed during our research, in particular for our collaborative-filtering recommendation approach, that there is a more general research question of analyzing a set of provenance documents. To better support such analysis, we discuss in the next chapter, a general provenance summary approach that processes diverse types of provenance (including evolution provenance) and it aims at facilitating users' task when exploring visually provenance.

Chapter 7

Visual analytics of provenance summary

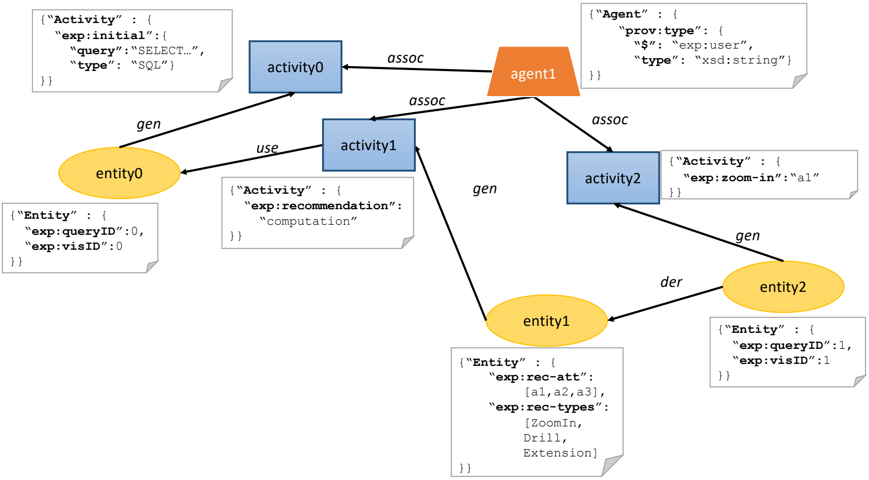
7.1. Introduction

In the previous chapters, we have described our provenance-based framework that provides a holistic approach to support users in exploring data visually. Part of our proposed solution are evolution provenance graphs that represent either individual user exploration sessions or summaries of multiple user sessions. During our research, we noticed the value of analyzing these graphs, especially the summary graph. One very simple example is the discussion in Section 5.4, where analyzing the topology of evolution provenance summary graph helped us to gain further insights on global trends commonly opted by users when exploring visually data. To better support such analysis, we present in this chapter a general approach meant to summarize various types of provenance conforming the standard exchangeable format for provenance PROV-JSON[1]. We further describe the analysis tasks that apply to these

[1] https://www.w3.org/Submission/2013/SUBM-prov-json-20130424/

summaries. Note that we use in this chapter the term *provenance trace* to refer to a single provenance graph collected in one run of the computational processes for which provenance is tracked. To clarify our contribution, let us consider the following example.

Example 16 *We performed a user study to examine methodologies and practices adopted when exploring data visually. To do that, we hired ten students and we asked them to explore various data warehouses using EVLIN, the instance of our framework ProvVDE. To better understand the behaviour of users during the user study, we have collected the evolution provenance tracking explorations made by each participant. These evolution provenance graphs are converted to the PROV-JSON, a standard, exchangeable format. Figure 7.1 depicts an excerpt of the evolution provenance (comprising around 100 edges/nodes) tracked from one exploration session made in the user study. This provenance trace conforms to the PROV-JSON format. Essentially, "agent1" corresponds to the user performing this exploration session. We have also in this excerpt of evolution provenance trace three manipulations that are represented in Fig-*



```
{"Activity" : {
  "exp:initial":{
    "query":"SELECT…",
    "type": "SQL"}
}}
```

```
{"Agent" : {
    "prov:type": {
      "$": "exp:user",
      "type": "xsd:string"}
}}
```

```
{"Entity" : {
  "exp:queryID":0,
  "exp:visID":0
}}
```

```
{"Activity" : {
  "exp:recommendation":
    "computation"
}}
```

```
{"Activity" : {
  "exp:zoom-in":"a1"
}}
```

```
{"Entity" : {
  "exp:queryID":1,
  "exp:visID":1
}}
```

```
{"Entity" : {
    "exp:rec-att":
         [a1,a2,a3],
    "exp:rec-types":
         [ZoomIn,
         Drill,
         Extension]
}}
```

Figure 7.1.: Excerpt of evolution provenance collected in EVLIN

ure 7.1 as three activities: "activity0", "activity1", and "activity2". "activity0" corresponds to the initial SQL query specified by the user in EVLIN to start an exploration session. "activity1" refers to the computation of recommendations in the course of the exploration while "activity2" corresponds to the user's request to investigate the Zoom-In recommended query (cf. Example 13) associated to attribute $a1$*. Those activities are associated with the user (see edges with labels "assoc"). These activities are involved in the generation of three entities: "entity0", "entity1", and "entity2". The two entities "entity0", and "entity1" correspond to the exploration steps while the entity "entity2" refers to an impact matrix containing scored recommendations (cf. Example 7).*

As mentioned in the example, we aim at studying users' behavior when exploring data using EVLIN. Hence, it would be interesting to find out how evolution provenance traces of participants relate to each other. Yet, a simple evolution provenance graph easily reaches a large size. Consequently, comparing all evolution provenance traces quickly becomes infeasible.

To simplify the comparative task motivated above, we propose in this chapter our structural-based summary approach. Our proposed approach infers initially the general structure of each provenance trace. The individual structures are then unified in a merged structure with annotations that quantify the coverages of sub-structures in the underlying collection of provenance traces. In what follows, a particular attention is given to the explanation of our proposed summary approach.

Example 17 *The structural summary of the ten evolution provenance traces collected in the course of our user study is depicted in Figure 7.2. Vertices present different structures available in the ten provenance traces. We distinguish mainly three families of structures: (i) entities structures: they are depicted as yellow circles in Figure 7.2 (ii) activities structures: they are depicted as blue boxes in Figure 7.2 and they present structures inferred from activities present in Figure 7.1 and (iii) agents structures: they are depicted in orange trapezoid in Figure 7.2 and they are inferred from agents present in Figure 7.1. Each structure definition is depicted in a white box near each vertex. Each edge is* *associated with two annotations that specify the type of provenance relationship and its frequency in the analyzed provenance traces. From this graph, we see that the activity structure “actST5” was the most present among available structures of activity. Recall that the structure “actST5” corresponds to the situation where the user is asking for recommendation to explore interesting regions of the data. This indicates that users rely mainly on recommendations suggested alongside the visual data exploration process. Based on the edge linking the structure “actST5” and the structure “entSt1”, we see that 90 recommendation sets were proposed to the ten participants in the user study. Among the 90 set of recommendations, we see that users prefer mostly to investigate recommendations of type “ Zoom-In” (see the cardinality of edge linking “actSt1” and “entST0”). Contrarily, a low number of recommendations of type extension were investigated by the participants in the user study (the cardinality of the edge between “actSt4” and “entST0” is equal to 25).*

From this summary, we can learn some common practices adopted by users when exploring data visually using EVLIN. For instance, it becomes clear that we can distinguish types of recommendations commonly opted (e.g. recommen-

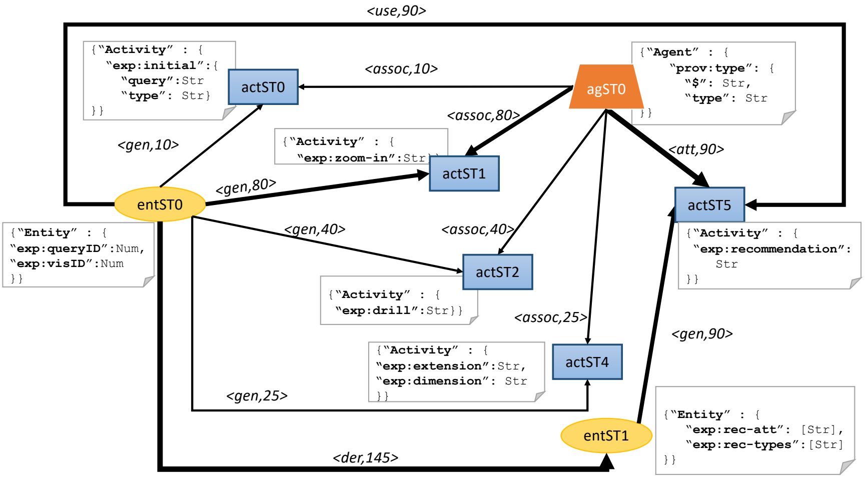


Figure 7.2.: Structural summary of ten evolution provenance traces

dations of type "Zoom-In") from those less solicited by users (e.g., recommendations of type "Extension") when exploring data using EVLIN.

The example above provides one motivating scenario that demonstrates the usefulness of the structural provenance summary approach. Further use cases are discussed in Section 7.6. To achieve the functionality described in the example, we propose an end-to-end solution that infers the schema of multiple provenance traces in W3C-PROV format to then summarize those in one graph. This approach has several benefits including (i) the summary allows to get a first rough overview about a collection of provenance traces; (ii) it is useful to highlight commonalities and differences among the traces for comparative analysis, and (iii) it allows to pinpoint corner cases and exceptions.

Overall, our proposed approach ensures the following contributions.

- **Provenance structure summarization**. We propose a two-phase algorithm to produce the structural summary of provenance traces. The first step infers the structure of each individual trace. These structures are then input to the second phase that produces the merged and weighted provenance structure summary graph, where weights translate the coverage of a relationship in the individual traces.
- **Provenance summary visualization and analysis.** We define several visual analysis tasks over the summary graph. For selected tasks, we showcase different visualizations as part of our preliminary use-case based evaluation.
- **System implementation and evaluation.** We implement the proposed approach in a prototype, for which we provide a preliminary performance evaluation. Results show the effectiveness of our approach in terms of runtime and conciseness of inferred summaries.

The remainder of this section is structured as follows. Section 7.2 discusses related work. Section 7.3 covers necessary definitions and formal preliminaries. We present our structure-based summary approach in Section 7.4. Visual analysis tasks and use cases are presented in Section 7.5 and Section 7.6. Finally, we discuss our implementation and evaluation in Section 7.7. Note that the content of this section is mainly based on methods and approaches described in [BH19a], which are based on our early work [BBC+17].

7.2. Related work

This section reviews research in fields related to our work, i.e., provenance summary and schema inference and integration.

7.2.1. Summary of provenance traces

Several strategies have been proposed to simplify a single provenance trace. One of them is the temporal-based strategy which was widely adopted by many works including [BYB+13; SLSG16]. This strategy consists of clustering provenance information that was tracked in the same time frame. While this strategy reduces successfully the complexity when dealing with one provenance trace, it does not apply on a set of provenance traces collected at different periods.

Other strategies include semantic summaries [ABD+15; KFS13; OMO+16]. These require expert knowledge to define semantic mappings between provenance components. The same holds for the user-defined summaries [BBD07; MBG+14], where sufficient user knowledge about the processed provenance trace is required to appropriately perform grouping operations.

Template-based summarization consists in merging sub-parts of an analyzed provenance trace when they have the same shape (template) [SLSG16]. However, the specification of these templates is left to users, i.e., they have to specify patterns they expect in their provenance traces that can be collapsed in a simplified visualization. In the same context, Moreau et al. [Mor15] propose a parametric summary solution that compresses firstly paths of size *k* and then merges compressed paths having the same shapes. While this solution could be applied on a set of provenance traces, it is still unclear how to set a reasonable value of k that generates a concise summary neither too general nor too specific.

Finally, approaches such as [CFDC17; MBH+18] declare a template subsequently collected provenance follows. This template can be seen as a static summary that is independent of the set of analyzed provenance traces. Opposed to generating a static summary, our approach considers actual provenance traces and exposes their similarities and differences. Our approach reveals also which parts of the static summary are actually covered by the set of analyzed provenance traces.

7.2.2. Schema inference and schema integration

Our approach is close in spirit to schema inference, where given a set of datasets, a common, generalized schema in a predefined data model is derived. Given our focus on semi-structured W3C PROV provenance traces, our work is closely related to schema inference for XML and JSON data [BBC+17; HNW06].

Our work is also related to schema integration, especially for semi-structured data [CKP08] where, given a set of schemas, a unified schema is determined. This latter covers all concepts and properties of the input schemas and correctly models the constraints defined in these input schemas.

