Interaction in Progressive Visual Analytics. An application to progressive sequential pattern mining

Vincent Raveneau

DOCTORAT BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE NANTES

THÈSE DE DOCTORAT DE

L'UNIVERSITE DE NANTES

Ecole Doctorale N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : INFO – Informatique

Par

« **Vincent RAVENEAU** »

« **Interaction in Progressive Visual Analytics** »

« An application to progressive sequential pattern mining »

Thèse présentée et soutenue à NANTES , le 4 Novembre 2020
Unité de recherche : LS2N – Laboratoire des Sciences du Numérique de Nantes

Rapporteurs avant soutenance :

Nicolas LABROCHE	Maître de conférences HDR, LIFAT, Université de Tours
Adam PERER	Assistant Research Professor, Human-Computer Interaction Institute, Carnegie Mellon University

Composition du jury :

Présidente :	Béatrice DAILLE	Professeure des universités, LS2N, Université de Nantes
Examinateur :	Jean-Daniel FEKETE	Directeur de Recherche, INRIA Saclay, INRIA
Dir. de thèse :	Yannick PRIÉ	Professeur des universités, LS2N, Université de Nantes
Co-enc. de thèse :	Julien BLANCHARD	Maître de conférences, LS2N, Université de Nantes

ACKNOWLEDGMENTS

First, I would like to thank the members of my PhD committee: Nicolas Labroche and Adam Perer, for agreeing to review my work and provide valuable feedback; Béatrice Daille, for agreeing to preside this committee; Jean-Daniel Fekete, for his role in the committee as well as for all the great discussions and support throughout the thesis.

Second, I would like to express my gratitude to my advisors, Yannick Prié and Julien Blanchard, for their guidance, support and help during this work.

I would also like to thank the members of the LS2N laboratory that I have been pleased to meet during these years. This includes members of the DUKe team, as well as my fellow PhD students – Adeline, David, Erwan, Ziwei, Jiajun and Adrien.

Finally, my sincere thanks goes to my friends and family for their continuous support, even though I spent too little time with them during these years. I am very much thankful to my girlfriend, Maëva, for her love and encouragements.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This manuscript presents the work conducted during my PhD thesis at the University of Nantes under the supervision of Yannick Prié and Julien Blanchard, as a member of the DUKe team (**D**ata **U**ser **K**nowledg**e**) of the LS2N laboratory (**L**aboratoire des **S**ciences du **N**umérique de **N**antes). The funding came from the French Ministère de l'Éducation Supérieure et de la Recherche. A timeline with the milestones of this work is available in figure 1.1.

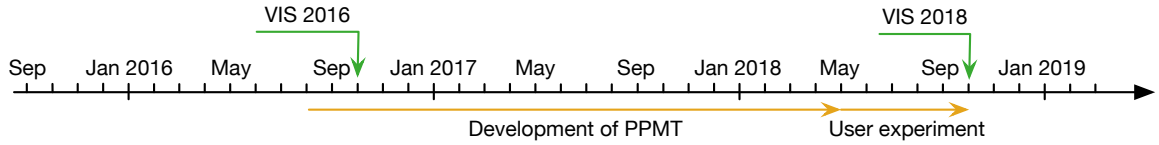


Figure 1.1: Timeline representing the different steps in our work. *Publications are indicated above the axis.*

Scientific context

While data analysis predates the invention of computers, the processing power they offer has led to many new possibilities besides raw number crunching, such as data transformation and visualization. The paradigm of Visual Analytics proposes that data analysis should benefit from both human and computer respective strengths (Bertini & Lalanne, 2010; D. Keim, Kohlhammer, Ellis, & Mansmann, 2010), by combining humans' ease to derive insights from data visualizations and make decisions, with computers' ability to process very large amounts of data. However, these processes suffer from one major pain point: when using time-expensive algorithms or working with large datasets, the analyst has to wait for the computation to complete. This speed bump (Pezzotti et al., 2017; Stolper, Perer, & Gotz, 2014) is detrimental to their focus and ability to explore the data at hand (Liu & Heer, 2014).

From this observation, new research domains have emerged that offer alternatives to these imposed idle times for the analyst. The paradigm of Progressive Visual Analytics is one of these, and its solution is articulated around two core ideas that define the characteristics of progressive algorithms. The first one is to output intermediate results of increasing quality during the computation, instead of waiting for a unique result at the end. This allows the analyst

to start working on a broad picture of what can be expected, that will be improved as the algorithm's computation continues. The second idea is that the analyst is able to interact with the running algorithm, by steering its remaining computation without restarting it. Combined with the intermediate results, this allows the analyst to leverage his interpretations and findings from the early results to guide the analysis towards targets of interest.

Our work takes place within the Progressive Visual Analytics paradigm, and focuses on the exploratory analysis of activity traces. This kind of temporal data consists of events recorded during specific human activities, and can be collected for a wide range of contexts, at various granularity levels (from broad actions to very precise events). As such, exploring activity data can be used to gain insights into the behavior of people, either in isolation or with regards to others' recordings. One can then infer useful knowledge, depending on the kind of data that was recorded. For example, records of medical events can be used to investigate bad combinations of treatments, or long-time side effects. Another application could be learning how software is really used from the recordings of its users, in order to improve it. Activity data usually consists of large amount of data (especially when recorded at a fine granularity), which means they can benefit from the Progressive Visual Analytics approach. This kind of data can be explored using various algorithmic techniques. In this work, we decided to use pattern mining.

Challenges and research goals

Our work is focused on the interaction between a human analyst and a progressive algorithm. Based on our review of the literature, we identified four challenges that drove our work on this matter. The first two are directly related to interaction in Progressive Visual Analytics (PVA), while the last two target Progressive Pattern Mining (PPM).

Challenge PVA1

Clarify what "interaction" means in the context of Progressive Visual Analytics, as well as investigate the role of the algorithm in the process.

Challenge PVA2

Investigate the consequences of interactions between the human and the algorithm on the analysis process.

Challenge PPM1

Investigate progressive pattern mining as a tool for data exploration and not only for pattern exploration.

Challenge PPM2

Investigate the ways to make a pattern mining algorithm progressive, the relevant interactions when using such algorithm and how they should be implemented.

Contributions

We strived to provide theoretical, practical and experimental contributions to our research domain. We identified the following six contributions, of which we consider four to be major.

Major contributions

C1. A study on interaction within a progressive analysis process. This contribution targets our PVA1 and PVA2 challenges, and encompasses the following elements:

- **C1.1.** A framework describing the actions an analyst can perform when interacting with a progressive algorithm;
- **C1.2.** A model of how these actions impact a generic data analysis system;
- **C1.3.** An evaluation of the impact of "progressiveness" on an algorithm's performances.

C2. A clarification of the process of Progressive Visual Analytics, with regards to the role of the algorithm in the process. This contribution targets our PVA1 challenge, and encompasses the following elements:

- **C2.1.** A clear definition for the notion of *steering*;
- **C2.2.** An updated definition of Progressive Visual Analytics, that offers a more precise vision of the interactions that can take place within Progressive Visual Analytics;
- **C2.3.** A model of the Progressive Visual Analytics process, based on a classical Visual Analytics model (Sacha et al., 2014).

C3. PPMT, a progressive pattern mining system. This contribution targets our PPM1 and PPM2 challenges, and encompasses the following elements:

- **C3.1.** The PPMT system itself, that supports the analysis tasks we identified (C5), implements all the actions from our framework (C1.1), and follows our guidelines (C6);
- **C3.2.** A review of PPMT's compliance with existing recommendations for the design of Progressive Visual Analytics systems.

C4. A user experiment focused on the impact of the interactions between the analyst and the algorithm on a progressive analysis process. This contribution targets our PVA2 challenge.

Secondary contributions

C5. A data and task model for an analyst exploring temporal data with patterns, specialized from the general model of Andrienko and Andrienko (2006). This contribution

targets our PPM2 challenge.

C6. A set of five guidelines for the design of a progressive algorithm for pattern mining in sequences. This contribution targets our PPM1 challenge.

Structure of the manuscript

The first part of the manuscript is entitled “State of the art”. It contains two chapters presenting our review of the state of the art, and identifying several challenges. In **chapter 2**, entitled “From Visual Analytics to Progressive Visual Analytics ”, we focus on Progressive Visual Analytics-related works, starting with an overview of Visual Analytics that leads to Progressive Visual Analytics. We then review existing Progressive Visual Analytics work, focusing on its defining characteristics, the existing systems and their design, as well as on existing interaction-related studies of Progressive Visual Analytics. We then switch in **chapter 3**, “Sequential Pattern mining”, to sequential pattern mining, presenting the different kinds of sequential patterns that can be found in the literature, as well as the different families of pattern mining algorithms.

The second part of the manuscript, “Propositions”, is dedicated to the presentation of our contributions. In **chapter 4**, entitled “Interactions in Progressive Visual Analytics ”, we address the question of interactions in Progressive Visual Analytics (C1) and propose an updated definition of the Progressive Visual Analytics paradigm (C2). **Chapter 5**, “Towards Progressive Pattern Mining ”, focuses on the notion of Progressive Pattern Mining, on the tasks an analyst exploring temporal data with patterns can perform (C5), and on the design of dedicated algorithms (C6). In **chapter 6**, “PPMT: a Progressive Pattern Mining Tool to explore activity data”, we present PPMT, our Progressive Pattern Mining system (C3), and its evaluation (C1). In **chapter 7**, “Comparing the effect of various progressive interactions on data analysis tasks”, we relate a user experiment we conducted that focused on the impact interactions between the analyst and the algorithm have on a progressive analysis process (C4).

In **chapter 8**, “General conclusion”, we finally reflect on various aspects of the presented work, while opening up on future works.

PART I

State of the art

In this first part, we present our review of exiting works in Progressive Visual Analytics and sequential pattern mining, the two research domains we were interested in. For the sake of clarity, each domain is presented in its own chapter.

Chapter 2 is about Progressive Visual Analytics, the data analysis paradigm at the core of our work. While some earlier works comply with its definition, its inception is fairly recent, and came as an answer to some limitations of the Visual Analytics paradigm. As such, in order to offer a comprehensive perspective on Progressive Visual Analytics, we open with a presentation of Visual Analytics in section 2.1. Throughout this section, we review the evolution of the definition of Visual Analytics, present the central concept of *insights*, and present the knowledge generation process and task models that have been proposed for this paradigm. We also present some of the existing Visual Analytics systems, and discuss their various degrees of automation. The section concludes with the limitations that Visual Analytics systems face when dealing with large datasets or time-consuming algorithms. Having presented the context that led to its inception, we focus on Progressive Visual Analytics in section 2.2. We present its fundamental concepts and definition, highlight the differences between progressive algorithms and other closely related algorithm types, and review existing implementations of the paradigm. We also present existing works on the design of Progressive Visual Analytics systems, as well as user studies that focus on interaction within this paradigm. In section 2.3, we conclude the chapter by identifying two main challenges for Progressive Visual Analytics.

Chapter 3 is focused on Sequential Pattern mining. While not exhaustive, this review aims at providing by itself a good understanding of the domain and what it offers in terms of data analysis techniques. In section 3.1, we present the general task of mining patterns in sequences. We first provide necessary definitions, before introducing the two types of Sequential Patterns that are found in the literature. The section closes with a presentation of the various constraints one can apply to the mining process, and of the different ways to count a pattern's occurrences. Section 3.2 is dedicated to pattern mining algorithms, in which we present the two families that can be found in the literature, Apriori-like and pattern growth, and discuss their available implementations. Section 3.3 concludes both this chapter and our review of the state of the art. In it, we consider the notion of Progressive Pattern Mining, and identify two additional challenges.

CHAPTER 2

FROM VISUAL ANALYTICS TO PROGRESSIVE VISUAL ANALYTICS

Associating humans and computers to benefit from the strengths of both has been suggested since the early days of computer science. Some research domains have emerged from this need, such as Visual Analytics, a data analysis paradigm built upon the idea of having a human analyst explore data visualizations. However, when datasets grow too large or when algorithms become too time-consuming, the promises of Visual Analytics are diminished by imposed waiting times for the analyst. To provide an alternative to these problematic cases, the paradigm of Progressive Visual Analytics has emerged, and offers to solve this problem by increasing the interactivity of the analysis process.

This chapter aims at providing an extensive review of the current state of research on Progressive Visual Analytics. Section 2.1 is dedicated to Visual Analytics, and presents the domain and its aspects that lead to the aforementioned limitations. Section 2.2 is then dedicated to the presentation and review of the current research on Progressive Visual Analytics.

1 Visual Analytics

1.1 Origins and early definitions

The first use of the term “Visual Analytics ” is by Wong and Thomas (2004), who define it as follows:

> *A contemporary and proven approach to combine the art of human intuition and the science of mathematical deduction to directly perceive patterns and derive knowledge and insight from them*.

They do so in an introduction to a series of six articles by Elizabeth G. Hetzler and Turner (2004), Teoh, Ma, Wu, and Jankun-Kelly (2004), D. A. Keim, Panse, Sips, and North (2004), Nesbitt and Barrass (2004), Schmidt et al. (2004) and Lee, Girgensohn, and Zhang (2004). Even though these articles do not use the term (except a brief mention by Teoh et al.), Wong and Thomas deem them representative of Visual Analytics as they conceive it. They view Visual

Analytics as a solution to deal with the size, variety and complexity of data that may need to be processed, expecting it to "[enable] detection of the expected and discovery of the unexpected within massive, dynamically changing information spaces".

The following year, Cook and Thomas (2005) published what they call a "research and development agenda for Visual Analytics ", in which they provide a shorter definition of Visual Analytics:

> *The science of analytical reasoning facilitated by interactive visual interfaces*. (Cook and Thomas (2005), page 4)

In this book, they provide a vision for the potential of Visual Analytics, almost exclusively stemming from the September 11, 2001 terrorist attacks and the need for better analysis tools to react during and before a crisis. They summarize the challenge they face as "the *analysis* of overwhelming amounts of disparate, conflicting, and dynamic information to identify and prevent emerging threats, protect our borders, and respond in the event of an attack or other disaster". In order to help tackle this problem, they present a vision of Visual Analytics that combines the following elements:

> - Analytical reasoning techniques *that enable users to obtain deep insights that directly support assessment, planning, and decision making*
> - Visual representations and interaction techniques *that take advantage of the human eye's broad bandwidth pathway into the mind to allow users to see, explore, and understand large amounts of information at once*
> - Data representations and transformations *that convert all types of conflicting and dynamic data in ways that support visualization and analysis*
> - *Techniques to support* production, presentation, and dissemination *of the results of an analysis to communicate information in the appropriate context to a variety of audiences.*
>
> (Cook and Thomas (2005), page 4)

Throughout the book, Cook and Thomas review the state of the art relevant for their vision of Visual Analytics to provide a set of recommendations to guide its development. In particular, they stress the importance of creating tools that will support a collaborative analytic reasoning about complex problems by building upon "existing theoretical foundations of reasoning, sense-making, cognition and perception". They suggest that this should be done through research, by developing "a science of visual representations" and "a science of interactions" to support the analytical reasoning process. In parallel with these scientific recommendations, they also stress the importance of building practical tools to implement these new paradigms and incentivize their insertion into operational environments.

1.2 Current definition

Cook and Thomas (2005) conclude their presentation of Visual Analytics by stressing the fact that the vision they give will most probably evolve in the following years, as Visual Analytics will mature. Indeed, a few years later, D. Keim et al. (2008) provide a more specific definition for Visual Analytics:

> *[A combination of] automated analysis techniques with interactive visualisation for an effective understanding, reasoning and decision making on the basis of very large and complex data sets*. (D. Keim et al. (2010), page 7)

Since the research community has responded to Cook and Thomas (2005)'s "call to action" on Visual Analytics, D. Keim et al. (2008) present a vision of Visual Analytics where the questions of adoption and integration into existing environments are absent. Instead, they are more focused on dealing with what they call the "information overload problem", which can lead to an analyst getting lost in data that may be either irrelevant to her current task, processed in an inappropriate way or presented in an inappropriate way. D. Keim et al. (2008) also keep the idea of Visual Analytics being applicable to a wide array of people (having varied skills, goals and priorities) in completely different circumstances.