While the above methods may complement our approach, they are left for future research. In particular, integrating such techniques requires to further investigate the information loss and associated impact on analytical applications on provenance traces. Our current focus lies on inferring a structural summary that reports primitive types of provenance components and that highlights dependencies (an important aspect for provenance analysis) between inferred structures.

7.3. Preliminaries and system overview

As discussed earlier, we propose a solution relying on structural aggregation for visual provenance analysis. This solution follows the pipeline shown in

Figure 7.3, briefly described below.

Individual W3C-PROV graphs. We assume that all input provenance traces conform to the W3C-PROV data model [BBC+12]. As defined in [MBG+14], this data model defines three types of sets: (i) Entities (*En*), i.e., data, documents; (ii) Activities (*Act*), i.e., processes, actions acting upon entities; and (iii) Agents (*Ag*), i.e., humans, software. The W3C-PROV data model further defines a set of relationships among entities, activities, and agents. Examples of relationships include for instance:

$usage : use \subseteq Act \times En \qquad attribution : att \subseteq En \times Ag$
$derivation : der \subseteq En \times En \qquad generation : gen \subseteq En \times Act$
$delegation : del \subseteq Ag \times Ag \qquad association : assoc \subseteq Ag \times Act$

Note that the W3C-PROV data model has many variations. In what follows, we focus mainly on the PROV-JSON representation of the W3C-PROV given the wide adoption of JSON as an exchangeable data format. This allows us to reuse existing solutions, e.g., for JSON schema inference [BBC+17] or visualization frameworks. Furthermore, JSON is semi-structured and thus allows to easily model heterogeneous provenance traces.

A PROV-JSON provenance trace follows the tree format displayed in Figure 7.4. Each information present at the top-key level maps to a W3C-PROV concept described above.

The provenance traces represented in PROV-JSON format generally form

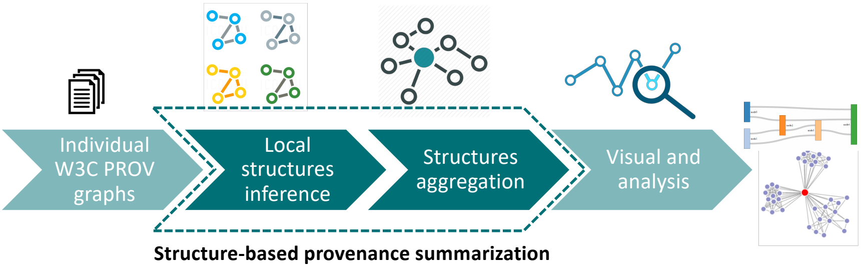

Individual W3C PROV graphs
Local structures inference
Structures aggregation
Visual and analysis
Structure-based provenance summarization

Figure 7.3.: Overview of the proposed approach

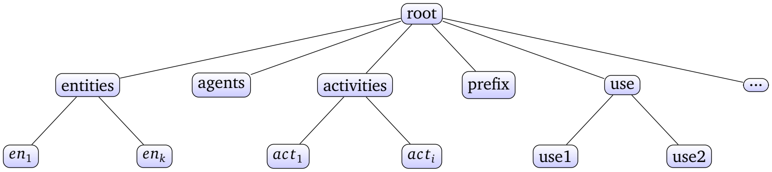


Figure 7.4.: Top-key level of a PROV-JSON trace

a graph, that we define as follows.

Definition 7.1 (Provenance graph) *A provenance graph is a directed graph $G_P(V,R)$, where V is the set of vertices and R maps to provenance relationships. Let $root \subseteq V$ be a set of vertices with an in-degree of 0 and an out-degree larger than 0. These nodes represent entities that are derived as described by the provenance trace. Successors of $root$ are entities $En \subseteq V$, activities $Act \subseteq V$, and agents $Ag \subseteq V$ involved in generating the entities described by $root$. Each edge $e = (\langle v_1, v_2 \rangle, t)$, represents a provenance relation $r \subseteq R$ of type t (e.g. gen, use) between v_1 and v_2 where $\{v_1, v_2\} \subseteq V$.*

Figure 7.1 depicts an example of a provenance graph. It describes the set of explorations steps (cf. Definition 4.1, Page 49) investigated by the user. The set $root$ contains the entity "entity2" (corresponding to the last reached exploration step) and the agent "agent1" (corresponding to the user). Figure 7.1 further includes white boxes associated with each node in the provenance graph. These contain the concrete provenance information recorded for each node. For instance, the provenance recorded about "entity2" includes the ids of the query and the visualization inspected by the users.

Structure-based provenance summarization. The set of provenance graphs defined above serve as input to our structure-based provenance summary approach, which consists of two steps. First, we infer individual provenance structures associated to individual provenance traces (local structures inference in Figure 7.3). The second step aggregates individual structures into a single W3C-PROV compliant graph that structurally summarizes input provenance traces. More formally, we define the structure-based summary graph as follows.

Definition 7.2 (Structure-based summary graph) *A structure-based summary graph is a W3C-PROV compliant directed graph $SSG(PS_s, RS_s)$ associated to a set of provenance graphs $G=\{G_{P1}, \ldots, G_{Pn}\}$ where PS_s is the set of vertices referring to provenance structures defined later in Definition 7.3 and RS_s are edges mapping to the set of inferred provenance relationships. Each relation $r \subseteq RS_s$ is presented as $(\langle ps_1, ps_2\rangle, t, c)$ where $\langle ps_1, ps_2\rangle$ is the edge connecting two structures $ps_1, ps_2 \in PS_s$, the type t presents the provenance relationship type, and the cardinality c is the number of provenance relationships in G, that follow the structure of r.*

Figure 7.2 shows an example of a structure-based summary graph. The content of this graph is discussed in Example 17.

Visualization and analysis. The last component of our pipeline generates visualizations suited for various visual analysis tasks that rely on the structure-based summary graph.

After this overview, we now delve into the details of individual components of our solution in Section 7.4 and Section 7.5.

7.4. Provenance structure inference

Figure 7.4 shows the top-key level structure of a PROV-JSON provenance trace, which contains nodes referring to W3C-PROV relationships (e.g., use,

$p ::= Act \mid En \mid Ag$	provenance high level types
$Act, En, Ag ::= \epsilon \mid R$	provenance types
$R ::= \{l_1 : s_1, \ldots, l_i : s_i\}$	Record type
$s ::= R \mid A \mid Bs$	structure types
$A ::= [s_1, \ldots, s_j]$	Array type
$B ::= null \mid Bool \mid \ Num \ \mid \ Str$	Basic type

Figure 7.5.: Syntax of provenance types

gen). Those nodes have always standard structures respecting the constraints defined in the W3C-PROV data model. In contrast, nodes mapping to entities, activities, and agents may have various structures (e.g., entities "entity0" and "entity1" in Figure 7.1) depending on the underlying provenance collector. Hence, a special focus is given in what follows to the subset of top-key level's elements, called provenance components, that contains entities, activities, and agents. The provenance components follow the syntax shown in Figure 7.5, extending the syntax proposed in [BBC+17] to fit the content of W3C-JSON provenance traces.

A provenance component p is either an activity (Act), entity (En) or agent (Ag) type. These provenance high level types encompass records that are sets of pairs. Each pair has a label l_i associated with a structure type s. This latter could be a record, an array, or a basic type. The array type is a sequence of structure types s while basic values B comprise $null$ values, booleans $Bool$, numbers Num, and strings Str.

7.4.1. Provenance component type inference

The first phase of our approach performs a type inference for each provenance component p present in the provenance trace. We adjust the inference rules proposed in [BBC+17] to infer types called in what follows *prov-types* defined in Figure 7.5.

Prov-type inference is done according to the inference rules in Figure 7.6. We distinguish two types of inference rules: (i) those without premise: they infer the prov-type of value by simply reflecting the type of the value itself and (ii) rules with premise: they require the recursive prov-type inference of each element present in the premise to generate the global prov-type indicated in the conclusion part.

Based on inference rules presented in Figure 7.6, we can define *prov-type*, the first step towards generating a structural provenance summary of a collection of provenance traces.

Definition 7.3 (Prov-type) *A prov-type for a vertex v present in $G_P(V,R)$ corresponds to the structure inferred for v.*

(TypeEmpty)

$$\frac{}{\vdash \{\} \rightsquigarrow \mathtt{Null}}$$

(TypeBool)

$$\frac{}{\vdash true/false \rightsquigarrow \mathtt{Bool}}$$

(TypeNumber)

$$\frac{s \in Number}{\vdash s \rightsquigarrow \mathtt{Num}}$$

(TypeString)

$$\frac{s \in String}{\vdash s \rightsquigarrow \mathtt{Str}}$$

(TypeArray)

$$\frac{\vdash s_1,\ldots s_n \rightsquigarrow T_1,\ldots,T_n}{\vdash [s_1,\ldots,s_n] \rightsquigarrow [T_1,\ldots,T_n]}$$

(TypeRec)

$$\frac{\vdash s_i \rightsquigarrow T_i \quad \vdash s_j \rightsquigarrow T_j \quad \forall\, i,j \; i \neq j \Rightarrow l_i \neq l_j}{\vdash \{l_1{:}s_i,\ldots,l_i{:}s_i\} \rightsquigarrow \{l_1{:}T_i,\ldots,l_i{:}T_i\}}$$

(TypeAgent)

$$\frac{\vdash s \rightsquigarrow T}{\vdash Ag(s) \rightsquigarrow Agent(T)}$$

(TypeAct)

$$\frac{\vdash s \rightsquigarrow T}{\vdash Act(s) \rightsquigarrow Activity(T)}$$

(TypeEntity)

$$\frac{\vdash s \rightsquigarrow T}{\vdash En(s) \rightsquigarrow Entity(T)}$$

Figure 7.6.: Prov-type inference rules.

Example 18 (Example of inferred prov-types) *Following our definition, the prov-type inferred for the entity “entity0” shown in Figure 7.1 is:*

```
{ “Entity”:
     {
          “exp:queryID”: Num,
          “exp:visID”: Num
     }
}
```

This prov-type is inferred using the recursive TypeRec *inference rule depicted in Figure 7.6. This rule is recursive. Accordingly, we iterate over key-values pairs present in entity “entity0” shown in Figure 7.1. For each key-value pair, we maintain the key and we apply a basic inference rule on the value (see* TypeNumber *rule depicted in Figure 7.6). Finally, the set of key and their associated basic types are grouped and returned as a result of the recursive* TypeRec *inference rule.*

7.4.2. Individual structural provenance graph

Once we compute the prov-type of each vertex in a provenance graph, we are able to infer the individual structural provenance graph that we define as follows.

Definition 7.4 (Individual structural provenance graph) *$GS(P_s, R_s)$ is an individual structural provenance graph associated to a provenance graph $G_P(V, R)$. It is a W3C-PROV compliant directed graph where P_s is the set of prov-types for all $v \in V$ and R_s is the set of inferred relationships structures: Each edge $e_s = (\langle ps_1, ps_2 \rangle, t, c)$ is a provenance relationship structure in R_s between two structures $\{ps_1, ps_2\} \subseteq P_s$. The edge e_s is labeled by t that refers to the type of provenance relationship and by a cardinality c computed as $c = |R_{sim}|$ such that $R_{sim} \subseteq R$ and $\forall e_r = (\langle v_1, v_2 \rangle, t') \in R_{sim}$, $e_s.t = e_r.t'$ holds and $\forall i \in \{1,2\}, \exists j \in \{1,2\}$ such that the prov-type of v_i is equal to ps_j.*

Note that the following properties hold for the individual structural provenance graph $GS(P_s, R_s)$:

Lemma 3 (Properties of structural provenance graph) *For a structural provenance graph $GS(P_s, R_s)$, we have*

- *$\forall ps, ps' \in P_s$, $ps \neq ps'$*
- *$\forall p_s \in P_s$, $\exists\ v \in V$ such that its prov-type is equal to p_s*
- *$\forall e_s = (\langle ps_1, ps_2 \rangle, t, c) \in R_s, \exists e = (\langle v_1, v_2 \rangle, t') \in R$ such that $e_s.t = e.t'$ and $\forall i \in \{1,2\}, \exists j \in \{1,2\}$ such that the prov-type of v_i is equal to ps_j.*

Figure 7.7 shows an example of individual structural provenance graph associated to the excerpt of W3C-PROV evolution provenance graph shown in Figure 7.1. It contains 6 nodes whose prov-types are shown in white boxes below each vertex. For instance, the node “actSt0” has a prov-type that was inferred from the node “Activity0” present in Figure 7.1. Figure 7.7 depicts also cardinalities and types associated with each edge. For instance, the edge between “actSt1” and “agSt0” has the type “association” and a cardinality equals to one. The cardinality is explained by the provenance graph shown in Figure 7.1 where we have one edge of type “association” between nodes “activity1” and “agent1”.