Two years later, D. Keim et al. (2010) propose a review of the state of Visual Analytics and its application in various research communities (data management, data mining, spatio-temporal data analysis, system infrastructure, perception, cognition and evaluation). They keep D. Keim et al. (2008)'s definition of Visual Analytics, and identify a set of challenges for Visual Analytics organized into four categories:

Data challenges. Handling large datasets poses challenges with regards to storage, retrieval, transmission, processing time and visualization scalability. The heterogeneous nature of the data increases the complexity of its processing, as does the variation in quality and formalism. Specific cases are also adding to the complexity, such as handling streaming data or semantic information (metadata) and their integration in the analysis process.

User challenges. Minimizing cognitive and perceptual biases requires users to have information about the data provenance, the transformations that were performed during the analysis pipeline, and uncertainties within the data. It is also important that the users are able to understand the simplifications that are done when extracting a model of information from the data. This need for guidance is especially noticeable when said users are not familiar with data analysis. Challenges also arise regarding user collaboration and the degree of interactivity offered by Visual Analytics tools.

Design challenges. Designing Visual Analytics tools should take advantage of the existing knowledge from the domains of visual perception and cognition. Guidance to choose the most suitable existing tools for a given task could also be helpful, as well as having available

test datasets, tools and results of evaluation studies. Although they note the difficulty of the challenge, they also advocate the use of a unified architectural model for Visual Analytics applications.

Technological challenges. The duration of the analysis phase in a data exploration is identified as a challenge, since it tends to be longer than traditional transactions due to the added cost of processing and presenting the data to the analyst. To that end, D. Keim et al. (2010) argue for the need for a "progressive availability" of the results that would give an analyst a rapid overview of what can be expected, allowing her to steer the analysis in particular directions. Another technological challenge is in the capacity to provide multi-scale analysis, along with a methodology for providing basic interactions with data visualizations, such as linking and brushing.

The disparities between Cook and Thomas (2005) and D. Keim et al. (2008)'s presentations of Visual Analytics illustrate the different visions of Europe and the USA with regards to data analysis. Regardless, the broad goals of Visual Analytics remained constant over the years, as evidenced by the similarity between Cook and Thomas (2005)'s recommendations and D. Keim et al. (2010)'s challenges. Visual Analytics aims at allowing one to explore data to extract knowledge from it by offering tools allowing to, in the words of both Cook and Thomas (2005) and D. Keim et al. (2010):

> - *Synthesize information and derive insight from massive, dynamic, ambiguous, and often conflicting data*
> - *Detect the expected and discover the unexpected*
> - *Provide timely, defensible, and understandable assessments*
> - *Communicate these assessments effectively for action.*
>
> (D. Keim et al. (2010), page 7)

1.3 The importance of *insights*

Visual Analytics as a paradigm is supposed to allow an analyst to gain *insights* from the data, as stated in several ways over different works. Card, Mackinlay, and Shneiderman (1999) state that "the purpose of visualization is insight", and Cook and Thomas (2005) describe Visual Analytics's purpose as enabling and discovering insights. The term is widely used, even though none of its existing definitions has been commonly accepted by the information visualization community as a whole (Sacha et al., 2014; Yi, Stasko, & Jacko, 2008). Depending on the focus of the work at hand, an insight can designate a particular moment when one makes sense out of a piece of information, or the newly obtained knowledge from this moment (Chang, Ziemkiewicz, Green, & Ribarsky, 2009). As such, an insight can be new information about

the content of the data, or an hypothesis based on the exploration, that will require further investigation of its veracity.

1.4 Generating knowledge with Visual Analytics

The question of how analysts interact with a system to generate knowledge has been the focus of several research works. A first step has been proposed by van Wijk (2005), in an attempt to assess the value of visualization and how analysts use it to generate knowledge. In his article, he proposes a general model for visualization that considers a *visualization* process that will create a *image* from the combination of *data* and a *specification*. This *image* will then be subject to the analyst's *perception*, which will generate *knowledge* that can either feed the *perception* process or lead to new hypotheses. These hypotheses can then lead to further *exploration* by changing the *specification* before going back into this cycle with a new *image*. The initial model was later augmented to denote the fact that *perception* is an important part of the *exploration* process, and that *exploration* in itself feeds *knowledge* reasoning (T. M. Green, Ribarsky, & Fisher, 2008; Tera Marie Green, Ribarsky, & Fisher, 2009). This whole process has later been referred to by D. Keim et al. (2008) as the *sense-making loop*, illustrated in figure 2.1.

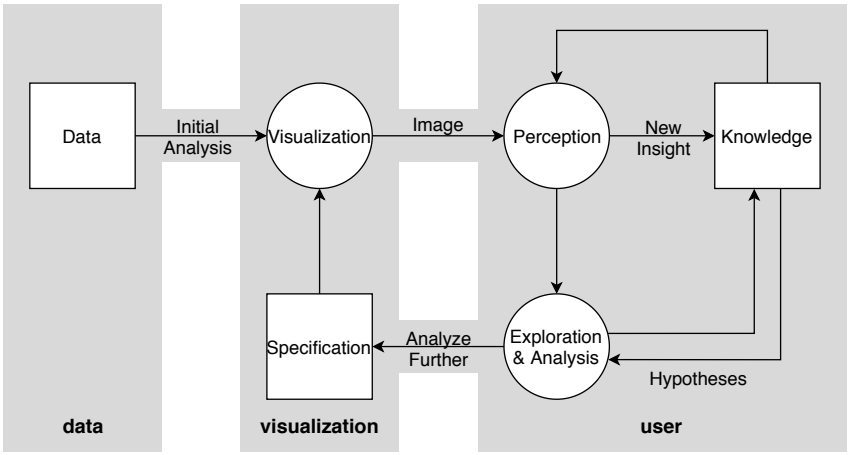


Figure 2.1: Sense-making loop (D. Keim et al., 2008). *Boxes denote containers, while circles are processes that produce an output from an input. The model originates from the works of van Wijk (2005), T. M. Green, Ribarsky, and Fisher (2008) and Tera Marie Green, Ribarsky, and Fisher (2009).*

Visual Analytics differ from the domain of data visualization in that an algorithmic process takes place both before and after the visual exploration process. From this observation, D. A. Keim, Mansmann, Schneidewind, and Ziegler (2006) extended the information seeking mantra from Shneiderman (1996), *"overview first, zoom and filter, details on demand"*, to propose the following Visual Analytics mantra:

Analyse first –
Show the important –
Zoom, Filter and Analyse Further –
Details on demand

This mantra is similar to the one by Shneiderman, with the addition of the two "analyse" steps that represent the addition of an algorithmic process. In accordance with this mantra, D. A. Keim, Mansmann, Schneidewind, Thomas, and Ziegler (2008) proposed a model for the Visual Analytics process, illustrated in figure 2.2. Their model include data transformations that are performed during the pre-processing stage of the analysis, leading to data that can be either visualized or fed to an algorithm to build a model from it. Following this first step, the model created from the data can be visualized, and the insights gained from visual exploration can be used to build a new model. By repeating these interactions, the analyst is then able to generate knowledge, going back to the data transformations if needed.

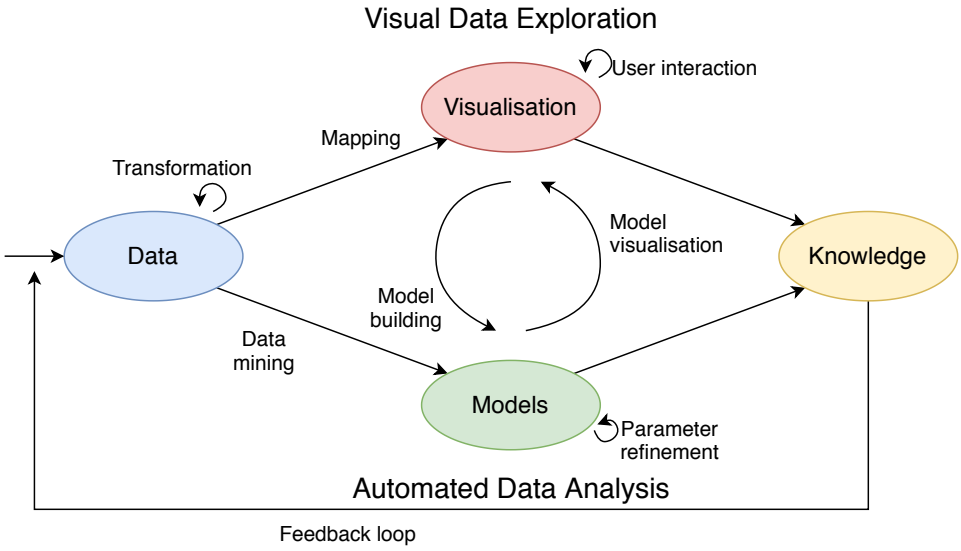


Figure 2.2: Visual Analytics workflow (D. A. Keim, Mansmann, Schneidewind, Thomas, & Ziegler, 2008). *Representation from D. Keim, Kohlhammer, Ellis, and Mansmann (2010).*

Building from D. Keim et al. (2008)'s model, Sacha et al. (2014) proposed a model of knowledge generation in Visual Analytics, illustrated in figure 2.3. In their model, the focus is on the actions a human analyst can make and which stage of D. Keim et al.'s model they impact, rather than the connections between the stages. The *preparation* action refers to data gathering and pre-processing. *Model building* and *visual mapping* actions respectively lead to the creation of a model and visualization from the data, that the analyst can then explore (*model usage* and *manipulation* actions). The *model-vis mapping* action represents actions that map models into visualizations, as well as the opposite one (creating a model from a visualization).

Observations lead to *findings*, which represent any interesting information the analyst obtains while working with the Visual Analytics system. They can lead to the generation of insights or to the continuation of the exploration process.

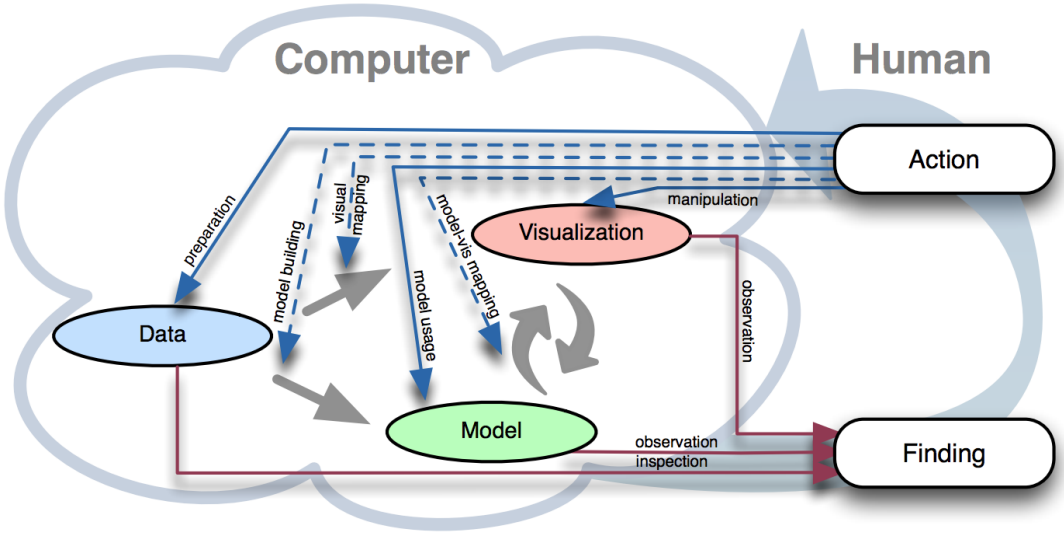


Figure 2.3: Knowledge generation process in Visual Analytics (Sacha et al., 2014). *Human actions are the blue arrows, which can lead to Visual Analytics components (filled arrows) or to the mappings between them (dotted arrows). Red arrows represent the human cognition paths, when generating findings by observing the system.*

When describing their model, Sacha et al. draw a parallel with two other processes described in the literature. The upper part (*Data* and *Visualization*) corresponds to the InfoVis pipeline by Card et al. (1999), while the bottom part (*Data* and *Model*) corresponds to the Knowledge Discovery in Databases process (KDD process, Fayyad, Piatetsky-Shapiro, and Smyth (1996)).

1.5 Modeling the tasks performed with Visual Analytics

Conducting a data analysis is a lengthy process, composed of several steps that an analyst repeats depending on her goals and on the task at hand. To obtain a better understanding of what this process encompasses, several research works have tackled the problem of offering a proper model for data analysis tasks. Here, we present a review of the literature on these models, structured around their focus, be it what the analyst wants to achieve, what actions the analyst performs to achieve their goal, or what data the analyst is working with. The following categories present the models in chronological order. Since they can be quite different in scope and design, the latter models do not necessarily build upon the earlier ones. A final important thing to note is that the following models have not necessarily been proposed with

Visual Analytics in mind. They are however general enough to be applied to this particular context.

1.5.1 Visual Analytics task models focused on analysis goals

Amar and Stasko (2004) propose a high level categorization of analysis tasks composed of two categories, the *rationale-based* and the *worldview-based* tasks. The former contains tasks that aim at supporting complex decision-making, in particular when dealing with uncertainty, and encompasses *exposing uncertainty*, *concretizing relations* and *formulating cause and effect*. The latter contains tasks that aim at supporting learning a domain, and encompasses *determining domain parameters*, *providing multi-level analysis of complex data* and *confirming hypothesis*.

Pirolli and Card (2005) propose a model of analyst's sense making built around six states that a dataset can be in, as illustrated in figure 2.4. In order of increasing structure, the first state is called "external data sources", and refers to the raw data. The "shoebox" is a subset of this raw data, from which one can extract parts of the items to create an "evidence file". "Schema" is a re-structuration of this information that is more suited to obtain conclusions, which are then supported with arguments to form the "hypotheses". The final state is the "presentation" of the knowledge gained from this process.

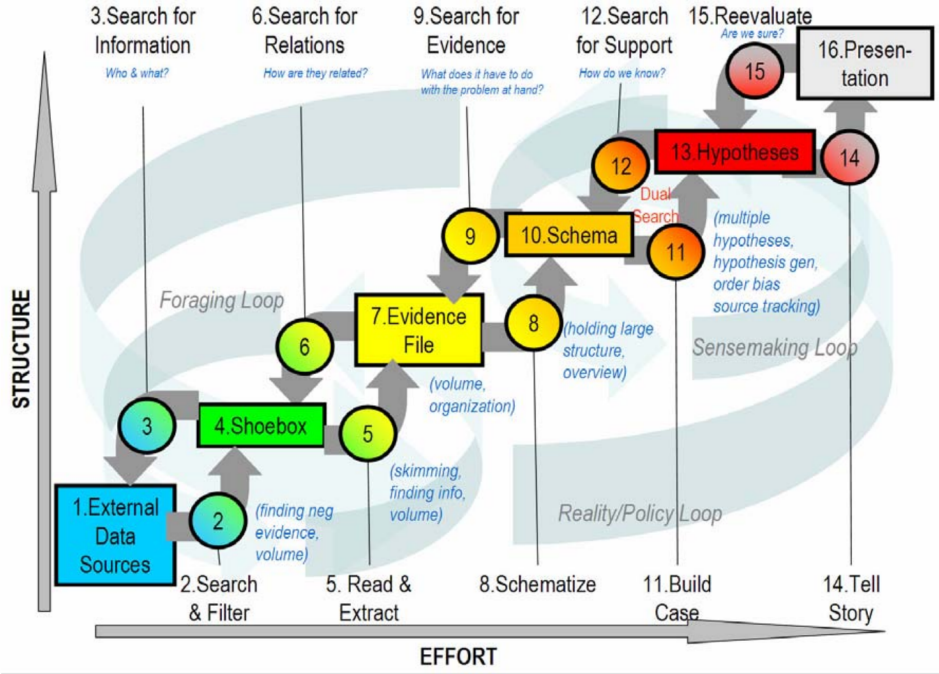


Figure 2.4: The model from Pirolli and Card (2005). *Illustration by Pirolli and Card (2005).*

Their task model consists of the ten actions one can perform to transition between two of the aforementioned states. They propose a bottom-up process (from “external data sources” to “presentation”) consisting of “search and filter”, “read and extract”, “schematize”, “build case” then “tell story”, as well as a top-down process (from “presentation” to “external data sources”) consisting of “reevaluate”, “search for support”, “search for evidence”, “search for relations” then “search for information”. Although these two processes are distinguished, the authors note that in practice, analysts use both in an opportunistic mix.