To infer an individual structural provenance graph, we leverage the following two fundamental definitions.

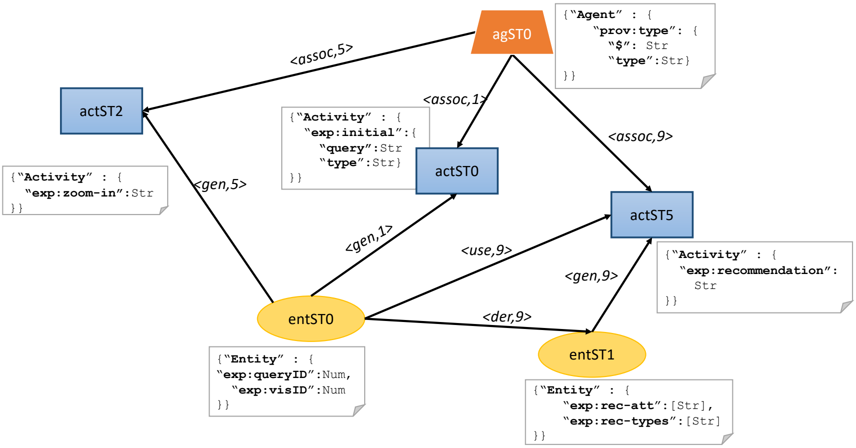

agST0
{"Agent" : { "prov:type": { "$": Str "type":Str} }}
<assoc,5>
actST2
<assoc,1>
{"Activity" : { "exp:initial":{ "query":Str "type":Str} }}
actST0
<assoc,9>
{"Activity" : { "exp:zoom-in":Str }}
<gen,5>
actST5
<gen,1>
<use,9>
{"Activity" : { "exp:recommendation": Str }}
<gen,9>
entST0
<der,9>
entST1
{"Entity" : { "exp:queryID":Num, "exp:visID":Num }}
{"Entity" : { "exp:rec-att":[Str], "exp:rec-types":[Str] }}

Figure 7.7.: Structural summary of MIT's ranking provenance

Definition 7.5 (Vertex structural equality) *Given a provenance graph $G_P(V,R)$, we say that two vertices are structurally equal if they have the same prov-type.*

Definition 7.6 (Edge structural equality) *We consider that edges $(\langle el1_1, el1_2\rangle, t_1)$ and $(\langle el2_1, el2_2\rangle, t_2)$ are structurally equal if they have the same provenance relationship type ($t_1 = t_2$), vertices $el1_1$ and $el2_1$ are structurally equal, and vertices $el1_2$ and $el2_2$ are structurally equal.*

Algorithm 9 leverages the two previous definitions to infer an individual structural provenance graph for a provenance graph G_p. Firstly, we go over the provenance components (lines 4–5) and we generate their associated prov-types using the function $InferType$ that implements the rules specified in Figure 7.6. Pairs of inferred structures and ids of provenance components are then stored in the map s_{cand}. In line 6, we aggregate similar structures present in s_{cand} based on Definition 7.5 and we store results in ST_{inf} that contains inferred prov-type s and the full ids list of provenance components that follow s.

Afterwards, we go over provenance relationships (lines 7– 12). Recall that

Algorithm 9: Individual Prov Structure Inf (G_p)

```
   Input: $G_p$ : provenance graph
   Output: $GS$: individual structural provenance graph
1  $s_{cand} \leftarrow$ an empty map of $\langle S, id \rangle$ elements, where $S$ is the prov-type,
     and $id$ is an identifier of a provenance component;
2  $ST_{inf} \leftarrow$ an empty hash-map of $\langle S, L(id) \rangle$ elements, where $S$ is the
     prov-type, and $L(id)$ is the list of provenance components ids
     following $S$;
3  $REL_{Inf} \leftarrow$ an initially empty list of $\langle \{d_1..d_k\}, t, c \rangle$ elements, where $t$
     refers to the type of provenance relation, $\{d_1..d_k\}$ are related
     provenance components and $c$ refers to the cardinality ;
   /* Inference of provenance components structures    */
4  foreach $e \in G_p.getVertices()$ do
5      $S_{cand} \leftarrow Add(\langle InferType(e), e.getId() \rangle)$;
   /* Aggregation of prov-types                          */
6  $ST_{inf} \leftarrow AggregateTypes(S_{cand})$;
   /* Inference of provenance relations structures     */
7  foreach $r \in G_p.getRelations()$ do
8      $r_{rel} \leftarrow r.getRelationsElements()$;
9      $s_{rel} \leftarrow \emptyset$ ;
10     foreach $comp \in r_{rel}$ do
11         $s_{rel} \leftarrow Add(ST_{Inf}.get(comp.getID()))$;
12     $REL_{Inf} \leftarrow Add(s_{rel}, r.getType, 1)$;
   /* Provenance relations structures aggregation      */
13 $REL_{Inf} \leftarrow AggregateRels(REL_{inf})$;
14 $GS \leftarrow$ new Graph ($ST_{inf}$,$REL_{inf}$);
15 return $GS$;
```

a provenance relationship specified in Definition 7.1 is an edge between two provenance components. Hence, we identify for each relationship r, ids of provenance components involved in r. We resort to the ST_{inf} to replace the set of ids by their corresponding prov-types. Later, we store identified prov-types, the type of r, and a cardinality (set to one) in REL_{inf} (line 12). Finally, we aggregate in line 13 the cardinality of relationships that are structurally equal using function $AggregateRels$ that implements Definition 7.6.

Once we specify the set of prov-types ST_{inf} and the set of inferred relationships REL_{inf}, our Algorithm 9 returns GS, the individual structural provenance graph of G_p.

7.4.3. Structure-based summary graph generation

At this stage, our goal is to further aggregate the set of individual structural provenance graphs returned by the previous step in order to generate a unique structure-based summary. We choose to apply an exact merge to fuse input structural summary graphs. For that, we leverage again Definition 7.5 and Definition 7.6 to merge vertices and edges of individual structural graphs that are structurally equal.

Our choice of exact merge is justified by its capability to mitigate the loss of information by providing exact structures available in a set of provenance traces. This is not the case for the inexact merge that fuses nodes having different yet compatible structures. While our approach could be extended to support the inexact merge, we prefer to leave it for future research to assess or convey the impact of inexact merge techniques on visual analytics tasks. Based on the current implementation of merge, we can redefine the structure-based summary graph as follows.

Lemma 4 (Structure-based summary graph properties) *The structure-based summary graph* $SSG(PS_s,RS_s) = \bigcup_{i=1}^{n} GS_i(P_{si},R_{si})$ *where* GS_i *are individual structural provenance graphs. It has a surjective map* M*:* $\bigcup_{i=1}^{n} P_{si} \longrightarrow PS_s$ *such that:*

- $\forall p \in P_{si}, \exists p^* \in PS_s$ *with* p *and* p^* *are structurally equal*
- $\forall r \in R_{si}, \exists\ r^* \in RS_s$ *with* r *and* r^* *are structurally equal*

While it is possible to incrementally perform a set of pairwise merges of individual structural provenance graphs to construct the structure-based summary graph, we choose to design the structure-based summary inference using a map-reduce model to ensure the capability of processing large collections of provenance traces. We justify our choice of this model by the soundness property already proved for the structure inference approach [BBC+17].

Our approach enjoys also commutativity and associativity properties as we use an exact matching strategy to aggregate structures. These are important properties since the map-reduce model may arbitrarily split the input set of provenance traces. We postpone the discussion of the performance study of our approach to the Section 7.6.

7.5. Visual analysis of summary graphs

Based on the structure-based summary graph for a collection of provenance traces obtained as described in the previous section, we define several visual analysis tasks that apply to this kind of summary. For each task, we also evoke visualization techniques/interactions that potentially fit the task.

Ⓐ *High level overview* The primary goal of our approach is to generate a structural summary that is easy to read and that allows analysts to easily grasp which different structures are present in the provenance traces. To avoid overwhelming analysts with too many details at an early stage of their analysis, the visualization should be limited to the rendering of basic information such as vertices and edges representative of structures and provenance relationships.

Ⓑ *Interactive visual analysis* Rendering a simple visualization of structure-based summary facilitates the analysis task. Further information can be offered on demand using interaction, e.g., hovering over nodes or edges.

Ⓒ *Visual comparison* Possible visual analysis tasks include the comparison between the summary and provenance instances. Here, an analyst may compare a particular provenance trace to the inferred structural summary graph. Brushing and linking interaction techniques could be employed in this task to render/highlight structures and their corresponding provenance traces.

Ⓓ *Homogeneity overview* Our approach assigns cardinalities to edges present in the provenance structural summary graph. This information should be communicated clearly as it serves to investigate the homogeneity/heterogeneity of analyzed provenance traces. Sankey diagrams [RHF05] are a candidate visualization for this task, given their ability to render edges with various width expressing the importance of cardinalities.

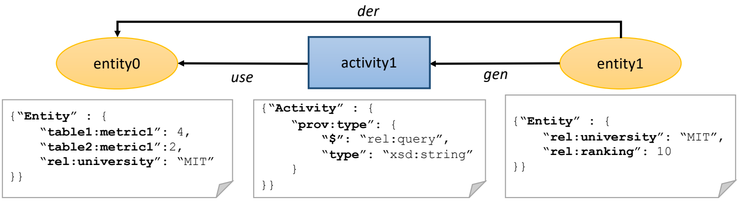
Ⓔ *Visual identification of patterns* Using cardinality information, we can reveal recurrent patterns among the set of analyzed provenance traces. The visualization should highlight sub-graphs having high cardinalities compared to other information present in the summary graph. Sankey diagram [RHF05] could be used also for this analysis task.

Ⓕ *Visualization of dense regions of the summary* Structure-based summary graphs may contain dense regions where vertices are highly connected. This specific range of nodes may be subject of bottleneck or may present a heavily shared component. Note that the presence of high connectivity can easily lead to a cluttered visualization. To avoid that, we can use force-directed graphs that reduce edge crossings by making edges repel each other.

The list of proposed visual analysis tasks and their visualizations is not exhaustive. Indeed, a thorough investigation of other possible visual analysis tasks is left for future research.

7.6. Case study

We have shown initially in Examples 16 and 17 that our structure-based summary graph facilitates the task of understanding users' behaviour when exploring visually data using our system EVLIN. In this section, we broaden the discussion to show the feasibility of our instance when exploring visually other types of provenance. Alongside the description of these scenarios, we illustrate the feasibility of some visual exploration tasks discussed in Section 7.5.



```
{"Entity" : {
    "table1:metric1": 4,
    "table2:metric1":2,
    "rel:university": "MIT"
}}
```

```
{"Activity" : {
    "prov:type": {
        "$": "rel:query",
        "type": "xsd:string"
    }
}}
```

```
{"Entity" : {
    "rel:university": "MIT",
    "rel:ranking": 10
}}
```

Figure 7.8.: Provenance for MIT custom rank

7.6.0.1. Visual debugging of a tracked process

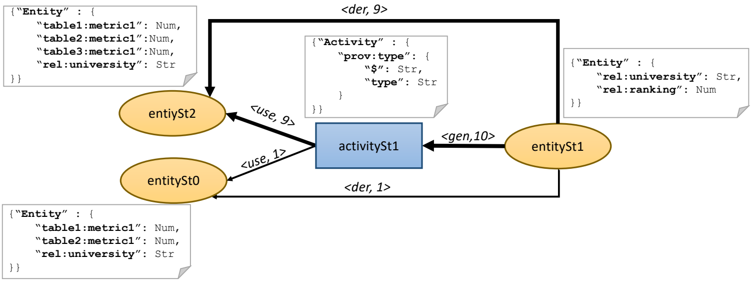
Assume that our goal is to define a university ranking that consolidates three well-known university rankings[1]. As the different source rankings use different criteria, the three source tables do not fully agree. Hence, consolidation requires the definition of a custom score derived from the three sources. Now, let us assume that the custom score yields some surprising results, e.g., the university "MIT" is ranked at tenth position. To better understand the working of the ranking, we compute the why-provenance of the suspicious result, as shown in Figure 7.8. This trace alone is not very insightful yet. To this end, we compute the structure-based summary graph that includes traces for all results (we limit to 10 results in our example).