For Ward, Grinstein, and Keim (2010), analysis tasks form three high level categories. When performing *exploration tasks*, analysts aim at formulating hypotheses, while *confirmation tasks* consist of verifying these hypotheses. The third category, called *presentation tasks*, encompasses the work on data realized with the aim of communicating what has been found through the other two types of tasks.

1.5.2 Task models focused on actions on data

MacEachren (1995) proposed a model composed of low-level tasks designed for temporal data, in order to cover all the possible questions an analyst can ask herself during her work. He provides the following list of questions, while noting that it is by no means exhaustive:

- Does a given element exist in the data?
- When does an element appear in the data?
- What time does an element span?
- Does an element present a given temporal pattern?
- What is the change frequency of an element?
- What is the sequence or order of appearance of some elements?
- Is there a synchronicity between elements?

Based on his mantra of “Overview first, zoom and filter, then details on demand”, Shneiderman (1996) proposes to group visual information seeking tasks into the following seven categories:

- *Overview tasks*, that provide a global view of the data at hand
- *Zoom tasks*, to zoom on elements of interest
- *Filter tasks*, to reduce data to elements of interest
- *Details on demand tasks*, to obtain details on one or more elements
- *Relating tasks*, to visualize relations between elements
- *History tasks*, to navigate the history of actions performed during the analysis
- *Extraction tasks*, to obtain subparts of the data or parameters used

Rather than using Shneiderman's visualization mantra, Yi, ah Kang, Stasko, and Jacko (2007) propose a classification of analysis tasks built from the intents of an analyst during her work. Doing so, they identify seven categories of tasks:

- *Selection tasks*, where the analyst marks some element as interesting
- *Exploration tasks*, where the analyst changes her focus from some elements onto others
- *Reconfiguration tasks*, where the analyst changes the spatial arrangement of the data to observe it under a different angle
- *Encoding tasks*, where the analyst changes the representations used
- *Abstract/Elaborate tasks*, where the analyst looks at the data with respectively less or more details
- *Filtering tasks*, where the analyst observe data elements under certain conditions
- *Connection tasks*, where the analyst looks at the relations between elements

The authors note that their classification is not exhaustive, citing actions such as "undo/redo" or "changing a system's configuration".

1.5.3 Task models focused on data

Andrienko and Andrienko (2006) propose a general high-level model designed to express the tasks an analyst may perform on temporal data. Their model relies on a formal description of the data to be explored, composed of two sets of elements: the *referrers*, which can be seen as the dimensions of the data, and the *attributes* which contain the values found either directly from the data or derived from it. By specifying the values of the *referrers*, one can obtain the associated *attributes*' values through what can be seen as a function from the domain of the *referrers* to the domain of the *attributes*. Similarly, one can perform the inverse operation by specifying the values of some attributes to retrieve the associated *referrers*' values.

Based on this data model, Andrienko and Andrienko identify 11 task categories, as illustrated in figure 2.5. *Elementary tasks* refer to tasks targeting a single data element, or a set of element where each one is considered individually. They oppose these tasks to the *synoptic tasks*, which target sets of data elements that are considered as a whole. In both categories, the *lookup* and *comparison* tasks can be either *direct* or *inverse*, the difference between these notions being in the subject of the task. *Direct tasks* specify the *referrers*' values to look for the associated *attribute* values, while *inverse tasks* specify the *attribute* values in order to find the associated *referrers*' values. While this is only an overview of their model, more details are provided in section 5.1.

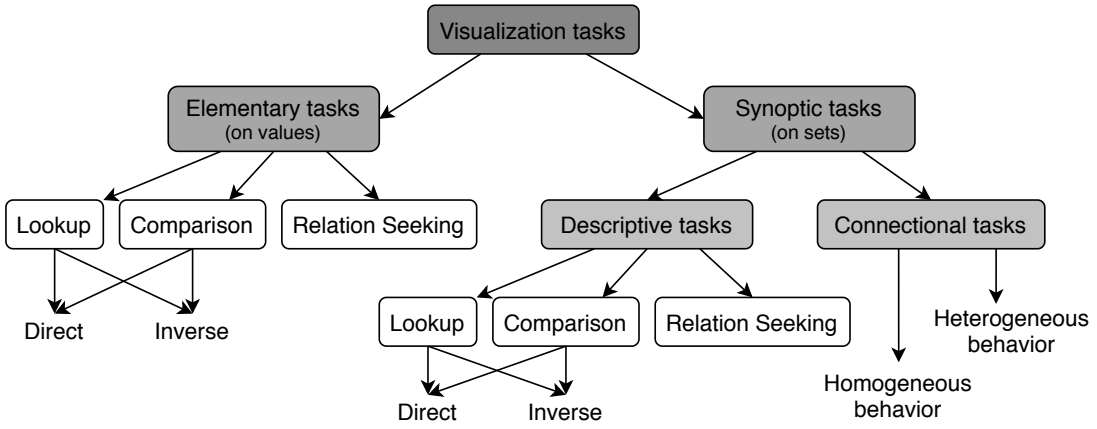


Figure 2.5: The task model from Andrienko and Andrienko (2006). *Illustration inspired by Aigner, Miksch, Schumann, and Tominski (2011).*

1.6 Examples of existing Visual Analytics tools

Visual Analytics has been applied to a number of domains, data and use cases. Offering a detailed review of existing Visual Analytics systems would be too long, and is not part of the scope of our works. As such, while far from exhaustive, the following list aims at offering an overview of the variety in existing Visual Analytics systems, in terms of algorithmic techniques used and of visualizations offered. No other criterion has been taken into account when choosing which systems would be included.

1.6.1 Interactive Parallel Bar Charts (Chittaro, Combi, & Trapasso, 2003)

Interactive Parallel Bar Charts (IPBC) allows the exploration and comparison of a collection of time-series, initially designed to explore medical data. It is illustrated in figure 2.6.

Visualization – A bar chart representation of the time-series is provided. It can be switched between 2D and 3D, and supports the flattening of subparts of the chart to avoid occlusion problems in 3D mode. Color is used to highlight query results, and water is displayed to represent a user-specified threshold, highlighting the emerged bars.

Interactions – The analyst can switch between different modes to explore, transform and manipulate the bar chart. She can also aggregate the data by specifying the duration represented in a bar. The analyst can specify range of values of interest, to formulate a query in the chart, or specify a threshold to highlight the data points above it. A pattern matching can also be requested with an SQL-like syntax.

Algorithms – Algorithms in IPBC support the construction of the bar chart from the initial data, as well as providing results for the user queries. Additional interactivity is provided by the ability to manipulate the display, by rotating and translating the visualization.

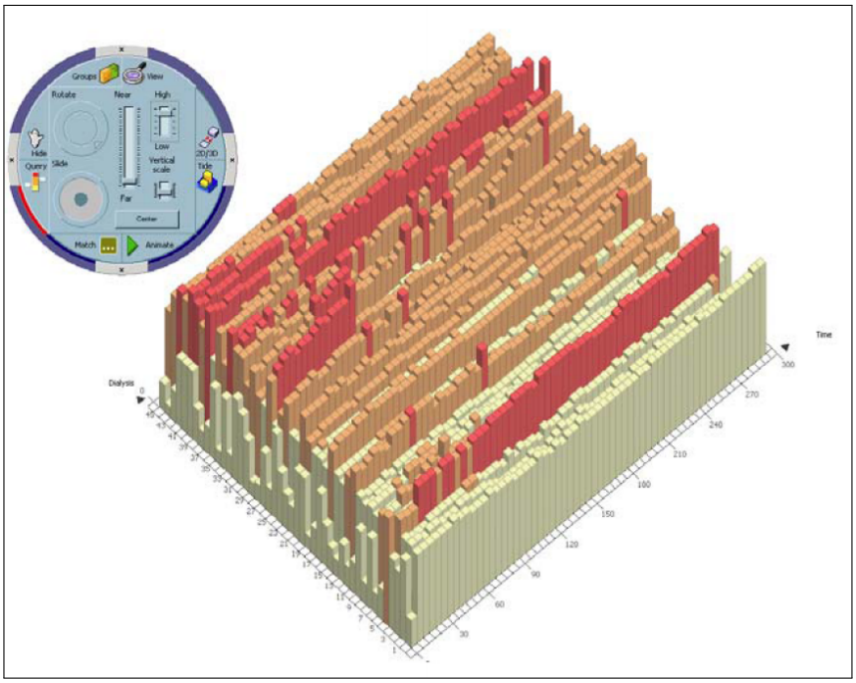


Figure 2.6: Interactive Parallel Bar Charts (Chittaro, Combi, & Trapasso, 2003).

1.6.2 VizTree (Lin, Keogh, Lonardi, Lankford, & Nystrom, 2004)

With VizTree, Lin et al. provide a system to visually explore time series in order to discover underlying patterns. The system was designed to address some challenges faced by the US Department of Defense when launching space vehicles, by allowing to simultaneously visualize the global and local structures of the data. VizTree allows one to discover frequent patterns, perform anomaly detection and query specific content within the explored data. It is illustrated in figure 2.7.

Visualization – The time series are equally divided in a number of sub-sequences, as is the range of the values present in the series. A letter refers to each value-range, and sub-sequences are then mapped to the letter representative of their mean value, thus forming a string representation of the time-series. A tree representation aggregates all the string-sequences, with the width of a branch representing the frequency of the corresponding pattern in the data. Each level of the tree is a segment of the original sequence. All possible combinations are displayed, even if they are not found in the data. Color is used to distinguished found patterns (red branches) from possible ones (grey branches). When a node is selected by the analyst, its sub-tree is displayed on the right, along with the corresponding sub-series, that are also highlighted in the top view of the whole time-series. When selecting two nodes, a diff-tree is displayed, showing the differences between them. These differences are also highlighted in the top view.

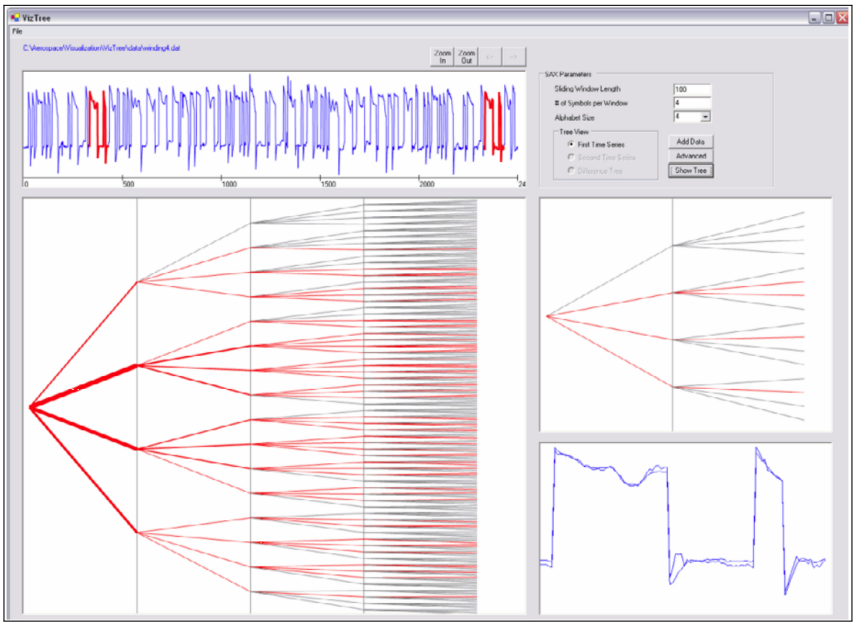


Figure 2.7: VizTree (Lin, Keogh, Lonardi, Lankford, & Nystrom, 2004).

Interactions – The analyst can select a node in the tree to zoom-in on sub-trees of interest and highlight them in the visualization. Two nodes can be selected to compare their differences. A few parameters are available, to select the number of sub-sequences and the number of value categories to create the string representations.

Algorithms – Besides converting the initial time-series to a string, algorithmic tools allow the retrieval of sub-sequences in the original time-series, and the comparison of two sub-sequences.

1.6.3 Lifelines2 (Wang et al., 2009)

LifeLines2 allows one to explore temporal categorical data, namely medical records of patients. The focus is on exploring and comparing multiple patient records at the same time, as opposed to its predecessor LifeLines (Plaisant et al., 1998) that was targeting a single record at a time. It is illustrated in figure 2.8.

Visualization – The visualization provided is a timeline-based representation of the medical records occupying most of the interface. It allows the analyst to visually compare side-by-side records to look for patterns or singularities in the data. A temporal summary of the data is also provided, where the analyst can decide which criterion is used for the summary (events, records, events per record). A comparison view is available, to display bar charts highlighting the differences between different records.

Interactions – The analyst can select events that will be used to align the different records

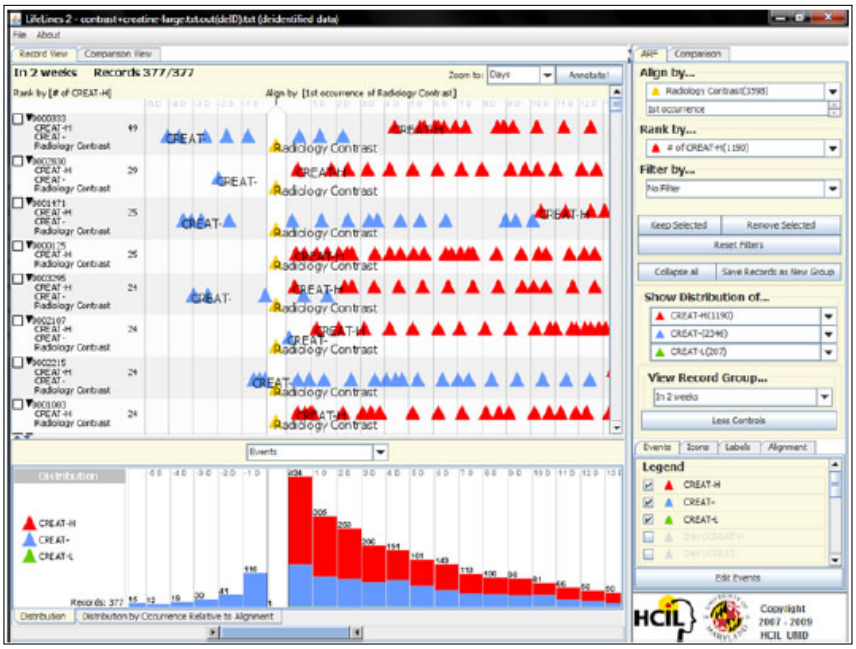


Figure 2.8: LifeLines2 (Wang et al., 2009).

in varying ways (n^{th} occurrence from the start or the end, or on occurrences of the same event). The system support searching for a target event type, and the records can be split into sub-records if needed. A time-interval can be selected by dragging over a visualization, which will automatically focus the other ones on the selected period. Additionally, the event types can be ranked and filtered, and various options are provided to tweak the records' display.