Our structure-based summary graph, shown in Figure 7.9 summarizes the ten why-provenance traces associated with the top-ten query results.

From this graph, we see (Task Ⓐ) that while most results (all conforming to the structure "entitySt1") are derived from entities conforming to the structure of "entitySt2", one entity is derived from a different structure, i.e., "entitySt0"(Task Ⓓ). Interaction with this summary graph allows us to identify "entitySt0" as the structure of the unexpected "MIT" result. Hence, this summary shows that "activity1" computing the consolidated ranking may be affected by an incomplete input table (i.e., no score from the third source is available).

Indeed, it turns out that the use of a full outer join in our ranking query is unreliable as it tolerates the integration of incomplete information. Specifi-

[1] https://www.kaggle.com/mylesoneill/world-university-rankings



```
{"Entity" : {
    "table1:metric1": Num,
    "table2:metric1":Num,
    "table3:metric1":Num,
    "rel:university": Str
}}
```

```
{"Activity" : {
    "prov:type": {
        "$": Str,
        "type": Str
    }
}}
```

```
{"Entity" : {
    "rel:university": Str,
    "rel:ranking": Num
}}
```

```
{"Entity" : {
    "table1:metric1": Num,
    "table2:metric1": Num,
    "rel:university": Str
}}
```

Figure 7.9.: Structural summary of ten provenance traces

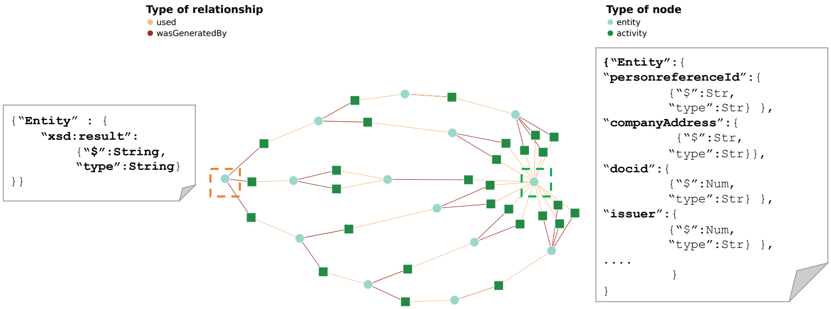
cally, MIT university has a different name in one of the three ranking tables. This entails the presence of a tuple resulting from joining two ranking tables, used later to compute the final score of MIT university.

7.6.0.2. Data integration flow transparency

Our second use case considers a data integration process specified using the high-level integration language (HIL) [HKK+13] incorporated in IBM Infosphere Master Data Management[1]. The sample integration flow extracts information about key people of the US financial sector. It takes a set of reports generated by companies to construct a set of individual reports about persons' careers. HIL was recently instrumented to capture provenance [OH18], allowing us to generate five provenance traces tracking the discussed flow when integrating information about five persons. These traces are converted subsequently to PROV-JSON format.

Assume now that we are interested in understanding and learning recurrent patterns in the discussed sample integration flow. Accordingly, we can compare visually the five generated provenance graphs. Yet, this process is tedious given the wealth of content of processed provenance traces. To this end, we resort to our structure-based summary approach that can facilitate the task of understanding the tracked data integration process.

[1] https://www.ibm.com/support/knowledgecenter/SSWSR9_11.4.0/com.ibm.swg.im.mdmhs.pmebi.doc/topics/using_hil.html



```
{"Entity" : {
    "xsd:result":
        {"$":String,
        "type":String}
}}
```

```
{"Entity":{
"personreferenceId":{
          {"$":Str,
          "type":Str} },
"companyAddress":{
           {"$":Str,
          "type":Str}},
"docid":{
          {"$":Num,
          "type":Str} },
"issuer":{
          {"$":Num,
          "type":Str} },
....
           }
}
```

Figure 7.10.: Structural summary for HIL provenance graphs

As shown in Figure 7.10, we render the summary output by our approach using a force-directed graph (Task Ⓐ) given its capacity to place in convenient way vertices and edges by assigning forces to them.

The inspection of Figure 7.10 reveals two important pieces of information. Firstly, we observe that the structure highlighted using an orange dashed box corresponds to the outcome of the implemented data integration flow as it is the only structure having only relationships of type "generation" with three activities structures. This reveals also that this particular structure was populated by three integration sub-flows (Task Ⓔ). By tracing back interactively these sub-flows, we can learn more about the implemented data integration flow.

Furthermore, Figure 7.10 shows a second important finding. Indeed, all data integration sub-processes stem from the single structure highlighted by a green dashed box (Task Ⓕ). This later presents the structure of input reports used by the tracked data integration process. By hovering over the interactive visualization of Figure 7.10, we can learn the structure of input reports. For instance, the input reports contain personal information about the key people including their address, their positions, as well as information related to the reports such as the issuer (the editor) of a report and the ID of the issuer.

pattern	clause
claim	wasGeneratedBy(e_1, a_1) with e_1 ={"foaf:name":{"$":"file.txt", "type":"string"},"prov:type":{"$": "kimlab","type":"qualified_name" }}, a_1 : {"kimlab:htseq-stranded": {"$": "gencode2","type":"string"}}
confirmation pattern	wasGeneratedBy(e_1, a_1) with e_1 = {"foaf:name": {"$": *,"type":*}, "prov:type":{"$": *,"type":* }}, a_1 = {"kimlab:htseq-stranded":{"$": *,"type":*}}
witness pattern	wasGeneratedBy(e_1, a_1) with $e_1 = \{*\}, a_1 = \{*\}$

Table 7.1.: Clauses used in the corroboration process

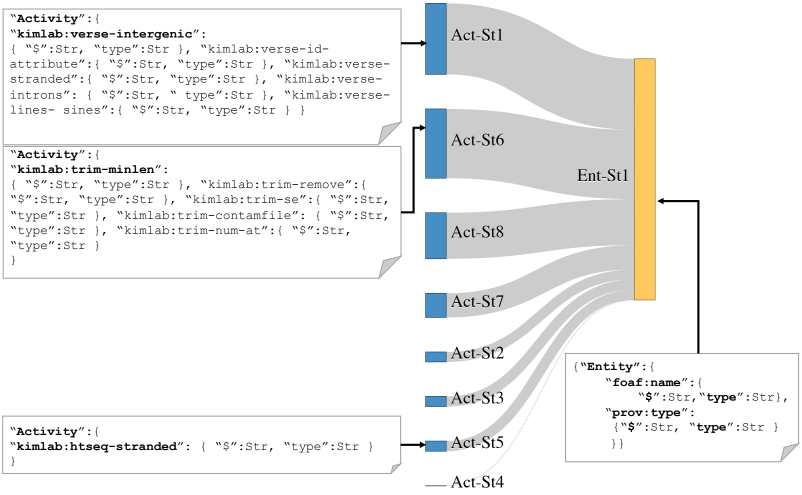
7.6.0.3. Corroboration

Our final use case comes from the life science domain relating to Next Generation Sequencing (NGS) and is inspired by [ACD+18]. This project presents a workflow including six possible analysis stages that are invoked differently depending on the version of the workflow. One major problem already mentioned in [ACD+18] is the lack of NGS workflow transparency. Tackling this problem resulted in the collection of more than 800 provenance traces, which are publicly available[1].

We assume that an analyst is working on this collection to study impacts of analysis stages on the generated results. Given the large size of this collection, the analyst randomly picks some provenance traces. The analysis of these traces reveals the presence of common information presented in the first row of Table 7.1. This clause states that an analysis stage called “HTSeq” presented as an activity a_1 is always involved in the generation of some intermediate results. As it is tedious to check all provenance traces, *the analyst claims that data input to the tracked workflow is necessarily processed by the analysis stage “HTSeq”*.

To assess the truthfulness of the analyst’s claim, we resort to the approach proposed in [BTGM17]. It extracts the set of confirmation patterns (witnesses confirming the structure of the claim) and the set of witness patterns (generic version of the claim) from the existing provenance traces. The second and

[1] https://github.com/alawinia/provClustering/



```
“Activity”:{
“kimlab:verse-intergenic”:
{ “$”:Str, “type”:Str }, “kimlab:verse-id-
attribute”:{ “$”:Str, “type”:Str }, “kimlab:verse-
stranded”:{ “$”:Str, “type”:Str }, “kimlab:verse-
introns”: { “$”:Str, “ type”:Str }, “kimlab:verse-
lines- sines”:{ “$”:Str, “type”:Str } }
```

```
“Activity”:{
“kimlab:trim-minlen”:
{ “$”:Str, “type”:Str }, “kimlab:trim-remove”:{
“$”:Str, “type”:Str }, “kimlab:trim-se”:{ “$”:Str,
“type”:Str }, “kimlab:trim-contamfile”: { “$”:Str,
“type”:Str }, “kimlab:trim-num-at”:{ “$”:Str,
“type”:Str }
}
```

```
“Activity”:{
“kimlab:htseq-stranded”: { “$”:Str, “type”:Str }
}
```

```
{“Entity”:{
    “foaf:name”:{
        “$”:Str,“type”:Str},
    “prov:type”:
     {“$”:Str, “type”:Str }
     }}
```

Figure 7.11.: Excerpt of structural provenance summary graph

third rows of Table 7.1 describe these two sets, which are finally compared to compute the reliability of the claim.

Note that clauses of confirmation and witness patterns can be easily inferred using our approach. Hence, we use our approach to summarize provenance traces available in this provenance repository.

Figure 7.11 depicts an excerpt of the structure-based summary graph. We choose to only render the set of structural relationships of type “generation” since they map to the witnesses set specified in the third row of Table 7.1 (Task Ⓔ). For that, we use a Sankey diagram as we need to highlight the cardinality of inferred relationships’ structures that will be used to estimate the reliability of the claim. Indeed, the width of edges in the Sankey diagram corresponds to the cardinality value of inferred relationships. For instance, we can easily find the set of confirmation patterns (confirming the second row of Table 7.1) which corresponds to the edge between nodes “ActSt5” and “EntSt1”. The cardinality of this relationship is not high given the mediocre width of this particular edge. Based on this visualization, we can get a rough idea about the reliability of the claim (Task Ⓓ) which seems not high as the number of confirming patterns is significantly less than the number of witnesses (sum of all edges). We can also learn interactively structures and exact cardinalities, that are used to compute the exact value of the reliability of the claim following formulas proposed in [BTGM17].

7.7. Evaluation of the structure-based provenance summary approach

While the previous section focused on presenting the use cases that illustrate the usefulness of our structure-based provenance summary, this section evaluates our approach qualitatively. More specifically, we evaluate in this section the performance of our structure-based provenance summary in terms of conciseness and runtime. To this end, we use in this experiment provenance traces of use cases $UC1$, $UC2$, and $UC3$, referring to the three case studies discussed in Section 7.6. We implement also two types of provenance generators: (i) the first generator gen_{prov} generates many provenance traces according to the structure of a real provenance traces, (ii) $gen_{struct_{prov}}$: generates provenance traces having new structures derived from the structures available in the seed provenance traces. It introduces random changes to the structure of each seed. The new structures are used to generated synthetic provenance traces.

We have studied firstly the conciseness of summary graphs output by our structure-based provenance summary approach. This metric is computed as the ratio of the number of inferred structures (of type entity, activity, and relations) to the total number of entities, activities, and relations in the input provenance traces set.As mentioned previously, we considered provenance traces of use cases $UC1$, $UC2$, and $UC3$, referring to the three case studies

Use case	avg(#activities)	avg(#entities)	avg(#relations)
$UC1$	1	11.4	21.8
$UC2$	114.2	115.3	228.6
$UC3$	5	6	23

Table 7.2.: Overview of processed provenance traces

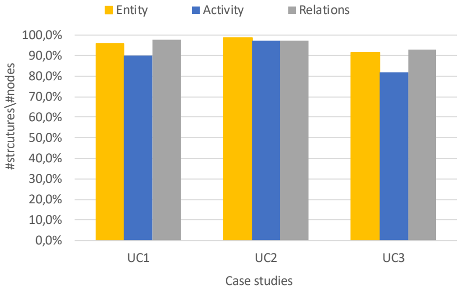


Figure 7.12.: Conciseness rate for diverse case studies

discussed in Section 7.6. For each use case, we collect ten provenance traces whose graphs information are summarized in Table 7.2. We generate for each provenance set ($UC1$, $UC2$, and $UC3$), its structure-based summary graph. Accordingly, we compute the conciseness rate by comparing the number of inferred structures (of type entity, activity, and relations) with the number of entities, activities, and relations in the input provenance traces set. As shown in Figure 7.12, our approach performs a high simplification rate above 80%) for the three studied use cases. This indicates that structure-based summary graphs allow producing concise summaries of collections of provenance traces.

So far, we have studied the conciseness of our structures-based summary graph when processing provenance sets that are highly homogenous (i.e., processed provenance traces in each set have roughly the same structure). Yet, this conciseness rate may be impacted by the degree of heterogeneity of processed set of provenance traces. To this end, we study in the following experiment the impact of heterogeneity degree (i.e, the inconsistency rate of structures of processed provenance traces) on the conciseness of inferred provenance summaries. To do that, we use first our provenance generator $gen_{struct_{prov}}$ (described above) to prepare four provenance traces set containing consecutively 10, 25, 50, and 100 provenance traces with different structures. Later, these provenance sets are increased using our provenance generator gen_{prov} (described above) until they reach the size of

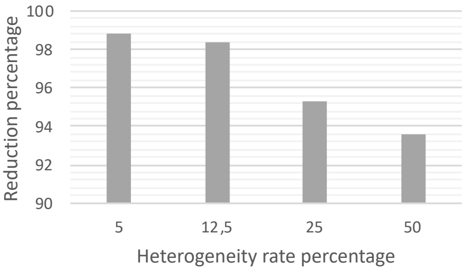


Figure 7.13.: Impact of input provenance heterogeneity on the structural provenance summary conciseness

200 provenance traces. Ultimately, we have four provenance traces sets, each containing 200 provenance traces but with various degree of heterogeneity (5%, 12.5%, 25% and 50%). Note that the degree of heterogeneity is computed as the set of different structures available in a provenance set divided by the size of the provenance set. For instance, to get a heterogeneity degree equal to 5%, our generator $gen_{struct_{prov}}$ outputs 10 provenance traces with 10 different structures. These structures are used by $gen_{prov}$to generate 200 provenance traces.

At this stage, we employ our structures-based summary approach to process provenance traces. Later, we measure the conciseness of our approach by comparing the number of inferred structures (of type agent, entity, activity, and relations) with the number of agents, entities, activities, and relations in each input provenance traces set. For each processed set of provenance traces, we compute the reduction rate that corresponds to the comparison between the size of the output summary graph's size with the size of provenance traces present in the processed set ($\frac{size\_summary\_graph}{\sum size\_processed\_provenance\_traces}$).

Figure 7.13 depicts the impact of heterogeneity degree on the conciseness of the inferred structure-summary graph. As expected, the heterogeneity of provenance traces impacts the conciseness. Indeed, the conciseness rate decreases as the heterogeneity between processed provenance traces increases. Yet, we see that our structure-based summary approach maintains

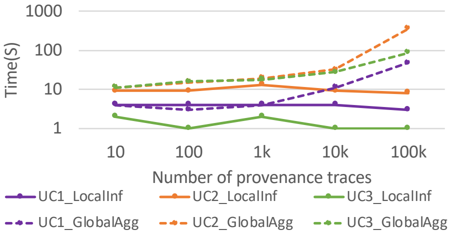


Figure 7.14.: Runtime of summary computation steps

an acceptable rate of conciseness even for highly heterogeneous provenance traces sets (e.g., when heterogeneity rate= %50).

We have also studied the runtime of our approach implemented using a map-reduce model. Accordingly, we generate using gen_{prov} several synthetic provenance traces based on real provenance traces available in $UC1$, $UC2$, and $UC3$. This results in several synthetic provenance trace sets of varying sizes, that are summarized later using our approach. We performed our experiments on a cluster containing 3 nodes, each containing 6 cores and 256GB of RAM. During experiments, we measured runtimes of the two main steps of our approach (shown in Figure 7.3): (i) $LocalInf$ maps to the local inference of structures and (ii) $GlobalAgg$ concerns the aggregation of structures. Figure 7.14 reports the runtimes of the two steps on a logarithmic scale over various sizes of provenance trace collections. The two steps have initially similar runtimes when processing small provenance traces sets. Yet, the gap between the two steps increases clearly with the number of processed provenance traces. Indeed, as the number of provenance traces increases, the size of the hash map storing inferred structures and their associated objects increases. Later, hash map values storing large lists, are accessed during the global aggregation to group edges that are structurally equal. This is costly and leads to an increase of global aggregation runtime.

7.8. Summary and future work

In this chapter, we present our approach that infers a structure-based summary from a set of provenance traces available in PROV-JSON format. We discuss our proposed solution as well as possible visual analytics tasks that may be applied to this type of summary. We show how our approach can contribute to the analysis of a set of evolution provenance graphs generated using our visual data exploration framework ProvVDE. Furthermore, we broaden the discussion of the utility of our approach by providing other illustrative use cases treating other types of provenance. Our preliminary experimental evaluation studies both the runtime and conciseness of summarization. Several points for future research have been mentioned throughout this work, including the integration of schema inference and schema integration techniques to our approach as well as a thorough study and evaluation of different visualizations for different analytical tasks.

Chapter 8

Conclusion and Future Work

Visual data exploration allows users to analyze datasets based on visualizations of interesting data characteristics, to possibly discover interesting information about the data. As the users are a priori unaware of the content of the explored data, the visual data exploration requires an investigative way of navigating through (portions of) the data to discover iteratively valuable information.

Nonetheless, existing visual data exploration tools typically require a tedious process of query and visualization specifications, preventing analysts from efficiently exploring different aspects of the data. As a result, users without enough querying skills or visualization background may fail to accomplish their exploration and overlook important insights or trends hidden in the analyzed data.

To address this challenge, this thesis contributed a set of approaches that support users in the visual data exploration process. Next, we summarize the contributions discussed in this thesis towards improving user 'experience when exploring data visually. Subsequently, we discuss interesting perspectives and open questions for future investigation.

8.1. Review of contributions

In this thesis, we proposed a set of new methods and approaches that aim at improving users' experiences when exploring data visually.

First, we introduced in Chapter 3 our framework ProvVDE meant to explore visually data. Furthermore, we described thoroughly the components present in our visual data exploration framework. Consequently, we introduced EVLIN, an instance of our framework that is meant to explore visually data warehouses.

Later, we proposed in Chapter 4 a new model of evolution provenance that captures prominent information in the course of the visual data exploration process including users' interactions, exploration queries, and visualizations.

After that, we discussed in Chapter 5 the set of provenance-based methods proposed to support users throughout the whole visual data exploration process. More specifically, we discussed a content-based query recommendation approach that returns the set of interesting queries worth inspecting next. To support the visual exploration process, we contributed also a visualization recommendation approach meant to render appropriately investigated recommended queries. Indeed, unlike existing visual data exploration that support users either in writing queries or in visualizing data, we proposed these two provenance-based recommendations approaches that assist users in querying and visualizing data.

Given the high diversity and possibly large number of recommendations produced by our content-based query recommendation approach, we proposed in Chapter 5 a quantification approach that measures the "interestingness" of each recommendation. The measure of recommendations interestingness relies on the deviation metric that compares the dissimilarity of recommendation's data distribution with the data distribution in the whole explored dataset. The computed interestingness scores are visualized as an impact matrix, pointing thereby users to potentially interesting recommendations to investigate next.

Furthermore, we described in the same Chapter 5 our merge approach that aggregates periodically the evolution provenance collected from many previous users' exploration jobs into a multi-user graph. Subsequently, this kind of multi-user graphs is exploited by our second query recommendation approach (described in Chapter 5) that computes collaborative-filtering recommendations. This new type of recommended queries is harnessed to improve the process of quantifying the interestingness of recommendations output by our content-based query recommendation approach. Indeed, we discussed in Chapter 5 how we leverage collaborative-filtering recommendation to diversify interestingness scores to guide better users to interesting portions of the studied data sets.

All aforementioned approaches underwent an extensive evaluation process that is thoroughly discussed in Chapter 6. Overall, quantitative experiments results showed the efficiency of our provenance-based methods implemented in EVLIN (our prototype for visual exploration of data warehouses). We performed also qualitative experiments results to study the effectiveness of our provenance-based methods. The results of qualitative experiments showed general satisfaction among users when visually exploring data using EVLIN.

Inspired by the important role of evolution provenance aggregation towards supporting collaborative-filtering recommendation in our framework ProvVDE, we proposed finally in Chapter 7 a new aggregation approach that summarizes provenance documents following W3C-PROV standard. Therefore, we showed the usefulness of our structure-based provenance summaries on several use cases, when using appropriate visualizations and interaction techniques. Additionally, we performed a preliminary experimental evaluation that studies the performance of our proposed provenance summary process. Overall, experiments results showed the capability of our proposed approach to process rapidly several input provenance traces and to output concise summaries easy to analyze.

8.2. Perspectives

In this section, we briefly present some possible research topics for future work building upon results from this thesis.

8.2.1. Interactive and comparative analysis of provenance traces

We are interested in researching methods to explore provenance traces, both individually and collectively using interactive visual analytics techniques. In our current work described in this manuscript, we have the opportunity to collect diverse types of real provenance such as: (i) the evolution provenance traces of data exploration sessions that we generate in our framework ProvVDE or that are collected using the visual analytics framework developed in [BBH+19] (ii) the why provenance traces collected also in our visual data exploration framework ProvVDE when investigating exploration queries' results (iii) the W3C-PROV provenance collected and summarized using our approach discussed in Chapter 7. We have also implemented diverse synthetic provenance generators that mimic different types of provenance e.g. the evolution provenance (in Chapter 6) and the W3C-PROV provenance traces (in Chapter 7).

Given the availability of various provenance information, an interesting new research direction would be the visual exploration of provenance data. Our work described in this thesis shows the prominent role played by visual exploration towards revealing and surfacing insights. Similarly, visual data exploration techniques including namely recommendations could be proposed to analyze provenance information.

The first step towards this goal was already done as described in Chapter 7. Hence, we have proposed a method to summarize a provenance set containing W3C-PROV provenance traces. Accordingly, we have defined a set of visual analysis tasks that could be applied to this kind of summary. Performing efficiently the set of defined visual analysis tasks requires the design of convenient visualizations that render appropriately provenance summaries. As a consequence, we need in the future to propose a new visualization recommendation approach that renders the suitable visualization of a provenance summary with respect to the visual analysis task.

8.2.2. Quantification of uncertain provenance traces

The provenance traces collected in this thesis are complete in the sense that they comprise all possible/modeled data of the provenance traces. Yet, collecting these complete traces may result in a significant overhead to both the process runtime and storage. Given also that we target interactive visual exploration applications, it is important to support quick access to provenance data in the provenance management backend system. There are already some existing solutions that tackle this issue. For instance, Diestelkämper et al. [DH20] propose an approach that reduces the overhead of provenance computation in DISC systems. Overall, these solutions provide interesting approaches that reduce significantly the runtime/storage overhead when computing provenance. Yet, these techniques typically incur some quality issues, e.g., uncertainty etc.