Algorithms – The algorithms provided deal with the presentation of the data, to filter it, compare selected records, build temporal distributions and adapt the visualization to the analyst's needs.

As noted at the start of this subsection, these few examples of existing Visual Analytics systems are far from exhaustive. They however illustrate how Visual Analytics as a data analysis paradigm has been applied to a wide range of tasks, using a variety of visualization and algorithmic techniques. As a result, the role of the algorithms involved varies from one system to another, which raises the question of how much of the analysis process should still be performed by the human analyst.

1.7 Degree of automation in Visual Analytics systems

By definition, Visual Analytics systems provide both data visualization and algorithmic processes to assist an analyst. However, the balance between these two components is not spec-

ified, and varies greatly from system to system. This subsection presents existing works that have studied the importance of both visualization and algorithms in existing Visual Analytics applications.

While Visual Analytics is at the intersection of Automatic Analysis and Visualization, D. A. Keim, Mansmann, and Thomas (2010) argue that Visual Analytics is not always a good solution for all problems from these domains. In the absence of a clear way to tell if a given problem should be handled with Visual Analytics tools, they propose a theoretical representation of the potential of Visual Analytics, as illustrated in figure 2.9. They conclude by stressing the interest of "[developing] methods for determining the optimal combination of visualization and automated analysis methods to solve different classes of problems by taking into consideration the user, the task and the characteristics of the data sets".

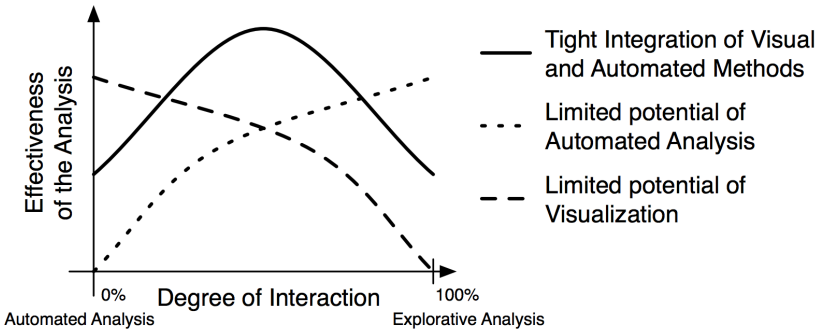


Figure 2.9: Potential of Visual Analytics (D. A. Keim, Mansmann, & Thomas, 2010).

The same year, Bertini and Lalanne (2010) proposed a review of 48 Visual Analytics systems, in which they identify three groups of systems:

- The Computationally enhanced Visualizations (V++), relying mostly of visualizations but supported by algorithmic processes
- The Visually enhanced Mining systems (M++), relying mostly on algorithmic processing but offering some visualization capabilities to deal with the results of the algorithm
- The Integrated Visualization and Mining systems (VM), where the balance of visualization and algorithmic processes does not allow one to evidence a clear dominance of one or the other.

Bertini and Lalanne conclude from their review that "the most interesting and promising direction for future research is to achieve a full mixed-initiative KDD process where the human and the machine cooperate at the same level". This paints VM systems as the most desirable category, even though it is the least populated (8 systems among 48). When studying the other

two categories, they observe that both use algorithms at the early stage of the analysis, while visualization is present in the final stages.

From their review of existing systems, they suggest several ways in which V++ and M++ systems could be improved, to bridge the gap towards the VM category. With regards to V++ systems, they first suggest the idea of Visual Model Building, in that visualizations (and the ways in which the user interacts with it) can be used as input for a model-building algorithm. Their second suggestion is to use algorithmic techniques to verify and refine patterns found in the data visualizations, in order to replicate the practice of communicating a level of trustworthiness of the extracted knowledge, common in automatic data mining. Finally, their last suggestion for V++ systems is to provide tools to deal with tasks of predictive modeling, that are usually not addressed by information visualization tools. With regards to M++ systems, their first suggestion is that they should leverage visualization to offer better representations of the parameter space of the algorithmic processes, and of the alternatives that different parameter values can offer. Their second suggestion is that visualization should be used as a way to bridge the semantic gap that can exist between the original data and the models that are built from them.

While they deliberately leave systems fully dedicated to either data visualization or algorithmic processing out of their study, one could easily extend their work to such systems. This would provide a categorization of existing systems as done in figure 2.10, with an axis having at one end *Data visualization*, with no algorithm besides the one responsible for building the visualization. An illustration could be the Perspective Wall visualization, by Mackinlay, Robertson, and Card (1991). At the other end is *Algorithmic processing*, with no visualization to support the exploration of the algorithm's result. An illustration could be the R language and environment, which provides text outputs after an analysis. In the middle, we can find Visual Analytics systems from Bertini and Lalanne's VM category, while the other categories are on the left and right side of the axis. A system's distance from the center will depend on its balance between visualization and algorithmic capabilities.

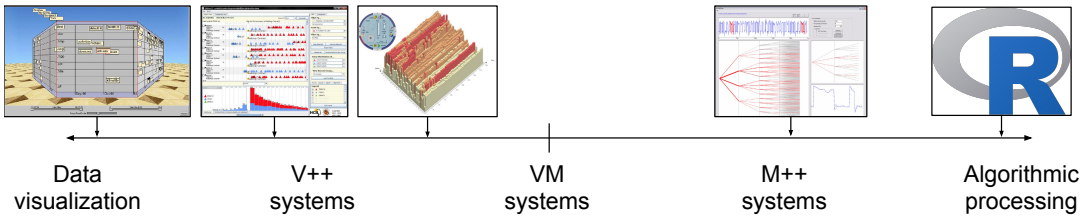


Figure 2.10: Visual representation of the categories from Bertini and Lalanne (2010). *We added two categories fully dedicated to either visualization or algorithmic processing. The systems presented in the previous subsection have been placed on this axis. The position of a system on the axis provides a broad estimate of its features.*

1.8 Towards Progressive Visual Analytics

Despite the advantages it offers, Visual Analytics is limited in that the analyst needs to have the visualization ready before he can explore the underlying data. As explained in the technological challenges for Visual Analytics by D. Keim et al. (2010), when dealing with large dataset or using time-consuming algorithms, the visualization might not be immediately available, which forces the analyst to wait before starting her work. This is obviously problematic in the case of long waiting time (hours or more), but even small delays of a few seconds can harm the analysis process if encountered repeatedly. In some cases, this can be solved by providing more memory or processing power to run the Visual Analytics system, but other paradigms have been suggested to tackle this challenge, such as Progressive Visual Analytics.

2 Progressive Visual Analytics

This section presents the paradigm of Progressive Visual Analytics, which brings an alternative to the aforementioned limitations of Visual Analytics. After a chronological presentation of the evolution of Progressive Visual Analytics' definition and its difference with closely related domains, we present some examples of existing Progressive Visual Analytics systems. A presentation of the works that have studied the design of such systems follows, before concluding the chapter.

2.1 Definition and origin of the progressive paradigm

Stolper et al. (2014) first defined Progressive Visual Analytics as follows:

> *Methods to avoid [...] speed bumps, where analysts can begin making decisions immediately [by] visualizing partial results [that] provide early, meaningful clues [...] without waiting for long-running analytics to terminate*.

This definition stems from existing work, notably in the domains of incremental visualization (Angelini & Santucci, 2013; Fisher, Popov, Drucker, & schraefel m.c., 2012) and dynamic visualization (E. G. Hetzler, Crow, Payne, & Turner, 2005). Stolper et al. aimed at unifying and extending these domains to offer an alternative to what they call *batch Visual Analytics*, where analysts need to wait for the end of a computation before visualizing the results (figure 2.11, top). They argue that having intermediate results allows an analyst to detect and cancel an analysis that is unlikely to generate interesting results, thus limiting wasting analysis and computation time. In addition, having partial results allows the analyst to start exploring the data right from the start, thus reducing or even eliminating any waiting time (figure 2.11, bottom).

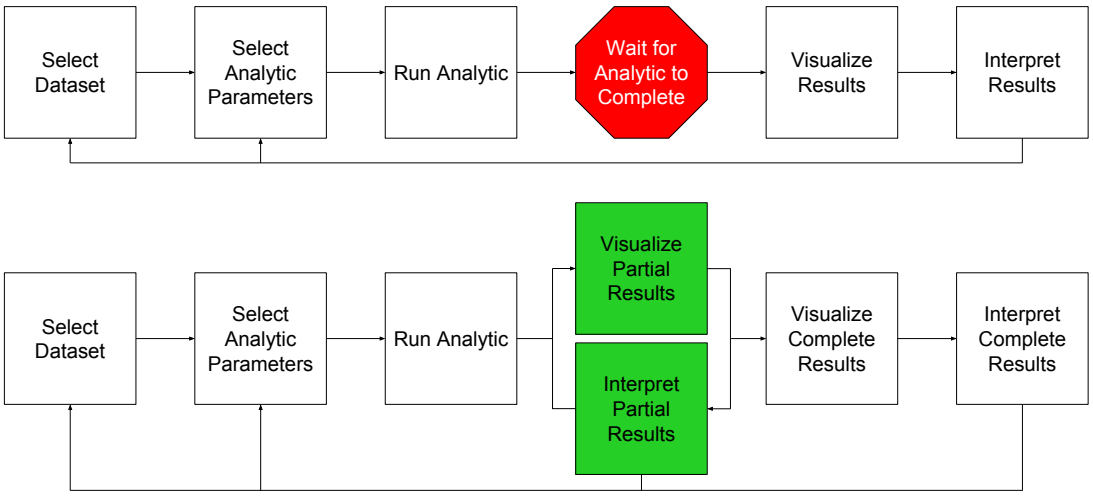


Figure 2.11: Batch Visual Analytics (top) vs Progressive Visual Analytics (bottom) (Stolper, Perer, & Gotz, 2014).

Along with the previous definition, Stolper et al. provide a list of 7 design goals for Progressive Visual Analytics systems:

- Design analytics component to:

  1. Provide increasingly meaningful partial results as the algorithm executes
  2. Allow users to focus the algorithm to subspaces of interest
  3. Allow users to ignore irrelevant subspaces

- Design visualizations to:

  4. Minimize distractions by not changing views excessively
  5. Provide cues to indicate where new results have been found by analytics
  6. Support an on-demand refresh when analysts are ready to explore the latest results
  7. Provide an interface for users to specify where analytics should focus, as well as the portions of the problem space that should be ignored

Though Stolper et al.'s definition was the first to provide a precise definition and a name for the principles behind Progressive Visual Analytics, some work predates this definition while complying with it. Most importantly, in their technological challenges for Visual Analytics, D. Keim et al. (2010) write the following:

> *One [challenge] is in relation to the duration of the analysis phase, which tends to be much longer than traditional transactions dealt with by a standard database management system. Therefore, methods are required not only to support long commit phases, but also to furnish partial results from the analysis. Providing this*

> *"progressive analysis" would give the analyst a rapid overview and hence, a basis for steering the analysis in a particular direction, from which details could be sought*.
> (D. Keim et al. (2010), page 147)

Implementations complying with Stolper et al.'s design goals can also be found predating their definition of Progressive Visual Analytics, such as Williams and Munzner (2004)'s Progressive MDS. This will be presented in more detail in subsection 2.2.3.

With regards to the notion of "progressive algorithms", none of the previously mentioned publications on Progressive Visual Analytics provide a clear and explicit definition. As such, the implicit definition of a progressive algorithm seems to be "an algorithm that complies with the concepts of Progressive Visual Analytics "; *i.e.* that provides intermediate results of increasing quality, while allowing the analyst to interact with its execution.

While these articles focus on the benefits of Progressive Visual Analytics, some work have been focused on highlighting the risks that the progressive paradigm might carry. Angelini and Santucci (2017) in particular stress that the important role played by partial results needs to be taken into account to prevent analysts to misinterpret an Progressive Visual Analytics system's output.

On October 2018, Progressive Visual Analytics was the focus of the Dagstuhl Seminar 18411, entitled "Progressive Data Analysis and Visualization". Its focus was on bringing together researchers from the database, visualization and machine learning communities, in order to discuss the challenges associated with the notion of progressiveness, that each of these communities had started to address in their own terms. The seminar's report (Fekete, Fisher, Nandi, & Sedlmair, 2019) as well as the articles that stem from it (Micallef et al., 2019; Turkay et al., 2018) are discussed in our conclusion (chapter 8) rather than here.

2.2 Progressive algorithms compared to related algorithm types

Due to their goal of offering an answer to waiting times imposed to the analyst by an algorithmic computation, it might seem relevant to assimilate progressive algorithms to other types of algorithms having a similar aim, such as anytime, online or incremental algorithms. This subsection is focused on how progressive algorithms are in fact different from these, and should not be confused with them (Fekete et al., 2019; Fekete & Primet, 2016).

2.2.1 Progressive versus anytime algorithms

Anytime algorithms (Dean & Boddy, 1988) can be described as algorithms that enable the trading of result precision for computation time. This is achieved through four aspects:

- Allow the output of a result upon interruption of the execution at any time;
- Provide results whose quality can be measured;
- The more processing time, the better the results;
- Predictability of the results quality, given then data and the allowed computation time.

From this, we can see that anytime algorithms differ from progressive ones in that they need to be interrupted to provide their result. Later, Zilberstein (1996) proposed that anytime algorithms could regularly output results during their execution, by regularly pausing the computation process. Based on the current output, the analyst could then choose between three options: 1/ resume until the next pause, 2/ resume until completion of the computation, and 3/ terminate the computation. However, the effectiveness of this proposition relies on the ability to pause and resume the computation with reduced overhead.

2.2.2 Progressive versus online algorithms

Online algorithms are algorithms that need to produce an output to answer requests using input data that is not entirely available at the time of their execution (Albers (2003), Borodin and El-Yaniv (2005) (Preface)). Since future input may be relevant to the current computation, this means that online algorithms must work with incomplete knowledge.

Should an online algorithm be ran several times to perform the same request, one could consider the earlier output to be intermediate results for the subsequent ones. From this, we can see that *online* and *progressive algorithms* both output intermediate results. However, *online algorithms* do not provide any interaction with their process, while providing a different result from what a non-online algorithm performing the same task would provide. Another difference between the two is that rerunning an online algorithm brings additional data to be processed, while progressive algorithms are supposed to work on the entirety of a dataset right from the start.

2.2.3 Progressive versus incremental algorithms

Incremental algorithms are a family of algorithms designed to process large datasets by chunks of increasing size. While each chunk includes the previous one, the algorithm only processes the new data. The output is then combined with the information extracted from the previous

chunk, and the next increment can occur. Even though these algorithms inspired Stolper et al. (2014) for their definition of Progressive Visual Analytics, and can be a basis for a progressive algorithm, they are not inherently progressive in nature. The main reason for this distinction is that processing the data in chunk does not imply that feedback is provided before the end of the process, or that interacting with the process is possible.

2.3 Existing implementations of Progressive Visual Analytics systems

Even though the paradigm of Progressive Visual Analytics is still young, a few implementations have been proposed in the literature. They vary in the way they approach Progressive Visual Analytics and in the use case they address, but offer a basis future systems will be able to build upon. As we will see in this subsection, these Progressive Visual Analytics systems all propose visualizations that present the analyst with what is essentially incremental visualization of results as they are being built. Interactions in this case are classical, such as zooming, filtering, ordering, highlighting, etc. Additional control may be available, such as tuning density map's rendering parameters (Pezzotti et al., 2017; Schulz, Angelini, Santucci, & Schumann, 2016) or altering the visualization's evolution speed (Badam, Elmqvist, & Fekete, 2017; Schulz et al., 2016). Some systems go further by proposing interactions between the analyst and the running algorithm. In addition to the following systems, one can also mention Fekete (2015)'s ProgressiVis, a python toolkit that can be used to implement progressive algorithms.