To this end, an interesting new research direction would be to quantify the quality of the provenance information collected by these systems. Accordingly, it will be highly interesting to communicate the provenance quality issues to the user as part of the interactive and visual representation of the provenance. To this end, an interesting research avenue will consist of studying the methods required to quantify and visualize the quality of provenance.

Bibliography

[ABD+15] E. Ainy, P. Bourhis, S. Davidson, D. Deutch, T. Milo. 'Approximated Summarization of Data Provenance'. In: *Conference on Information and Knowledge Management (CIKM)*. 2015, pp. 483–492 (cit. on p. 170).

[ABGK13] P. Alper, K. Belhajjame, C. A. Goble, P. Karagoz. 'Enhancing and Abstracting Scientific Workflow Provenance for Data Publishing'. In: *EDBT/ICDT Workshops*. 2013, pp. 313–318 (cit. on p. 30).

[ABJ06] I. Altintas, O. Barney, E. Jaeger-Frank. 'Provenance collection support in the Kepler scientific workflow system'. In: *International Provenance and Annotation Workshop (IPAW)*. 2006, pp. 118–132 (cit. on p. 31).

[ACD+18] A. Alawini, L. Chen, S. Davidson, S. Fisher, J. Kim. 'Discovering Similar Workflows via Provenance Clustering: A Case Study'. In: *International Provenance and Annotation Workshop (IPAW)*. 2018, pp. 115–127 (cit. on p. 186).

[ACN00] S. Agrawal, S. Chaudhuri, V. R. Narasayya. 'Automated Selection of Materialized Views and Indexes in SQL Databases'. In: *Conference on Very Large Data Bases (VLDB)*. 2000, pp. 496–505 (cit. on p. 76).

[AGN15] Z. Abedjan, L. Golab, F. Naumann. 'Profiling relational data: a survey'. In: *The VLDB Journal* 24.4 (2015), pp. 557–581 (cit. on p. 66).

[BBC+12] K. Belhajjame, R. B'Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne, T. Lebo, J. McCusker, S. Miles, J. Myers, S. Sahoo, C. Tilmes. *PROV-DM: The PROV Data Model*. Tech. rep. 2012. URL: http://www.%20w3.%20org/TR/prov-dm/ (cit. on p. 172).

[BBC+17] M. A. Baazizi, H. Ben Lahmar, D. Colazzo, G. Ghelli, C. Sartiani. 'Schema Inference for Massive JSON Datasets'. In: *Conference on Extending Database Technology (EDBT)*. 2017, pp. 222–233 (cit. on pp. 170–172, 175, 180).

[BBD07] O. Biton, S. C. Boulakia, S. B. Davidson. 'Zoom*UserViews: Querying Relevant Provenance in Workflow Systems'. In: *Conference on Very Large Data Bases (VLDB)*. 2007, pp. 1366–1369 (cit. on p. 170).

[BBH+19] V. Bruder, H. Ben Lahmar, M. Hlawatsch, S. Frey, M. Burch, D. Weiskopf, M. Herschel, T. Ertl. 'Volume-based large dynamic graph analysis supported by evolution provenance'. In: *Multimedia Tools and Applications Journal* 78.23 (2019), pp. 32939–32965 (cit. on pp. 16, 58, 196).

[BDM09] R. E. Burkard, M. Dell'Amico, S. Martello. *Assignment Problems*. Society for Industrial and Applied Mathematics, 2009 (cit. on p. 99).

[BH17] H. Ben Lahmar, M. Herschel. 'Provenance-based Recommendations for Visual Data Exploration'. In: *Workshop on Theory and Practice of Provenance (TAPP)*. 2017 (cit. on pp. 16, 17, 20, 21, 47, 61, 114).

[BH19a] H. Ben Lahmar, M. Herschel. 'Structural summaries for visual provenance analysis'. In: *Workshop on Theory and Practice of Provenance (TAPP)*. 2019 (cit. on pp. 19, 21, 170).

[BH19b] H. Ben Lahmar, M. Herschel. 'Towards integrating collaborative filtering in visual data exploration systems'. In: *Advances in Databases and Information Systems (ADBIS)*. 2019, pp. 153–160 (cit. on pp. 17, 20, 21, 47, 61, 79, 82, 85, 88, 114).

[BH20] H. Ben Lahmar, M. Herschel. 'Collaborative filtering over evolution provenance data for interactive visual data exploration'. In: *Journal of Information Systems*. 2020 (cit. on pp. 17, 18, 20, 21, 61, 79, 91, 114).

[BHBK18] H. Ben Lahmar, M. Herschel, M. Blumenschein, D. A. Keim. 'Provenance-Based Visual Data Exploration with EVLIN'. In: *Conference on Extending Database Technology (EDBT)*. 2018, pp. 686–689 (cit. on pp. 19, 20, 36, 61, 72).

[BKT01] P. Buneman, S. Khanna, W. C. Tan. ‘Why and Where: A Characterization of Data Provenance’. In: *International Conference on Database Theory (ICDT)*. 2001, pp. 316–330 (cit. on p. 32).

[BOH11] M. Bostock, V. Ogievetsky, J. Heer. ‘D3: Data-Driven Documents’. In: 17.12 (2011), pp. 2301–2309 (cit. on p. 50).

[BTGM17] L. Barakat, P. Taylor, N. Griffiths, S. Miles. ‘Corroboration via Provenance Patterns’. In: *Workshop on Theory and Practice of Provenance (TAPP)*. 2017 (cit. on pp. 186, 188).

[BVW03] T. Brijs, K. Vanhoof, G. WETS. ‘Defining interestingness for association rule’. In: *Information Theories Applications* 10 (Jan. 2003) (cit. on p. 78).

[BYB+13] M. A. Borkin, C. S. Yeh, M. Boyd, P. Macko, K. Z. Gajos, M. Seltzer, H. Pfister. ‘Evaluation of filesystem provenance visualization tools’. In: *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 19.12 (2013), pp. 2476–2485 (cit. on p. 170).

[CBYE19] Z. Cui, S. K. Badam, M. A. Yalçin, N. Elmqvist. ‘DataSite: Proactive visual data exploration with computation of insight-based recommendations’. In: *Information Visualization Journal* 18.2 (2019) (cit. on p. 24).

[CCT09] J. Cheney, L. Chiticariu, W. C. Tan. ‘Provenance in Databases: Why, How, and Where’. In: *Foundations and Trends in Databases* 1.4 (2009), pp. 379–474 (cit. on p. 62).

[CFDC17] V. Curcin, E. Fairweather, R. Danger, D. Corrigan. ‘Templates As a Method for Implementing Data Provenance in Decision Support Systems’. In: *Journal of Biomedical Informatics* 65 (2017), pp. 1–21 (cit. on p. 171).

[CFS+06] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, H. T. Vo. ‘VisTrails : Visualization meets Data Management’. In: *ACM Conference on the Management of Data (SIGMOD)* (2006), pp. 745–747 (cit. on pp. 31, 48).

[CKP08] L. Chiticariu, P. G. Kolaitis, L. Popa. ‘Interactive generation of integrated schemas’. In: *ACM Conference on the Management of Data (SIGMOD)*. 2008, pp. 833–846 (cit. on p. 171).

[CM84] W. S. Cleveland, R. McGill. 'Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods'. In: *Journal of the American Statistical Association* 79.387 (1984), pp. 531–554 (cit. on p. 103).

[CW01] Y. Cui, J. Widom. 'Lineage Tracing for General Data Warehouse Transformations'. In: *Conference on Very Large Data Bases (VLDB)*. 2001, pp. 471–480 (cit. on p. 32).

[DBK+15] S. Dey, K. Belhajjame, D. Koop, M. Raul, B. Ludäscher. 'Linking Prospective and Retrospective Provenance in Scripts'. In: *Workshop on Theory and Practice of Provenance (TAPP)*. 2015 (cit. on p. 31).

[DDL+19] M. Djedaini, K. Drushku, N. Labroche, P. Marcel, V. Peralta, W. Verdeaux. 'Automatic assessment of interactive OLAP explorations'. In: *Journal of Information Systems* 82 (2019), pp. 148–163 (cit. on p. 79).

[DH20] R. Diestelkämper, M. Herschel. 'Tracing nested data with structural provenance for big data analytics'. In: *Conference on Extending Database Technology (EDBT)*. 2020, pp. 253–264 (cit. on p. 197).

[Dij59] E. W. Dijkstra. 'A Note on Two Problems in Connexion with Graphs'. In: *Numer. Math.* 1.1 (1959), pp. 269–271 (cit. on p. 107).

[DLMP17] M. Djedaini, N. Labroche, P. Marcel, V. Peralta. 'Detecting User Focus in OLAP Analyses'. In: *Advances in Databases and Information Systems (ADBIS)*. Vol. 10509. Lecture Notes in Computer Science. 2017, pp. 105–119 (cit. on p. 79).

[DP13] M. Drosou, E. Pitoura. 'YmalDB: Exploring relational databases via result-driven recommendations'. In: *The VLDB Journal* 22.6 (2013), pp. 849–874 (cit. on p. 14).

[EAPS14] M. Eirinaki, S. Abraham, N. Polyzotis, N. Shaikh. 'QueRIE: Collaborative Database Exploration'. In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 26.7 (2014), pp. 1778–1790 (cit. on p. 101).

[EKA+08] T. Ellkvist, D. Koop, E. W. Anderson, J. Freire, C. T. Silva. 'Using Provenance to Support Real-Time Collaborative Design of Workflows'. In: *International Provenance and Annotation Workshop (IPAW)*. 2008, pp. 266–279 (cit. on p. 48).

[ESC18] H. Ehsan, M. A. Sharaf, P. K. Chrysanthis. 'Efficient Recommendation of Aggregate Data Visualizations'. In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 30.2 (2018), pp. 263–277 (cit. on pp. 25, 26, 29, 30, 71–73, 78).

[FLM+10] W. Fan, J. Li, S. Ma, H. Wang, Y. Wu. 'Graph Homomorphism Revisited for Graph Matching'. In: *Proceedings of the VLDB Endowment (PVLDB)* 3.1-2 (2010), pp. 1161–1172 (cit. on pp. 83, 84).

[GA09] B. Glavic, G. Alonso. 'The Perm Provenance Management System in Action'. In: *ACM Conference on the Management of Data (SIGMOD)*. 2009, pp. 1055–1058 (cit. on pp. 62, 114).

[GGY+14] M. Gupta, J. Gao, X. Yan, H. Cam, J. Han. 'Top-K interesting subgraph discovery in information networks'. In: *IEEE International Conference on Data Engineering (ICDE)*. 2014, pp. 820–831 (cit. on p. 100).

[GH06] L. Geng, H. J. Hamilton. 'Interestingness Measures for Data Mining: A Survey'. In: *ACM Computing Surveys (CSUR)* 38.3 (2006) (cit. on p. 78).

[GLG+16] S. Gratzl, A. Lex, N. Gehlenborg, N. Cosgrove, M. Streit. 'From Visual Exploration to Storytelling and Back Again'. In: *European Conference on Visualization (EuroVis)* 35.3 (2016), pp. 491–500 (cit. on pp. 24, 48, 49, 53).

[GS62] D. Gale, L. S. Shapley. 'College Admissions and the Stability of Marriage'. In: *The American Mathematical Monthly* 69.1 (1962) (cit. on pp. 82, 85, 86).

[HDB17] M. Herschel, R. Diestelkämper, H. Ben Lahmar. 'A survey on provenance: What for? What form? What from?' In: *The VLDB Journal* 26.6 (2017), pp. 881–906 (cit. on pp. 23, 30–32, 48).

[HKK+13] M. A. Hernández, G. Koutrika, R. Krishnamurthy, L. Popa, R. Wisnesky. 'HIL: a high-level scripting language for entity integration'. In: *Conference on Extending Database Technology (EDBT)*. 2013, pp. 549–560 (cit. on p. 184).

[HN01] P. Healy, N. S. Nikolov. 'How to Layer a Directed Acyclic Graph'. In: *International Symposium on Graph Drawing (GD)*. 2001, pp. 16–30 (cit. on p. 85).

[HNW06] J. Hegewald, F. Naumann, M. Weis. ‘XStruct: Efficient Schema Extraction from Multiple and Large XML Documents’. In: *IEEE International Conference on Data Engineering (ICDE)*. 2006, p. 81 (cit. on p. 171).

[IPC15] S. Idreos, O. Papaemmanouil, S. Chaudhuri. ‘Overview of Data Exploration Techniques’. In: *ACM Conference on the Management of Data (SIGMOD)*. 2015, pp. 277–281 (cit. on p. 24).

[Irv94] R. W. Irving. ‘Stable marriage and indifference’. In: *Discrete Applied Mathematics* 48.3 (1994), pp. 261–272 (cit. on p. 82).

[JV87] R. Jonker, A. Volgenant. ‘A shortest augmenting path algorithm for dense and sparse linear assignment problems’. In: *Computing* 38.4 (1987), pp. 325–340 (cit. on p. 99).

[KB16] M. Kaminskas, D. Bridge. ‘Diversity, Serendipity, Novelty, and Coverage: A Survey and Empirical Analysis of Beyond-Accuracy Objectives in Recommender Systems’. In: *ACM Trans. Interact. Intell. Syst.* 7.1 (2016), 2:1–2:42 (cit. on p. 78).

[KB55] T. Koopmans, M. J. Beckmann. *Assignment Problems and the Location of Economic Activities*. Cowles Foundation Discussion Papers 4. 1955 (cit. on p. 99).

[KFS13] D. Koop, J. Freire, C. T. Silva. ‘Visual summaries for graph collections’. In: *Asia Pacific Symposium on Information Visualisation (PacificVIS)*. 2013, pp. 57–64 (cit. on p. 170).

[Kim96] R. Kimball. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley, 1996 (cit. on p. 27).

[KKBS10] N. Khoussainova, Y. Kwon, M. Balazinska, D. Suciu. ‘SnipSuggest: Context-aware Autocompletion for SQL’. In: *Proceedings of the VLDB Endowment (PVLDB)* 4 (2010), pp. 22–33 (cit. on p. 101).

[Kuh55] H. W. Kuhn. ‘The Hungarian method for the assignment problem’. In: *Naval Res. Logist. Quart* 2 (1955), pp. 83–97 (cit. on p. 99).

[LDH+19] D. J.-L. Lee, H. Dev, H. Hu, H. Elmeleegy, A. Parameswaran. ‘Avoiding Drill-down Fallacies with VisPilot: Assisted Exploration of Data Subsets’. In: *International Conference on Intelligent User Interfaces (IUI)*. 2019, pp. 186–196 (cit. on pp. 73, 78).

[LHS09] M. Leordeanu, M. Hebert, R. Sukthankar. 'An Integer Projected Fixed Point Method for Graph Matching and MAP Inference'. In: *Advances in Neural Information Processing Systems (NIPS)*. 2009, pp. 1114–1122 (cit. on p. 99).

[LQ14] Z.-Y. Liu, H. Qiao. 'GNCCP: Graduated NonConvexity and Concavity Procedure'. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 36.6 (2014), pp. 1258–1267 (cit. on p. 99).

[LSDK18] Y. Liu, T. Safavi, A. Dighe, D. Koutra. 'Graph Summarization Methods and Applications: A Survey'. In: *ACM Comput. Surv.* 51.3 (2018), pp. 1–34 (cit. on p. 100).

[Mac86] J. D. Mackinlay. 'Automating the Design of Graphical Presentations of Relational Information.' In: *ACM Trans. Graph.* 5.2 (1986), pp. 110–141 (cit. on pp. 103, 111).

[MBC+14] L. Murta, V. Braganholo, F. Chirigati, D. Koop, J. Freire. 'noWorkflow: Capturing and Analyzing Provenance of Scripts'. In: *International Provenance and Annotation Workshop (IPAW)*. 2014, pp. 71–83 (cit. on p. 31).

[MBG+14] P. Missier, J. Bryans, C. Gamble, V. Curcin, R. Dánger. 'ProvAbs: Model, Policy, and Tooling for Abstracting PROV Graphs'. In: *International Provenance and Annotation Workshop (IPAW)*. 2014, pp. 3–15 (cit. on pp. 170, 172).

[MBH+18] L. Moreau, B. V. Batlajery, T. D. Huynh, D. Michaelides, H. Packer. 'A Templating System to Generate Provenance'. In: *IEEE Transactions on Software Engineering (TSE)* 44.2 (2018), pp. 103–121 (cit. on p. 171).

[MDB+13] P. Missier, S. Dey, K. Belhajjame, V. Cuevas-Vicenttin, B. Ludäscher. 'D-PROV: Extending the PROV Provenance Model with Workflow Structure'. In: *Workshop on Theory and Practice of Provenance (TAPP)*. 2013, 9:1–9:7 (cit. on p. 31).

[MHS07] J. Mackinlay, P. Hanrahan, C. Stolte. 'Show Me: Automatic Presentation for Visual Analysis'. In: *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 13.6 (2007), pp. 1137–1144 (cit. on pp. 14, 24–26, 103, 110).

[Mor15] L. Moreau. ‘Aggregation by Provenance Types: A Technique for Summarising Provenance Graphs’. In: *Electronic Proceedings in Theoretical Computer Science (ETAPS)*. 2015, pp. 129–144 (cit. on p. 170).

[MS16] T. Milo, A. Somech. ‘REACT: Context-Sensitive Recommendations for Data Analysis’. In: *ACM Conference on the Management of Data (SIGMOD)*. 2016, pp. 2137–2140 (cit. on pp. 14, 25, 26, 29, 30, 48, 49, 101).

[MS18] T. Milo, A. Somech. ‘Next-Step Suggestions for Modern Interactive Data Analysis Platforms’. In: *Conference on Knowledge Discovery & Data Mining (KDD)*. 2018, pp. 576–585 (cit. on pp. 14, 25, 26, 29, 30, 101).

[MSK18] R. Mafrur, M. A. Sharaf, H. A. Khan. ‘DiVE: Diversifying View Recommendation for Visual Data Exploration’. In: *Conference on Information and Knowledge Management (CIKM)*. 2018, pp. 1123–1132 (cit. on pp. 25, 26, 29, 30, 71, 72, 78).

[MVT16] B. Mutlu, E. Veas, C. Trattner. ‘VizRec: Recommending Personalized Visualizations’. In: *ACM Trans. Interact. Intell. Syst.* 6.4 (2016), 31:1–31:39 (cit. on pp. 14, 25, 26, 110, 111).

[OH18] S. Oppold, M. Herschel. ‘Provenance for Entity Resolution’. In: *International Provenance and Annotation Workshop (IPAW)*. 2018 (cit. on p. 184).

[OMO+16] W. M. de Oliveira, P. Missier, K. Ocaña, D. de Oliveira, V. Braganholo. ‘Analyzing Provenance Across Heterogeneous Provenance Graphs’. In: *International Provenance and Annotation Workshop (IPAW)*. 2016, pp. 57–70 (cit. on p. 170).

[PBRT99] J. Puzicha, J. M. Buhmann, Y. Rubner, C. Tomasi. ‘Empirical evaluation of dissimilarity measures for color and texture’. In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. 1999, pp. 1165–1172 (cit. on p. 73).

[Qli] QlikView. *QlikView*. URL: https://www.qlik.com (visited on 03/27/2016) (cit. on pp. 29, 30).

[RHF05] P. Riehmann, M. Hanfler, B. Froehlich. 'Interactive Sankey diagrams'. In: *IEEE Symposium on Information Visualization (INFOVIS)*. 2005, pp. 233–240 (cit. on p. 182).

[RHS14] S. Ranu, M. Hoang, A. Singh. 'Answering Top-k Representative Queries on Graph Databases'. In: *ACM Conference on the Management of Data (SIGMOD)*. 2014, pp. 1163–1174 (cit. on pp. 95, 100).

[Sar00] S. Sarawagi. 'User-Adaptive Exploration of Multidimensional Data'. In: *Conference on Very Large Data Bases (VLDB)*. 2000, pp. 307–316 (cit. on p. 79).

[SK16a] T. Sellam, M. Kersten. 'Fast, Explainable View Detection to Characterize Exploration Queries'. In: *Conference on Scientific and Statistical Database Management (SSDBM)*. 2016, 20:1–20:12 (cit. on p. 73).

[SK16b] T. Sellam, M. Kersten. 'Ziggy : Characterizing Query Results for Data Explorers'. In: *Proceedings of the VLDB Endowment (PVLDB)* 9.13 (2016), pp. 1473–1476 (cit. on pp. 14, 24–26, 29, 30, 71, 72, 78).

[SKL+16] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, A. Parameswaran. 'Effortless Data Exploration with Zenvisage: An Expressive and Interactive Visual Analytics System'. In: *Proceedings of the VLDB Endowment (PVLDB)* 10.4 (2016), pp. 457–468 (cit. on p. 78).

[SLSG16] H. Stitz, S. Luger, M. Streit, N. Gehlenborg. 'AVOCADO: Visualization of Workflow-Derived Data Provenance for Reproducible Biomedical Research'. In: *Comput. Graph. Forum* 35.3 (2016), pp. 481–490 (cit. on p. 170).

[SMWH17] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, J. Heer. 'Vega-Lite: A Grammar of Interactive Graphics'. In: 23.1 (2017), pp. 341–350 (cit. on p. 50).

[STH02] C. Stolte, D. Tang, P. Hanrahan. 'Polaris: A system for query, analysis, and visualization of multidimensional relational databases'. In: *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 8 (2002), pp. 52–65 (cit. on pp. 24, 103, 110).

[SZG+18] C. Schulz, A. Zeyfang, M. van Garderen, H. Ben Lahmar, M. Herschel, D. Weiskopf. 'Simultaneous Visual Analysis of Multiple Software Hierarchies'. In: *IEEE Working Conference on Software Visualization (VISSOFT)*. 2018, pp. 87–95 (cit. on p. 85).

[Tab] Tableau. *Tableau*. URL: https://www.tableau.com (visited on 04/12/2016) (cit. on pp. 29, 30).

[THY+17] B. Tang, S. Han, M. L. Yiu, R. Ding, D. Zhang. 'Extracting Top-K Insights from Multi-dimensional Data'. In: *ACM Conference on the Management of Data (SIGMOD)*. 2017, pp. 1509–1524 (cit. on pp. 14, 25, 26, 29, 30, 71, 72, 79).

[TMF+10] W. Tan, P. Missier, I. Foster, R. Madduri, D. De Roure, C. Goble. 'A comparison of using Taverna and BPEL in building scientific workflows: The case of caGrid'. In: *Concurrency Computation Practice and Experience* 22.9 (2010), pp. 1098–1117 (cit. on p. 31).

[Tuk77] J. W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977 (cit. on p. 13).

[VRM+15] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, N. Polyzotis. 'SEEDB: Efficient Data-Driven Visualization Recommendations to Support Visual Analytics'. In: *Proc. VLDB Endow.* 8.13 (2015), pp. 2182–2193 (cit. on pp. 14, 24–26, 29, 30, 71–73, 78).

[VS99] P. Vassiliadis, T. Sellis. 'A survey of logical models for OLAP databases'. In: *ACM SIGMOD Record* 28 (1999), pp. 64–69 (cit. on pp. 27, 28).

[WMA+16] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, J. Heer. 'Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations'. In: *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 22.1 (2016), pp. 649–658 (cit. on pp. 14, 24–26, 29, 30, 71, 72, 103, 104, 110).

[Won18] K. Wongsuphasawat. *Augmenting Exploratory Data Analysis with Visualization Recommendation*. University of Washington Libraries, 2018. URL: https://books.google.de/books?id=wqAuxQEACAAJ (cit. on pp. 24, 103).

[WPM+17] E. Wu, F. Psallidas, Z. Miao, H. Zhang, L. Rettig. 'Combining Design and Performance in a Data Visualization Management System'. In: *Conference on Innovative Data Systems Research (CIDR)*. 2017 (cit. on p. 50).

[WQM+17] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. Mackinlay, B. Howe, J. Heer. 'Voyager 2: Augmenting Visual Analysis with Partial View Specifications'. In: *Conference on Human Factors in Computing Systems (CHI)*. 2017, pp. 2648–2659 (cit. on pp. 14, 25, 26, 29, 30, 71, 72, 110, 158).