2.3.1 MDSteer (Williams & Munzner, 2004)

Williams and Munzner propose MDSteer, a steerable system that allows the user to progressively guide the MDS layout process. Multidimensional Scaling (MDS) is a general term that refers to any approach that attempts to represent multi-dimensional data in a lower dimension space while preserving inter-point distance as best as possible. A common application of such technique is to project N-dimensional data ($N > 2$) into a 2-dimensional space, which enables a scatterplot representation of the data. MDSteer is illustrated in figure 2.12.

Visualization – The visualization in MDSteer consists of the scatterplot representation of the data points' projection in the low-dimensional space. During the computation, the 2D space is divided in several regions (boxes in figure 2.12). The color of a region's outline indicates whether all of the data points that are likely to be projected into it are displayed (black) or not (red).

Algorithmic processes – The only algorithm involved is Progressive MDS, which alternates between two phases. The *binning* phase involves dividing the low-dimensional space into regions and associating them to the data points they contain. The *layout* phase adds a number of points to the low-dimension space and updates the representation. The *binning* phase is triggered after a number of *layouts* have been performed.

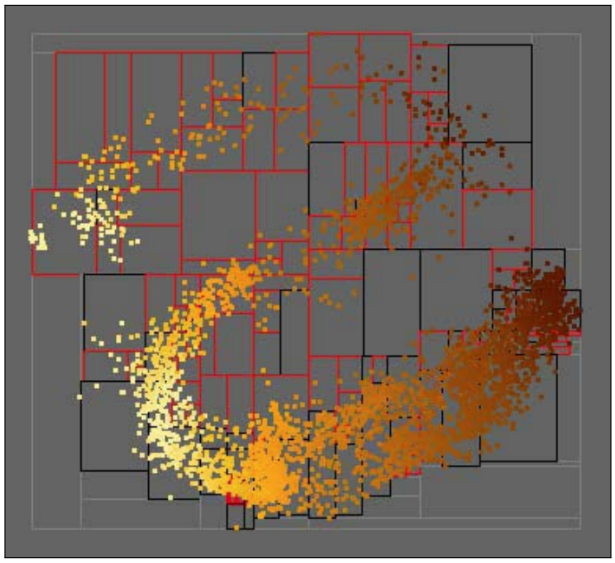


Figure 2.12: Progressive MDS (Williams & Munzner, 2004).

Interaction with the algorithm – The analyst is able to interact with Progressive MDS by selecting regions of the low-dimension space. This impacts the selection of random points that will be added to the representation during the next *layout* phase, ensuring that it selects points that are likely to be projected into the selected regions (based on their high-dimensional similarity with already placed points).

2.3.2 Progressive Insights (Stolper et al., 2014)

Stolper et al. implemented a system for the exploration of medical data using sequential pattern mining, built around a progressive version of an existing algorithm. Their system displays the patterns both in a hierarchical list and in graphical representations that the analyst can interact with. Progressive Insights is illustrated in figure 2.13.

Visualization – Progressive Insights' interface offers three main data visualization views. First, a list view of the top patterns, according to several ranking measures. Two instances of this view are present, allowing the analyst to compare different ranking measures (**C** in figure 2.13). On the left side, each pattern in the list is paired with a bar of the selected measure's histogram. A mini-map of the complete pattern list is provided on the right. In order to reduce visual noise, newly discovered patterns are not added to the list, but are instead represented as purple horizontal marks within the list. These can be clicked to update the list, and are reflected on the mini-map.

The second view is a scatterplot (**A** in figure 2.13) whose axes are the two metrics selected in the previously described list views. The top-n patterns according to a third user-selected metric are represented using orange dots. When a pattern is selected by a click (either in the

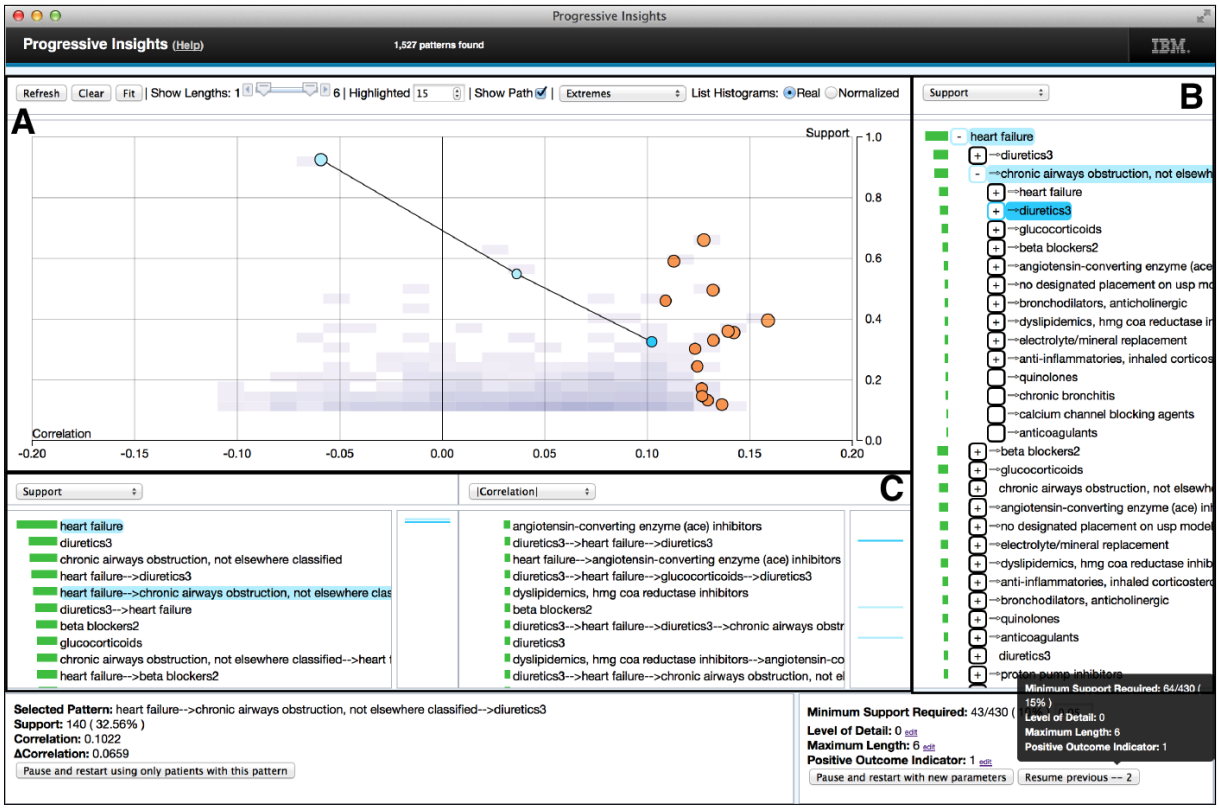


Figure 2.13: Progressive Insights (Stolper, Perer, & Gotz, 2014).

lists or in the scatter plot), its prefixes and the pattern itself are highlighted in blue. The analyst can then observe the pattern-to-prefix relationship within the current metric space by displaying a black line that connects the selected pattern to its ancestors. Rather than abruptly removing the points associated to pattern that fall out of the top-n, these are progressively transitioned to smaller, white-filled dots. A "Clear" button is available in the top-left, should the analyst want to get rid of these. In addition to the scatterplot for the top patterns, the entirety of the mining output is displayed in the background using a heat map of the pattern density within the metric space. Cells of the heat map can be hovered over to see detailed information about their content, and double-clicking a cell adds its content to the scatterplot, in addition to the top-n patterns.

The last view is a tree view (**B** in figure 2.13), which provides a collapsible hierarchical view of the patterns. It allows analysts to direct the ongoing computation, by steering towards specific pattern prefixes or by marking some patterns as undesirable, thus preventing the search for patterns that extend them. Similarly to the list view, a bar chart is displayed on the left of the list, representing the ordering according to a user-selected metric.

All three views are linked, allowing the analyst to select a pattern in any view to highlight it

in all of them.

Algorithmic processes – The progressive algorithm in Progressive Insights is adapted from the SPAM algorithm (Ayres, Flannick, Gehrke, & Yiu, 2002), and outputs frequent patterns as soon as they are discovered. Their modifications to SPAM include making the algorithm use a breadth-first search strategy (see subsection 3.2.1), effectively prioritizing the shorter patterns in the first steps of the computation.

Interaction with the algorithm – During the algorithm computation, the analyst is able to specify (by clicking in the pattern list on the right) that a given pattern should be prioritized. This forces the algorithm to look for new frequent patterns having the selected pattern as prefix. In the same way, a pattern can be forbidden, preventing the search of new patterns having it as a prefix. Either during or after the computation, the analyst can restart the algorithm over a subset of the data, or with different parameter values, the system keeping a record of the previously selected sets of parameters.

2.3.3 Similarity search (Schulz et al., 2016)

Schulz et al. propose a system for similarity search in large datasets, illustrated in their article with a dataset containing all car crashes in the USA between 2001 and 2009. They use an algorithm that incrementally finds the data points similar to a target point defined by the analyst. Their system is illustrated in figure 2.14.

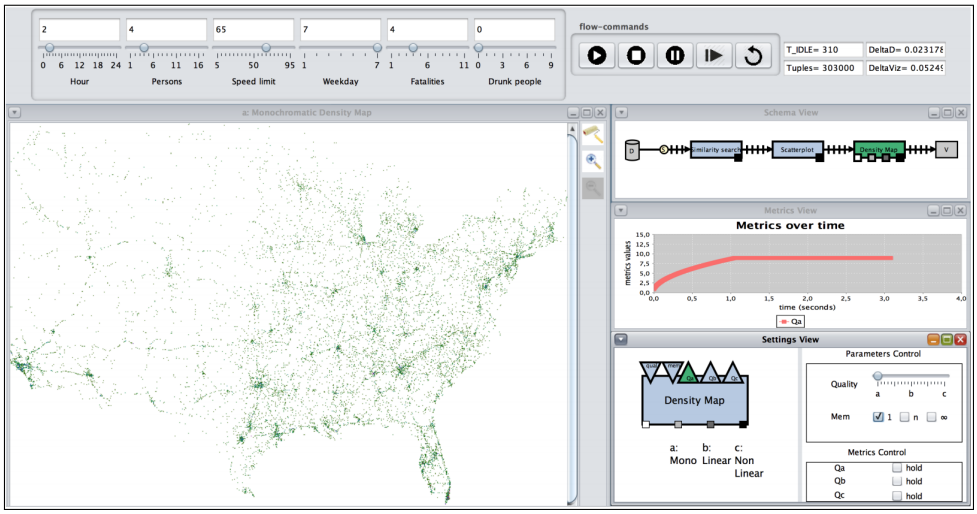


Figure 2.14: Similarity search (Schulz, Angelini, Santucci, & Schumann, 2016).

Visualization – The main visualization is a geospatial representation of the data points output by the algorithm. This visualization is incrementally updated as the algorithm discovers results. Representations of the process from data to visual representation are displayed on the side to provide insights about the data processing pipeline. The top part of the interface offers

sliders for the attributes that can be used to define the search target, as well as controls over the computation (pause, resume, restart. . .).

Algorithmic processes – The algorithm takes a target defined by a set of attributes, allowing one to describe a configuration that may not be present in the data, and searches for similar data points. This is done by iteratively selecting a sample of the unexplored data, and displaying the data points that are similar to the given target.

Interaction with the algorithm – The analyst can change the search target, and can pause, resume and stop the algorithm during its execution, as well as restart the procedure with new parameters.

2.3.4 PIVE (Kim et al., 2017)

PIVE (**P**er-**I**teration **V**isualization **E**nvironment) is a system designed to work with algorithmic processes that iteratively refine their output. By displaying the output of each iteration, the system allows one to perform visual data exploration while interacting with the running process. PIVE is described as a generic platform able to adapt to various algorithms, that Kim et al. illustrate with dimension-reduction and clustering techniques. PIVE is illustrated in figure 2.15.

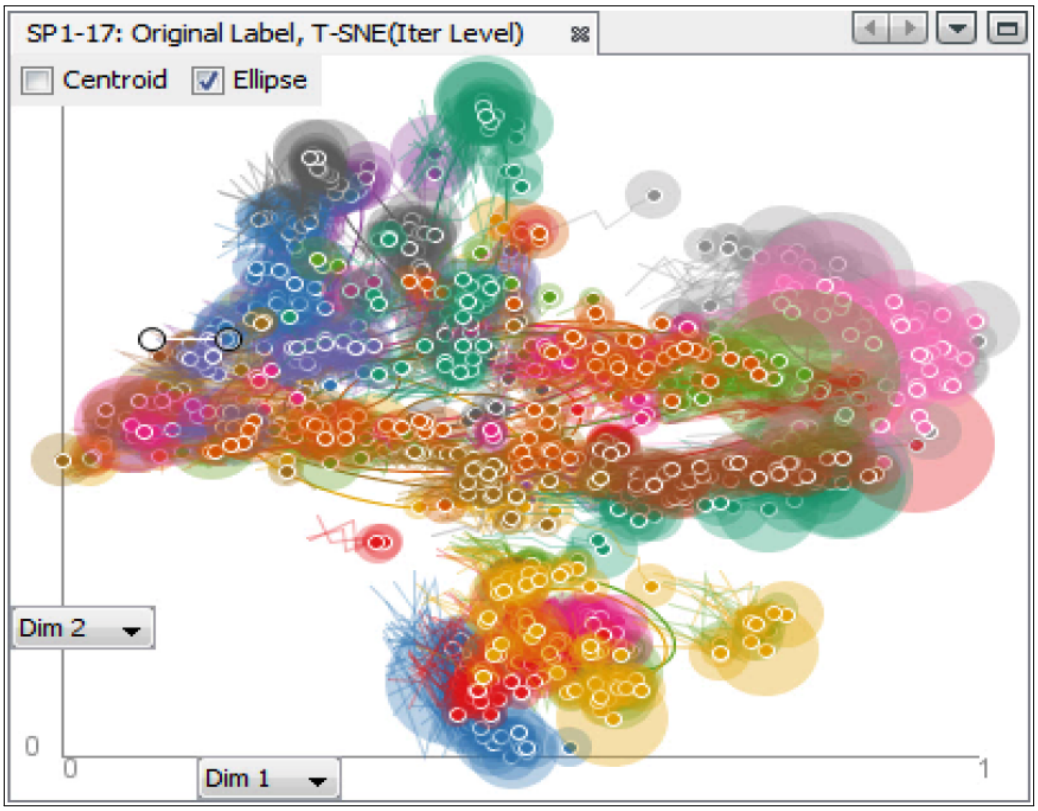


Figure 2.15: PIVE (Kim et al., 2017).

Visualization – While the visualization offered could vary depending on the algorithmic technique in use, there should be a visual display of an iteration's output. In the illustrations provided by the authors it consists of a 2D scatterplot, generated by dimension reduction and clustering. Over successive iterations, the display is updated to reflect the most up to date results.

Algorithmic processes – In their article, Kim et al. use several dimension reduction algorithms (t-distributed stochastic neighborhood embedding (tSNE), multidimensional scaling (MDS)) and clustering techniques (k-means, latent Dirichlet allocation). The algorithms are modified to output the result of each of iteration and allow the analyst to interact with the outputs.

Interaction with the algorithm – The analyst can manually alter MDS and t-SNE algorithms' outputs (*i.e.* change the position of points in the low-dimensional space), either permanently or temporarily, to guide the remaining computation until convergence is reached. Merging or splitting selected k-means clusters permanently or temporarily is also possible, which again provides guidance for the system.

2.3.5 Approximated tSNE (Pezzotti et al., 2017)

Pezzotti et al. (2017) propose a system built around their progressive adaptation of the tSNE algorithm. Their system is focused on the visual analysis of high-dimensional data, with the ability for the analyst to interact with the computation.

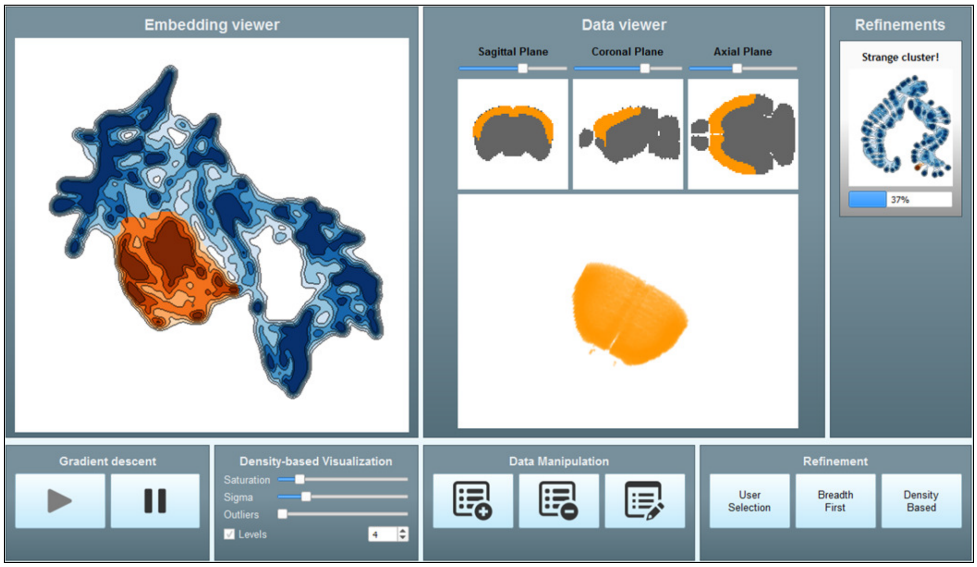


Figure 2.16: Pezzotti et al. (2017)'s system.

Visualization – Pezzotti et al.'s system provides several linked views to the analyst, arranged in two rows. The top one shows (from left to right) the intermediate output (the data

embedding), the original (high-dimensional) data and the algorithm's progression (refinement). The bottom row provides controls over (from left to right) the intermediate data generation, the intermediate data visualization, the data manipulation and the algorithmic process. The analyst can select parts of the embedding, which are highlighted in the high-dimensional representation (orange areas in figure 2.16).

Algorithmic processes – Pezzotti et al. present a progressive version of t-SNE called Approximated-tSNE (A-tSNE). This algorithm approximates the computation of high-dimensional similarities, providing intermediate output as the approximation's precision increases. The cost function of this output is then minimized using a gradient descent, which provides an approximated embedding of the high-dimensional data.

Interaction with the algorithm – Analysts can select a region in the low-dimensional space to decrease the approximation level for the corresponding data points, thus increasing representation precision. They can also add, modify or remove data points on the fly. If they add or remove data attributes, then the algorithm is restarted. In the lower-right section of the interface, three strategies are provided to steer the computation of the embedding. *User selection* prioritizes the data points selected by the analyst. *Breadth first* processes the data points based on their neighborhoods (*i.e.* prioritizing data points close to already processed ones). *Density based* prioritizes data points that are present in the least dense regions of the data.

2.3.6 InsightsFeed (Badam et al., 2017)

Badam et al. propose a system for the exploration of Twitter data, supporting sentiment extraction from the tweets. Their algorithm is not progressive in itself, but the way it is used in combination with an incremental visualization allows them to simulate the kind of interaction possible between a progressive system and an analyst. InsightsFeed is illustrated in figure 2.17.

Visualization – Besides a list of all the tweets, three visualizations are provided. The first one is a vertical bar chart displaying the ratio between negative, neutral and positive sentiments in the data. The second one is a horizontal bar chart indicating the popularity of the users in the dataset (in number of tweets). The third and main visualization is a 2-dimensional projection of the tweets, coming from the t-SNE projection algorithm (van der Maaten & Hinton, 2008) using a semantic similarity measure of the tweets' content. The tweets are aggregated into clusters using the k-means algorithm, and the main keywords that define each cluster are displayed over the map. For each visualization, and also for the global process, a progression indicator is available, indicating how many tweets have been analyzed to create the currently visible representation.

Algorithmic processes – Successive runs of t-SNE and k-means algorithms are performed over increasingly larger subsets of the data. Each run starts with sentiment extraction from the

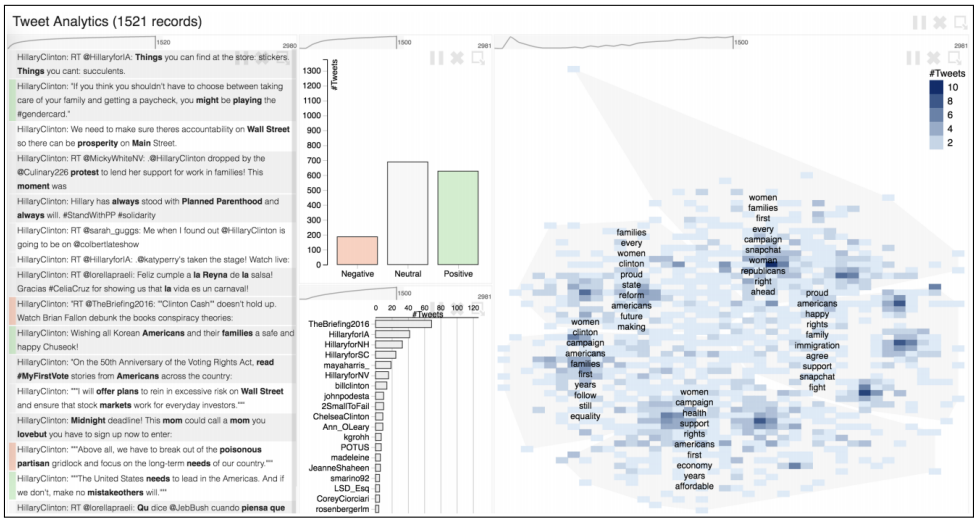


Figure 2.17: InsightsFeed (Badam, Elmqvist, & Fekete, 2017).

tweets' content, before updating the user popularity and updating the t-SNE and clustering.

Interaction with the algorithm – The analyst is able to interact with the algorithm by changing various parameters in each visualization, such as the number of clusters or the number of tweets processed in each step. The new values for the parameters will be taken into account for the subsequent runs of the algorithm. At any moment, the analyst can select a region of the heat map with a click, which pauses the update of the entire interface to allow the analyst to explore the data without updates getting in the way. A "play" button is then available to resume the update.

Even though these systems offer very different ways of interacting with their respective algorithms, let us remark that the authors systematically use the expression "steering the algorithm" (or "steering the computation" for Badam et al. (2017)) to describe some of the interactions they propose. This expression then simultaneously means prioritizing parts of the data (Pezzotti et al., 2017; Williams & Munzner, 2004), prioritizing parts of the results (Stolper et al., 2014), changing parameter values (Badam et al., 2017) and altering the results (Kim et al., 2017). Badam et al. (2017) already identified such semantic variety, stating that "[steering] encompasses multiple types of operations, from filtering data if the analyst wants to focus on specific values, to tuning algorithm parameters and behavior dynamically".

2.4 Designing Progressive Visual Analytics systems

This subsection focuses on the existing work about designing efficient Progressive Visual Analytics systems. We start by reviewing the literature on interaction with algorithms, which has not

always been proposed with Progressive Visual Analytics in mind. The second part is dedicated to the various works having proposed requirements for the design of Progressive Visual Analytics systems, usually encompassing both the algorithm, the visualization and the interactions between the two.

2.4.1 Involving the user in the algorithmic process

Mühlbacher, Piringer, Gratzl, Sedlmair, and Streit (2014) presented four strategies to increase user involvement with algorithms in general. Their work was not focused on Progressive Visual Analytics, but the proposed strategies and the associated algorithm requirements are useful to design PVA systems and define adequate algorithms.

Strategy S1: Data subsetting. Illustrated in figure 2.18, this strategy involves repeatedly running the algorithm over increasingly large subsets of the initial data. It has the advantage of not requiring any knowledge of the inner workings of the algorithm, and offers interaction opportunities to the analyst between each run. However, this strategy necessitates that the data can be sampled efficiently to provide meaningful results at the early steps of the process. If the data subsets are processed in parallel, this approach produces significant overhead with regards to memory and computational power. However, given sufficient resources, this can bring the final result in the same order of time than the default case, with an increased user involvement during the process. Badam et al. (2017) have used this strategy to simulate a progressive system using a non-progressive algorithm, as described in the previous subsection.

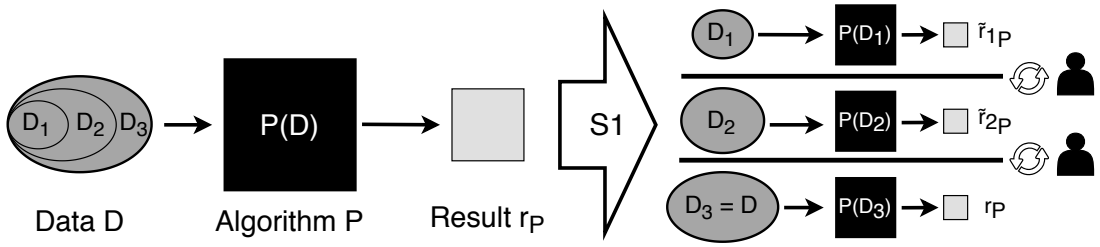


Figure 2.18: Strategy S1: Data subsetting (Mühlbacher, Piringer, Gratzl, Sedlmair, & Streit, 2014). *The complete data D is divided in increasingly larger chunks D_1, D_2 and D_3, that are processed by the unmodified algorithm P. The user can interact with the process once each chunk has been processed. The final chunk corresponds to the original data.*

Strategy S2: Complexity selection. Illustrated in figure 2.19, this strategy involves running the algorithm with different parameter sets of varying complexity over the whole data. This approach is only applicable to algorithms that support a trade-off between result quality and execution time. Similarly to strategy 1, the executions are independent and can be run in

parallel at the cost of some overhead.

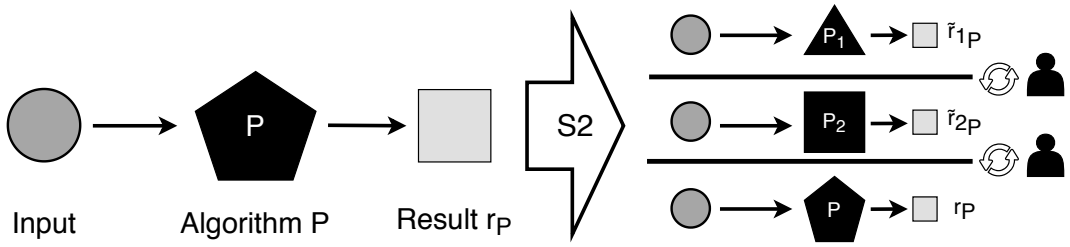


Figure 2.19: Strategy S2: Complexity selection (Mühlbacher, Piringer, Gratzl, Sedlmair, & Streit, 2014). *The data D is processed by several algorithms $P1$, $P2$ and $P3$ of increasing complexity (represented by their number of vertices), derived from the original P algorithm. The final algorithm corresponds to the original one.*

Strategy S3: Divide and combine. Illustrated in figure 2.20, this strategy involves a process where some workload has to be carried out by an algorithm. This workload needs to be divisible into parts, either by dividing the data or the parameter space of the algorithm (e.g. dividing a range of values). The algorithm then produces a result for each part, before combining them into the final result. The strategy suggests that the parts are performed by separate runs of the algorithm (either sequentially or in parallel), before being presented to the user (independently or as a combination of several parts) to provide intermediate feedback and allow user involvement during the process.

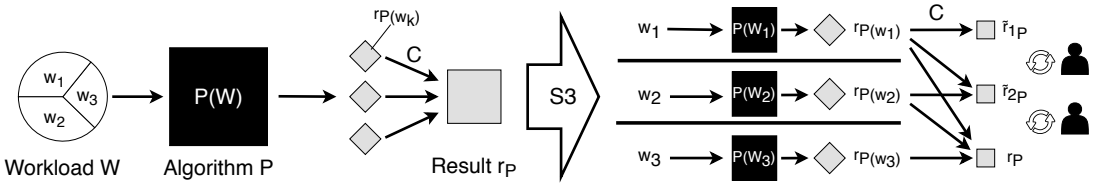


Figure 2.20: Strategy S3: Divide and combine (Mühlbacher, Piringer, Gratzl, Sedlmair, & Streit, 2014). *The workflow W is split in several w_i sub-workflows that are processed in parallel by the unmodified algorithm. Their outcomes are combined as soon as available, allowing interaction with the combination result, even if all sub-workflows are not yet complete.*

Strategy S4: Dependent subdivision. Illustrated in figure 2.21, this strategy consists of dividing an algorithm in a series of steps, each step taking as input the output of the previous one, allowing user involvement between each step. Compared to the first two strategies, no redundant computation is necessary, since the steps take into account the previous ones. Mühlbacher et al. remark that while this strategy is applicable to all iterative algorithms, it is also relevant for sequential processing of an ordered domain, for example signal processing.

In such case, the steps divide the data domain rather than the processing logic.

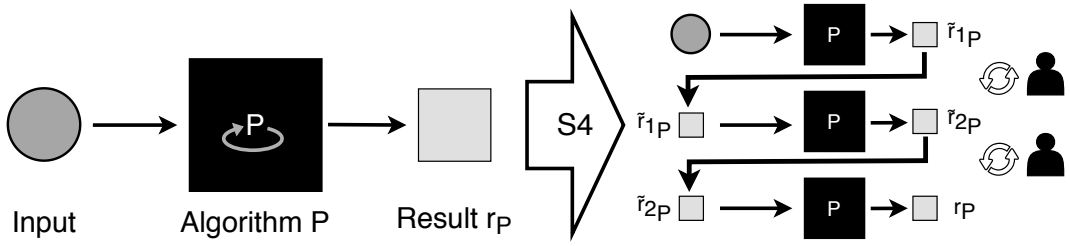


Figure 2.21: Strategy S4: Dependent subdivision (Mühlbacher, Piringer, Gratzl, Sedlmair, & Streit, 2014). *The initial algorithm is broken into several steps that run in sequence, each taking as input the output of the preceding one. Interaction happens at the end of each step, where the partial result is provided to the user.*

Although not a strict implementation of this strategy, existing work from Fekete and Primet (2016) can be seen as a modified version of it. Taking the "Visual" out of Progressive Visual Analytics, they explore the idea of "Progressive Analytics" to provide a formal definition of the term "progressive". To the classical notion of a *function*, they oppose the notion of a *progressive function*. Both require as input a machine state, some parameters and data tables, but the *progressive function* additionally requires a *quantum*. This *quantum* represents the amount of time available to the function, at the end of which a result has to be provided. Through successive executions, these (partial) results converge towards the final result. This differs from Mühlbacher et al.'s S4 strategy in that the execution P provides its $r_{\tilde{P}_n}$ output when its *quantum* is exhausted, whether the task is completed or not.

2.4.2 Types of user involvement

Mühlbacher et al. (2014) propose to classify the ways in which the user of an interactive visualization can be involved in the underlying algorithmic computation, by considering two orthogonal dimensions. The first is the *direction of information*, which represents whether the information is passed from the computation to the user (*feedback*) or the other way round (*control*). The second dimension is the *entity of interest*, which can either be the *execution* of the process or its *result* (final or intermediate). Using the Cartesian product of these two dimensions, Mühlbacher et al. identify four types of user involvement: Execution Feedback, Result Feedback, Execution Control and Result Control (figure 2.22).