[YYL+16] J. Yan, X.-C. Yin, W. Lin, C. Deng, H. Zha, X. Yang. 'A Short Survey of Recent Advances in Graph Matching'. In: *ACM International Conference on Multimedia Retrieval (ICMR)*. 2016, pp. 167–174 (cit. on p. 99).

[ZQYC12] Y. Zhu, L. Qin, J. X. Yu, H. Cheng. 'Finding Top-k Similar Graphs in Graph Databases'. In: *Conference on Extending Database Technology (EDBT)*. 2012, pp. 456–467 (cit. on pp. 95, 100).

List of Figures

List of Algorithms

List of Tables

Definitions

Appendix A

We provide in this appendix further details about our visual recommendation approach discussed in Section 5.5. More specifically, we show in this appendix the pseudocode of the $ReuseVIS$ function that is invoked when recommending visualizations in EVLIN, the instance of our framework meant to explore data warehouses. Note that this function is invoked in Algorithm 7. It consists on maximizing the usage of visual encodings properties generated previously in order to increase consistency between visualizations output by our framework ProvVDE. Algorithm 7 describes this function $ReuseVIS$. It receives initially two inputs that are the recommended visualization under construction V_{curr} and a previous visualization V_i. Note that the recommended visualization V_{curr} is not yet completely specified. Hence, it contains some missing visual components. The goal of Algorithm 7 is to complete the recommended visualization V_{curr} by re-using information available in the visualization V_i. To do that, we adopt in Algorithm 7 a rule-based strategy to construct the recommended visualization. At each condition (see Lines 1, 3, 5, 9, 13), we check first whether a specific visual component is already constructed or not for the recommended visualization. If such visual encoding is not yet designed for the recommended visualization V_{curr}, we verify further conditions (e.g, in lines 5, 9, and 13) to verify the feasibility of re-suing such visual encoding from V_i. If these conditions are fulfilled, the previous visual encoding allocated for V_i, is re-used again in the recommended visualization V_{curr}.

Algorithm 10: ReuseVis(V_{curr}, V_i)

```
   Input: $V_{curr}$: recommended visualization under construction ,$V_i$: a
          previous visualization whose resources will be re-used
   Output: $V_{curr}$: Recommended visualization associated to $Q_{curr}$
1  if ($V_{curr}$.getLength().isEmpty) then
2      $V_{curr}$ ← SetLength($V_i$.length) ;
3  if ($V_{curr}$.getWidth().isEmpty) then
4      $V_{curr}$ ← SetWidth($V_i$.length) ;
5  if ($V_{curr}.ScaleX.isEmpty$ AND $groupBy(Q_{curr})=groupBy(Q_i)$) then
6      $V_{curr}$ ← SetScaleX($V_i$.scaleX);
7      $V_{curr}.getMark.encodingX \leftarrow V_i.getMark.encodingX$;
8      $V_{curr}$ ← SetXAxe($V_i$.getX);
9  if ($V_{curr}.ScaleY.isEmpty$ AND $aggFct(Q_{curr})=aggFct(Q_i)$ AND
     $whereClause(Q_{curr})=whereClause(Q_i)$) then
10     $V_{curr}$ ← SetScaleY($V_i$.scaleY) ;
11     $V_{curr}.getMark.encodingY \leftarrow V_i.getMark.encodingY$ ;
12     $V_{curr}$ ← SetYAxe($V_i$.getY) ;
13 if ($V_{curr}.ScaleZ.isEmpty$ AND
     $groupBy(Q_{curr}).get(2)=groupBy(Q_i).get(2)$) then
14     if ($V_i.hasColorScale$) then
15         $V_{curr}$ ← SetScaleColor($V_i$.scaleColor);
16         $V_{curr}.getMark.encodingfill \leftarrow V_i.getMark.encodingfill$;
17     else if ($V_i.hasColorSize$) then
18         $V_{curr}$ ← SetScaleSize($V_i$.scaleSize);
19         $V_{curr}.getMark.encodingsize \leftarrow V_i.getMark.encodingsize$
20     else
21         $V_{curr}$ ← SetScaleShape($V_i$.scaleShape);
22         $V_{curr}$.getMark.encodingShape ← $V_i$.getMark.encodingShape;
23 Return $V_{curr}$;
```

Appendix B

We have described in Section 5.4.3.3, our parenthood based pruning approach to optimize the computation of collaborative filtering when processing multi-users graphs. We have also provided in the same Section 5.4.3.3 a proof of our parenthood based approach applied for This proof applies for exploration queries of types “Zoom IN” and “Extension”.

In this appendix, we provide remaining proofs that demonstrate the applicability of our parenthood based pruning approach on other types of exploration queries generated using our system EVLIN (cf. Section 5.2.3, Page 67). More specifically, we provide first a proof demonstrating that our pruning approach holds for queries of types “Zoom IN/Slice” or “Extension/Slice” . We provide later a second proof showing the applicability of our approach of queries of types “Drill” or “Drill down”.

Proof 5 *This proof applies for edge e between X_{parent} to X_{child} having “Zoom IN/Slice” or “Extension/Slice” as label for the operation type.*

Let $getSelect(Q) = \{a, f(m)\}$, $getSelect(Q_{parent}) = \{a_p, f_p(m_p)\}$ and $Q_{child} = \{a_c, f_p(m_p)\}$. Note that the select clauses of Q_{parent} and Q_{child} are guaranteed to overlap only in $f_p(m_p)$) based on our derivation rules described *in Figure 5.2.*

$$Jaccard(Q_{child}, Q) = \frac{\{a, f(m)\} \cap \{a_c, f_p(m_p)\}}{\{a, f(m), a_c, f_p(m_p)\}}$$

By removing a_c from the denominator, $Jaccard(Q_{child}, Q) \leq \frac{\{a, f(m)\} \cap \{a_c, f_p(m_p)\}}{\{a, f(m), f_p(m_p)\}}$ By adding a_p to the numerator and to the the denominator, $Jaccard(Q_{child}, Q) \leq \frac{\left(\{a, f(m)\} \cap \{a_c, f_p(m_p)\}\right) \cup \{a_p\}}{\{a, a_p, f(m), f_p(m_p)\}}$

This is equivalent to: $Jaccard(Q_{child}, Q) \leq \frac{\{a, a_p, f(m)\} \cap \left(\{a_c, a_p, f_p(m_p)\}\right)}{\{a, a_p, f(m), f_p(m_p)\}}$

$$\Rightarrow Jaccard(Q_{child}, Q) \leq \frac{\{a, a_p, f(m)\} \cap \left(\{a_c, a_p, f_p(m_p)\}\right)}{\{a, a_p, f(m), f_p(m_p)\}}$$

$$\Rightarrow Jaccard(Q_{child}, Q) \leq \frac{\left(\{a, a_p, f(m)\} \cap \{a_p, f_p(m_p)\}\right) \cup \left(\{a, a_p, f(m)\} \cap \{a_c\}\right)}{\{a, a_p, f(m), f_p(m_p)\}}$$

$$\Rightarrow Jaccard(Q_{child}, Q) \leq \frac{\left(\{a, f(m)\} \cap \{a_p, f_p(m_p)\}\right) \cup (\left(\{a_p\} \cap \{a_p, f_p(m_p)\}\right) \cup \left(\{a, a_p, f(m)\} \cap \{a_c\}\right)}{\{a, a_p, f(m), f_p(m_p)\}}$$

$$\Rightarrow Jaccard(Q_{child}, Q) \leq \frac{\left(\{a, f(m)\} \cap \{a_p, f_p(m_p)\}\right) \cup (\left(\{a_p\}\right) \cup \left(\{a, a_p, f(m)\} \cap \{a_c\}\right)}{\{a, a_p, f(m), f_p(m_p)\}}$$

$$\Rightarrow Jaccard(Q_{child}, Q) \leq \frac{\left(\{a, f(m)\} \cap \{a_p, f_p(m_p)\}\right)}{\{a, a_p, f(m), f_p(m_p)\}} + \frac{\left(\{a_p\}\right) \cup \left(\{a, a_p, f(m)\} \cap \{a_c\}\right)}{\{a, a_p, f(m), f_p(m_p)\}}$$

$$\Rightarrow Jaccard(Q_{child}, Q) \leq Jaccard(Q_{parent}, Q) + \frac{\left(\{a_p\}\right) \cup \left(\{a, a_p, f(m)\} \cap \{a_c\}\right)}{\{a, a_p, f(m), f_p(m_p)\}}$$

$$\Rightarrow Jaccard(Q_{child}, Q) \leq Jaccard(Q_{parent}, Q) + \frac{(\{a, f(m)\} \cap \{a_c\})}{\{a, f(m), f_p(m_p)\}}$$

Proof parenthood lower bound Drill/Drill-down

Proof 6 *This proof applies for edge e between X_{parent} to X_{child} having "Drill" or "Drill down" as label for the operation type.*

Let $getSelect(Q) = \{a, f(m)\}$, $getSelect(Q_{parent}) = \{a_p, f_p(m_p)\}$ and $Q_{child} = \{a_c, a_g, f_p(m_p)\}$. Note that the select clauses of Q_{parent} and Q_{child}

are guaranteed to overlap only in $f_p(m_p)$) based on our derivation rules (cf. Figure 5.2)

$$Jaccard(Q_{child}, Q) = \frac{\{a, f(m)\} \cap \{a_c, a_g, f_p(m_p)\}}{\{a, f(m), a_c, a_g, f_p(m_p)\}}$$

By removing a_c and a_g from the denominator, $Jaccard(Q_{child}, Q) \leq \frac{\{a,f(m)\}\cap\{a_c,a_g,f_p(m_p)\}}{\{a,f(m),f_p(m_p)\}}$ By adding a_p to the numerator and to the the denominator, $Jaccard(Q_{child}, Q) \leq \frac{\{a_p,a,f(m)\}\cap\{a_p,a_c,a_g,f_p(m_p)\}}{\{a_p,a,f(m),f_p(m_p)\}}$

This is equivalent to:

$Jaccard(Q_{child}, Q) \leq \frac{\big(\{a_p,a,f(m)\}\cap\{a_p,f_p(m_p)\}\big)\cup\big(\{a_p,a,f(m)\}\cap\{a_c,a_g\}\big)}{\{a,a_p,f(m),f_p(m_p)\}}$

$Jaccard(Q_{child}, Q) \leq \frac{\big(\big(\{a,f(m)\}\cap\{a_p,f_p(m_p)\}\big)\cup\big(\{a_p\}\cap\{a_p,f_p(m_p)\}\big)\big)\cup\big(\{a_p,a,f(m)\}\cap\{a_c,a_g\}\big)}{\{a,a_p,f(m),f_p(m_p)\}}$

$\Rightarrow Jaccard(Q_{child}, Q) \leq Jaccard(Q_{parent}, Q) + \frac{\{a_p\}\cap\{a_p,f_p(m_p)\}}{\{a,a_p,f(m),f_p(m_p)\}} + \frac{\{a_p,a,f(m)\}\cap\{a_c,a_g\}}{\{a,a_p,f(m),f_p(m_p)\}}$

$\Rightarrow Jaccard(Q_{child}, Q) \leq Jaccard(Q_{parent}, Q) + \frac{a_p}{\{a,a_p,f(m),f_p(m_p)\}} + \frac{\{a_p,a,f(m)\}\cap\{a_c,a_g\}}{\{a,a_p,f(m),f_p(m_p)\}}$

$\Rightarrow Jaccard(Q_{child}, Q) \leq Jaccard(Q_{parent}, Q) + \frac{\{a_p\}\cup\big(\{a_p,a,f(m)\}\cap\{a_c,a_g\}\big)}{\{a,a_p,f(m),f_p(m_p)\}}$

$\Rightarrow Jaccard(Q_{child}, Q) \leq Jaccard(Q_{parent}, Q) + \frac{\{a,f(m)\}\cap\{a_c,a_g\}}{\{a,f(m),f_p(m_p)\}}$

Clarification

I hereby declare that, with the exception of the expressly designated aids and the advice of persons listed by name, I have independently written the dissertation.

(Houssem Ben Lahmar)