When considering these types of user involvement, their focus was to identify elements that could improve the user's exploration experience, either through information visualization (*feedback* in figure 2.22) or by giving the user additional power over the process (*control* in figure 2.22). They remain algorithm-agnostic in their discussion, but clearly state that the usefulness and difficulty of implementing their various propositions might vary depending on the algorithmic process at hand.

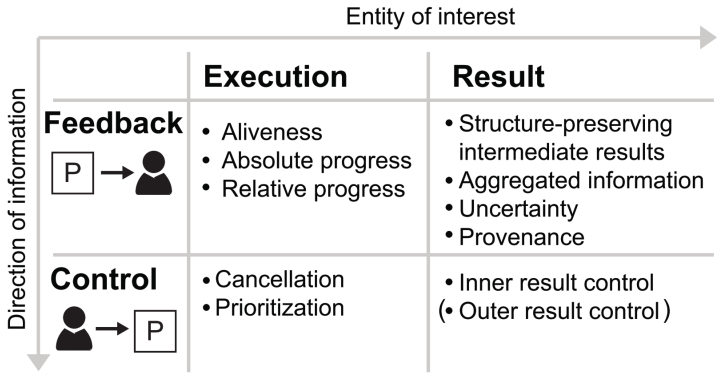


Figure 2.22: Types of User Involvement (Mühlbacher, Piringer, Gratzl, Sedlmair, & Streit, 2014).

2.4.3 Requirements for Progressive Visual Analytics systems

Improving interaction between an analyst and an analytics system has been the focus of existing works outside of the Progressive Visual Analytics paradigm, which served as a basis for subsequent works on the design of Progressive Visual Analytics systems. This led to the identification of requirements and guidelines to improve the effectiveness and facilitate the development of such systems.

While tackling the problem of moving from static data visualization to dynamic visualization, E. G. Hetzler et al. (2005) identified the following four design guidelines that would benefit such system:

- Allowing one to monitor the visualization and see new elements when the visualization is updated
- Minimize disruption to the analytic and interaction process
- Provide as much interactivity as possible
- Provide dynamic update feature (*i.e.* varying granularity of the visualization)

Stolper et al. (2014) extended these guidelines to enumerate their goals for Progressive Visual Analytics, that they organize as analytics component-related goals and visualization-related goals. Analytics components should provide increasingly meaningful partial results during the algorithm's execution and allow users to both focus the algorithm to subspaces of interest and ignore irrelevant subspaces. Visualizations should minimize distractions by not changing views excessively, provide cues to indicate where new results have been found and support on-demand refresh when analysts are ready to explore the latest results. Echoing the second analytics component-related goal, visualizations should also provide an interface for

users to specify where analytics should focus, as well as the portions of the problem space that should be ignored.

Badam et al. (2017) built on the works of E. G. Hetzler et al. (2005), Stolper et al. (2014) and Mühlbacher et al. (2014) to propose an aggregated list of 18 requirements for the design of Progressive Visual Analytics systems, numbered R1 to R18. They organize the requirements according to Mühlbacher et al.'s types of user involvement, thus identifying four categories: requirements for feedback on the result, requirements for feedback on the execution, requirements for control over the result and requirements for control over the execution. These categories are presented in table 2.1.

Table 2.1: Requirements for the design of Progressive Visual Analytics systems (Badam, Elmqvist, & Fekete, 2017).

	Result	**Execution**
Feedback	R1: Meaningful partial results R2: Structure-preserving intermediate results R3: Retaining cognitive workflow on updates R4: Minimized distraction during updates R5: Cues for new results R6: Aggregated information R7: Uncertainty R8: Provenance information on demand R17: Provide similarity anchors in complex visualizations R18: Consistently offer quality measures	R9: Aliveness R10: Absolute progress R11: Relative progress
Control	R12: Full interactivity R13: Support two modes: constant update and on-demand refresh R14: Steer results	R15: Cancellation R16: Prioritization

In this subsection about the design of Progressive Visual Analytics systems, we saw that earlier work explored changing algorithms to better involve the user in their execution, while more recent work also consider the system within which the analyst and the algorithm need to interact. However, due to the limited number of existing implementations of Progressive Visual Analytics systems, these works are for the most part theoretical, with only a few practical studies having been conducted.

2.5 Interaction-related user studies in Progressive Visual Analytics

Fisher et al. (2012) explored the impact of incremental visualization on decision-making and data exploration. 3 groups of experts had to perform an exploratory analysis with aggregate

queries (sums, averages, counting) on data they were familiar with. Results were incrementally displayed based on ever-larger portions of the data, along with a representation of uncertainty (confidence bounds). Incremental computation and visualization were shown to be effective and to allow for new exploration behaviors, as long as the analyst had a good estimation of the current result's confidence bounds.

Glueck, Khan, and Wigdor (2014) explored progressive loading of large time series. They asked 12 volunteers to explore data they were familiar with, concluding that progressiveness allowed early exploration and accurate report of important insights that could be confirmed by a prolonged analysis if needed.

Zgraggen, Galakatos, Crotty, Fekete, and Kraska (2017) compared the effects of various conditions for 24 students having to explore 3 datasets: blocking (results are available at the end of the computation), progressive, and instantaneous (an ideal system which displays accurate results immediately for any query). They confirmed previous findings by Liu and Heer (2014) that latency has a greater impact on brushing than on panning and zooming, and that instantaneous and progressive visualization lead to a similar number of insights, while the latter increases user involvement compared to blocking condition.

Badam et al. (2017) compared analysts' sense-making with classical and Progressive Visual Analytics tools, focusing on how they build their confidence when working with approximate answers. 10 participants from their HCI/visualization lab had to explore data to answer precise questions. They found that users approach early results in very different ways, some preferring a broad overview, others wanting as much information displayed as possible. This stresses the importance of providing ways for the user to control the process, for example having control over the speed and quantity with regards to the update of intermediate results. It also highlights the importance of displaying information about intermediate results' provenance and quality, in order to allow one to efficiently leverage these controls. Perhaps more importantly, they note that answers given based on the progressive strategy's partial results are similar to those obtained from the complete data, which supports the fact that Progressive Visual Analytics does not lead to worse results than classical analysis techniques.

From our review of the literature, it appears that user studies about interaction in Progressive Visual Analytics systems mainly focus on interactions between the analyst and the visualization. This leaves the interaction between the analyst and the progressive algorithms relatively unexplored.

3 Conclusion – Challenges for Progressive Visual Analytics

As presented in the first section of this chapter, Visual Analytics is a data analysis paradigm that aims at increasing the user involvement in the analysis process through the use of visualiza-

tions. While this approach has proven successful, an increase in the amount of data to handle and in algorithm complexity can be detrimental to its efficiency. The paradigm of Progressive Visual Analytics has been proposed as a solution to these problems, and several approaches exist that target different application cases.

The current definition of Progressive Visual Analytics states that analysts need to be able to interact with the algorithmic process. However, it does so using a very permissive description of what "interaction" stands for in this case, which enabled a variety of approaches. While this has most certainly been beneficial to spread awareness about Progressive Visual Analytics, we now face the fact that existing works in this domain are not consistent on the terms they use. From one work to another, different actions are designated using the same word (with *steering* being the most prominent example), and similar actions are referred to in different ways.

The definition of Progressive Visual Analytics also highlights that the main difference from Visual Analytics is the addition of intermediate results and the ability to interact with the algorithm, making it clear that the algorithm plays a key role in any progressive system. This increased complexity of the algorithmic process effectively makes process models inherited from Visual Analytics obsolete since they often consider the algorithm as a black box, or even a simple transition between data and visualization. A possible consequence of this legacy is that consideration of the ways in which one can interact with the algorithm is minimal in Progressive Visual Analytics works. From these observations, we identify our first challenge for Progressive Visual Analytics:

Challenge PVA1

Clarify what "interaction" means in the context of Progressive Visual Analytics, as well as investigate the role of the algorithm in the process.

The definition of Progressive Visual Analytics advocates a tight coupling between the user and the analytic system. When considering the existing literature, most work on interaction has been focused on interaction between the human and the visualization, dealing with handling intermediate results and the uncertainty that comes with them. In comparison, interaction between the human and the algorithm are less studied, even though being able to interact with this part of the system is a key aspect of the definition of Progressive Visual Analytics. Considering that interacting with the algorithm can impact the execution time, the amount of output or even the analysis process at large, it is all the more important to address this matter. This is the basis of our second challenge for Progressive Visual Analytics:

Challenge PVA2

Investigate the consequences of interactions between the human and the algorithm on the analysis process.

CHAPTER 3

SEQUENTIAL PATTERN MINING

Among the various analysis techniques available to extract knowledge from data, pattern mining aims at extracting repeating parts from the data. In a broad sense, a *pattern* can be seen as a piece of data. As such, patterns can provide an interesting representation of data thanks to three main characteristics:

- They are **intelligible**, due to the fact that they are formed from data elements. This is in contrast with black box processes such as neural networks, that can for example be used in machine learning.
- They provide very **specific information**, thus highlighting the presence (or absence) of precise elements within the data.
- Their combinatorial nature leads to **unexpected discoveries**, as opposed to target-centered data analysis. However, this also leads to a large amount of patterns, even in small datasets.

Given the wide range of pattern mining techniques, providing a detailed review of the state of the art is beyond the scope of this manuscript. Instead, we focus on existing work on our topic of interest, sequential data. Moreover, the fact that pattern mining has been studied for many years has led to the design of efficient algorithms for the general task. As such, recent work within the field are mostly focused on specific approaches (considering either specific types of data or specific types of patterns). Due to our focus on Progressive Visual Analytics, we limited the scope of our work to the general task of sequential pattern mining, which explains the lack of references to recent works in section 3.2, as well as in Appendix A.

Section 3.1 provides a general presentation on the matter of Sequential Pattern mining. It opens with a number of important definitions, and then introduces the two categories of patterns that can be encountered in the literature. The section continues with a presentation of the various ways in which one can constrain the pattern mining process, and ends on the matter of pattern occurrence counting. Section 3.2 presents the two categories of pattern mining algorithms that exist in the literature. Section 3.3 concludes both this chapter and the first part of this manuscript, highlighting two challenges for Progressive Pattern Mining.

Note: In the literature, the expression "sequential pattern" is used to designate both patterns extracted from event sequences in general, and patterns in the specific approach of Agrawal and Srikant (1995). In order to distinguish between these two, we use "Sequential Pattern" for the general name and "sequential pattern" for the specific approach (note the difference in capitalization).

1 Mining patterns in sequences

Sequential Pattern mining refers to the application of pattern mining techniques on sequential data. While this type of data can be encountered in almost every domain, and as such can take different shapes, it always involves an ordering of the data items (either a partial or total one). For example, sequences can be encountered when considering DNA data, texts, or temporal sequences of events such as logs. Sequences contain symbolic items, as opposed to time series, which contain numerical values obtained by observing a phenomenon over time (Fournier-Viger et al., 2017).

Due to the extensive existing work, several surveys on Sequential Pattern mining have been written over the years, such as the ones by Mooney and Roddick (2013) and Fournier-Viger et al. (2017).

1.1 General definitions

There exist two reference approaches to mine patterns from sequential data: the *sequential pattern* approach by Agrawal and Srikant (1995) and the *episode* approach by Mannila, Toivonen, and Verkamo (1995). They are presented respectively in subsections 3.1.2 and 3.1.3. Even though these two approaches share a lot of similarities, presenting them using a unified formal representation that takes into account every edge case quickly leads to complex expressions. For this reason, in this subsection we propose general informal definitions that encompass the two approaches. For a unified view with formal notations, one can refer to Joshi, Karypis, and Kumar (1999).

1.1.1 Events

An event, in the general sense, can be defined as "something that has happened at some point in time". In the context of Sequential Pattern mining, this definition can be extended to "something that has happened at some point in time, and has been recorded". As such, an event has at least three properties:

- Its *event type*, describing *what* has happened. The list of all event types in a dataset can be considered the alphabet of the data.

- Its *sequence*, *i.e.* the record series it belongs to.
- Its *timestamp*, describing *when* it happened. Even though the word "timestamp" usually refers to a date, it can also be recorded in other ways such as the index at which the event is found in the data. Depending on the data, the timestamp can either be a single value (indicating an instantaneous event) or a pair of value (indicating the beginning and end of the interval over which the event has been recorded).

Additional properties can be tied to an event depending on the kind of data and how it was collected.

1.1.2 Sequences

A sequence is an ordered list of events (Agrawal & Srikant, 1995; Mannila & Toivonen, 1996). Some examples can be sequences of events performed by a given subject, or sequences of failures and breakdowns of different machines as parts of a production line. Sequences carry two properties:

- Their *size* (or *length*), representing the number of events within the sequence. A sequence of size k is sometimes called a k-sequence.
- Their *duration*, which is the time difference between their first and last events.

Just like events, sequences can also carry additional properties depending on the data they represent. The ordering of a sequence is based on the *timestamp* of the events it contains, and as such can be time-based or index-based. Finally, an important property of such sequence is whether or not two events can occur at the same position.

1.1.3 Sequence databases

A sequence database is a collection of sequences (Agrawal & Srikant, 1995). Its main property is its *size*, the number of sequences it contains.

1.1.4 Patterns

In the context of sequential data, a pattern is a list of event types, either totally or partially ordered. Every pattern has at least three defining properties:

- Its *syntax*, represented by the list of its event types.
- Its *size*, the total number of event types contained in this list. A pattern of size k is sometimes called a k-pattern.

- Its *support* (or *frequency*), the number of its occurrences (*i.e.* the number of times it is encountered in a given sequence database). This attributes relies on the way pattern occurrences are counted, and can be expressed either in an absolute way, or relatively to the dataset's size.

Additionally, a *frequent pattern* is a pattern whose *support* is at least equal to a user-given threshold value. Since pattern mining can output large number of patterns even on small datasets, in practice a pattern mining task is always a frequent pattern mining task.

Other variations on the notion of pattern have been proposed in the literature, such as *closed patterns*, frequent patterns that are not included in longer patterns of similar support. Another example, more restrictive, is *maximal patterns*, which are not included in longer frequent patterns (regardless of their support).

1.1.5 Sub-pattern

Given two patterns A and B, B is said to be a *sub-pattern* of A if its syntax is included in A's syntax, in the same order.

1.1.6 Pattern occurrences

The concept of "pattern occurrence" has no absolute definition. In fact, the way an occurrence is defined depends on the strategy used to identify it, *i.e.* the occurrence counting strategy used in pattern mining algorithm (see subsection 3.1.5). Broadly speaking, an occurrence of a given pattern in a given dataset is a sub-part of the dataset where the pattern is found. Contrary to patterns, pattern occurrences are time stamped; they have a beginning and an end in the sequence. A pattern occurrence has the following properties:

- A *duration*, indicating the difference between the timestamps of its first and last events.
- A *length* (or *size*), indicating the number of events within it.

More properties can be tied to an occurrence depending on the properties of the events they involve.

Figure 3.1 presents an illustration of the previous definitions. The sequence S of size 8 contains 3 event types (A, B and C). One can consider the pattern <B;A> of size 2. In S, <B;A> has a support of 2. The first occurrence has a length of 3, starts at $t = t_6$ and ends at $t = t_8$, thus having a duration of 3. The second occurrence of <B;A> has a length of 2, starts at $t = t_{12}$ and ends at $t = t_{13}$, thus having a duration of 2.

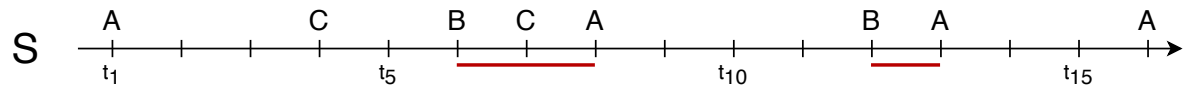


Figure 3.1: An example of sequential data. *<B;A> is a pattern, whose two occurrences are underlined in red.*

1.2 Sequential patterns

Sequential patterns constitute most of the literature on the subject of Sequential Patterns. They were defined by Agrawal and Srikant (1995) in the context of market basket analysis, along with their formulation of the sequential pattern mining task:

> "*Given a database of sequences, where each sequence consists of a list of transactions ordered by transaction time and each transaction is a set of [event types], sequential pattern mining is to discover all sequential patterns with a user-specific minimum support, where* ***the support of a pattern is the number of data-sequences that contain the pattern***".

This definition highlights the key aspects of *sequential patterns*:

1. They are to be discovered within multiple sequences, where several events may occur at the same time;
2. The number of sequences in which the pattern is found has more value than the number of times the pattern is found in each sequence.

This second aspect is also present in later definitions of sequential pattern mining, such as the one by Garofalakis, Rastogi, and Shim (1999):

> "*Given a set of data sequences, the problem is to discover sub-sequences that are frequent, i.e.* ***the percentage of data sequences containing them*** *exceeds a user-specified minimum support*".

1.3 Episodes

The notion of *episode* has been proposed by Mannila et al. (1995), which offer the following definition:

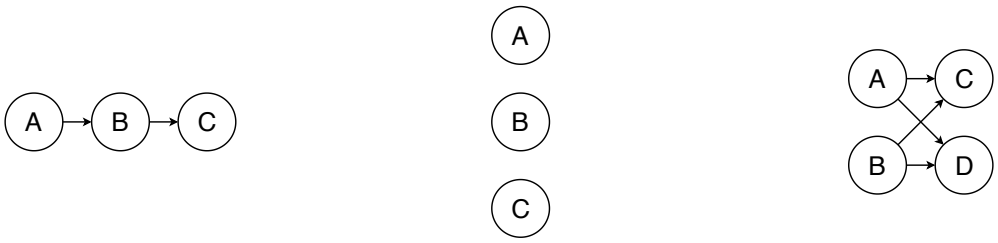
> "*[An episode is a] collection of [event types] that occur relatively close to each other in a given partial order*".

While their definition could be assimilated with the one for *sequential patterns*, the distinctive aspect is that episodes are extracted from a unique sequence rather than a set of sequences.

As such, the *support* of an episode represents the actual number of times it can be found in the data rather than the number of sequences in which it is present. In practice, when dealing with several sequences, one can extend the definition of the *support* of an episode to be the sum of its support within each individual sequence.

Mannila et al. (1995) allow a partial ordering of the event types within episodes. In particular, they highlight two specific types of episodes:

- Serial episodes, where the event types are totally ordered (figure 3.2a);
- Parallel episodes, where no constraints are given on the relative order of the event types (figure 3.2b).



(a) A serial episode

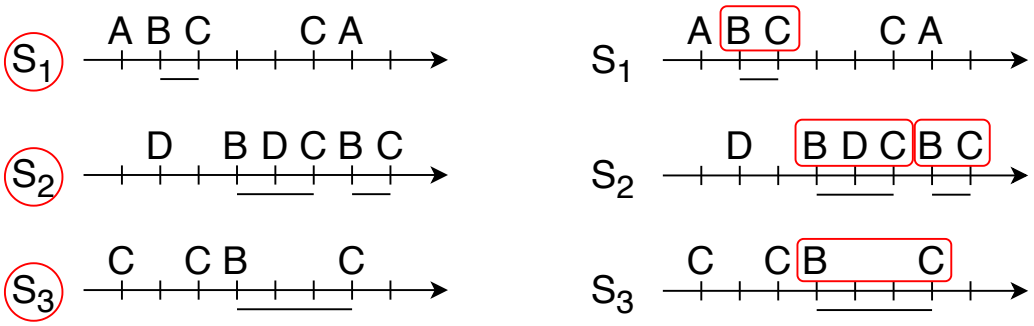
(b) A parallel episode

(c) A serial combination of parallel episodes

Figure 3.2: Different types of episodes proposed by Mannila, Toivonen, and Verkamo (1995). *Episodes are represented as directed acyclic graphs.*

They also mention a third case, which is a serial combination of parallel episodes (figure 3.2c). In this subsection, we focus on serial episodes, as does most of the existing literature.

The difference in the way sequential patterns and episodes consider pattern occurrences is illustrated in figure 3.3. On the left, <B;C> is considered as a sequential pattern. Even though it can be seen four times in the data, it is encountered in three sequences, leading to a support of 3 (or 100% if we consider relative support). On the right, <B;C> is considered as an episode. Its support would be 1 in sequence S_1, 2 in sequence S_2 and 1 in sequence S_3. This implies a global support of 4 when considering all three sequences.



(a) <B;C> as a sequential pattern

(b) <B;C> as an episode

Figure 3.3: Comparison between sequential patterns and episodes over the same data. *Places where the pattern <B;C> can be found are underlined. Occurrences of the pattern are highlighted in red.*

1.4 Constraints on the mining process

Adding constraints to the pattern mining process allows one to restrict its output, either to shorten the computation or to target interesting patterns more precisely. Based on previous work by Pei and Han (2002), Mooney and Roddick (2013) identified several types of constraints that we organize into two categories.

1.4.1 Constraints on the pattern

As the name suggests, constraints on patterns target the pattern itself and its properties. Mooney and Roddick identify the following three types of such constraints:

- **Support constraints:** Specifying a constraint based on the support of a pattern is done to restrict the mining output to *frequent* patterns; *i.e.* patterns that are encountered at least as often as specified by the minimum support value. Frequently called *support threshold*, this constraint is at the core of frequent pattern mining algorithms' logic, and as such is implemented in every pattern mining system (see subsection 3.1.1.4).

- **Syntax constraints:** These constraints dictate the elements one is looking for. Depending on the work, algorithm and use case, it can be specified in multiple ways, either in reference to another pattern (*model-based constraint* in Mooney and Roddick (2013)), by specifying the type or property of some of its elements (*item constraint* in Mooney and Roddick (2013)), using regular expressions to describe the search target . . .

- **Length constraints:** These constraints restrict the set of patterns for which occurrences should be found, depending on the number of event types they contain.

1.4.2 Constraints on the occurrences

The idea of constraints applied to an occurrence was first introduced with Srikant and Agrawal (1996)'s GSP algorithm, that proposed the notion of *gap* and *span* (under the name *window-size*). The following constraints can be used:

- **Span constraints:** These constraints restrict the time difference allowed between the first and the last events involved in a pattern occurrence.
- **Gap constraints:** These constraints target the relation between two consecutive events involved in a pattern occurrence. They can either be expressed as a duration (time between the events) or a number of events that do not belong to the occurrence.
- **Aggregate constraints:** These constraints target the attributes of the events that compose a pattern occurrence, through a numerical aggregate function such as average, minimum, maximum... For example, when considering events indicating a stock value either increasing or decreasing, one could be looking for the pattern `<increase;decrease;increase>` with the aggregate constraint that the total variation must be above 2%.

In both constraint categories, numerical constraints (length, support, duration, gap, aggregate) can be expressed either with a minimum or a maximum value, or both at the same time. On the other hand, syntactic constraints can be expressed as requirements or as interdictions. While these constraints can be used as a post-process step to reduce the number of patterns in the final output, Garofalakis et al. (1999) stress that they should be "pushed" inside the mining process whenever possible. Doing so can provide significant performance benefits, by getting rid of useless patterns earlier in the process.

1.5 Counting pattern occurrences

While it may appear trivial, counting the number of times a pattern appears in a sequence is not a well-defined task. For example, if we consider the sequence in figure 3.4, one could identify occurrences of the pattern `<A;B>` in several ways. The sequence could be scanned left to right, with an occurrence starting from an event of type A until the next event of type B, without allowing a given event to be part of more than one occurrence. Using this strategy, `<A;B>` has two occurrences : $(t_3;t_6)$ and $(t_4;t_{10})$. One could also follow the same strategy, but allow an event to be part of several occurrences. This would lead to `<A;B>` having three occurrences: $(t_3;t_6)$, $(t_4;t_6)$ and $(t_8;t_{10})$. Another possibility could be to prioritize minimal occurrences (*i.e.* the ones that do not contain other occurrences), leading to `<A;B>` having two occurrences: $(t_4;t_6)$ and $(t_8;t_{10})$. While non-exhaustive, this example highlights the fact that several ways of counting a pattern's occurrences exist, without one being clearly better than the others. In

fact, two counting strategies can provide two sets of occurrences O and O' that differ in support (*i.e.* $|O| \neq |O'|$) or in location (*i.e.* $|O| = |O'|, O \neq O'$).

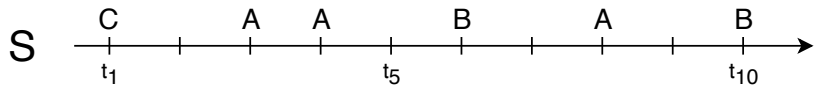


Figure 3.4: *A sequence that can contain occurrences of the pattern* `<A;B>`

From the literature, Joshi et al. (1999) identify five methods for counting the occurrences of a pattern. They are organized in the following three categories:

- **Counting sequences.** With this strategy, the support of a pattern is equal to the number of sequences in which the pattern is found, no matter how many times it is present in these sequences. This strategy is the one used by the GSP algorithm for sequential patterns (Srikant & Agrawal, 1996).
- **Counting the windows in which the pattern is present.** This strategy consists in sliding a fixed-size window along the sequence, with the pattern's support being the number of windows in which it has been found. This method is the one used by the WINEPI algorithm for mining occurrences of an episode in a sequence (Mannila, Toivonen, & Inkeri Verkamo, 1997).
- **Counting the sets of events that match the pattern.** In this strategy, an occurrence of a pattern is a subpart of the sequence. More precisely, an occurrence of a pattern in a sequence is a list of events included in the sequence that:
  - begins with the first event type of the pattern's syntax;
  - ends with the last event type of the pattern's syntax;
  - contains all the other event types of the pattern's syntax, in the correct order.

  Joshi et al. identify two variants of this strategy, depending on whether or not an event in the sequence can be part of several occurrences of the same pattern. In the first case, two occurrences differ by at least one event, while in the second case they have to share no event. In both cases, occurrences may overlap. These methods are the ones used by the MINEPI algorithm for mining occurrences of an episode in a sequence (Mannila et al., 1997)

Another comparison of several counting strategies is proposed by Gan and Dai (2010), more focused on the properties of the different strategies than on their inner workings. While Joshi et al. (1999) were considering strategies applicable to both sequential patterns and episodes,

Gan and Dai (2010) focus on episodes, and compare the seven strategies they identified in the literature, according to three properties:

- **Use of a search window:** using a window allows one to constrain the time between data events involved in a pattern occurrence. Depending on the strategy used, a window can be fixed (*i.e.* having the same size for all patterns), variable (*i.e.* having its size depend on the pattern) or unbounded (*i.e.* spanning the full sequence, which amounts to not using a window).
- **Anti-monotonicity:** when considering a sequence S and any two patterns p_1 and p_2 with p_1 being a sub-pattern of p_2, a support measure $supp$ is anti-monotonic if $supp(S, p_1) \geq supp(S, p_2)$. When used in combination with a minimum support constraint, this property allows one to safely prune the Sequential Pattern search space.
- **Maximum support:** indicates that the strategy is able to find the greatest number of occurrences, compared to the other strategies. It is important to note that this property only applies to support measures that are anti-monotonic.

Table 3.1 relates their comparison of the following strategies[1] one can use to measure the support of an episode in a sequence:

- **fixed-win-supp** (Mannila et al., 1995): using a sliding window of a fixed size w, the support is the number of windows that contain the pattern.
- **mo-supp** (Mannila & Toivonen, 1996): In this strategy, an occurrence of a pattern is directly a subpart of the sequence. More precisely, an occurrence of a pattern in a sequence is a set of events included in the sequence that:
  - begins with the first event type of the pattern;
  - ends with the last event type of the pattern;
  - contains all the other event types of the pattern between the two extreme events in the correct order.

  A minimal occurrence is an occurrence that does not include another occurrence. The support is the number of minimal occurrences.
- **auto-win-growth-supp** (Casas-Garriga, 2003): for a given max_gap value, a sliding window of a variable size $w = (l-1) \times max\_gap$ is used, where l is the size of the considered pattern. The support is the number of windows having $size \leq w$ that contain the pattern.

[1] For the sake of consistency, we use the term "support" rather than "frequency". The names of the categories also reflect this decision, for example using "fixed-win-supp" instead of "fixed-win-freq".

- **maxgap-mo-supp** (Méger & Rigotti, 2004): given values for max_gap and max_span, the support is the number of minimal occurrences that satisfy the max_gap and max_span constraints.

- **T-supp** (Iwanuma, Ishihara, Yo Takano, & Nabeshima, 2005): using a sliding window of a fixed size w, for every sub-pattern of the considered pattern this method registers the number (called H-supp) of windows that contain the sub-pattern and start with its first event type. The support of the pattern is the lowest H-supp value among the sub-patterns.

- **non-overlapped-supp** (Laxman, Sastry, & Unnikrishnan, 2007): the search targets the set of non-overlapped occurrences of the pattern. Two occurrences are non-overlapped if no event corresponding to one occurrence appears in between events corresponding to the other. The support of the pattern is the number of these occurrences.

- **distinct-bound-st-supp** (Huang & Chang, 2008): using a sliding window of a fixed size w, this method registers the number of windows that contain the pattern and start with its first event type. The support of the pattern is the number of these windows.

Table 3.1: Properties of episode occurrences counting strategies (Gan & Dai, 2010). *The last column indicates the corresponding category from Joshi, Karypis, and Kumar (1999).*

Strategy	Window	Anti-monotonicity	Max support	Joshi et al.
fixed-win-supp	fixed-win	Yes	Yes	Windows
mo-supp	unbounded	Yes	No	Event sets
auto-win-growth-supp	variant-win	No		Windows
maxgap-mo-supp	variant-win	Yes	No	Event sets
T-supp	fixed-win	Yes	No	Windows
non-overlapped-supp	unbounded	Yes	No	Event sets
distinct-bound-st-supp	fixed-win	No		Windows

When looking at the three occurrence-counting methods from Joshi et al. (1999), the strategies reviewed by Gan and Dai (2010) use a mix of the last two (counting windows and counting sets of events). The absence of the first method (counting sequences) is expected, due to the focus on episodes, thus on single-sequence datasets. From their review, Gan and Dai (2010) draw five observations:

- Using a restricted window-size based on the episode's length is necessary to ensure that an occurrence is compact enough.
- Anti-monotonicity is necessary to obtain an accurate measure of support, but insufficient.
- Whether they satisfy maximum support or not, anti-monotonic measures could introduce inaccuracies into support measures.

- Since they impact the support measure, the first three observations impact the discovery of frequent episodes.
- All existing support measures have inherent inaccuracies, impacting the soundness and completeness of the results.

In order to avoid both false and missed frequent patterns, they recommend to choose the window size based on the considered episode's size and max_gap, that support measures should be strictly anti-monotonic and have the property of maximum support. They define strict anti-monotonicity as being anti-monotonic while only considering non-redundant sets of occurrences. This means that any two occurrences of a given pattern should not involve the same event type/timestamp pair at the same position.

2 Sequential Pattern mining algorithms

The Sequential Pattern mining problem is essentially an enumeration problem, that pattern mining algorithms address by exploring the search space of Sequential Patterns (Fournier-Viger et al., 2017). All Sequential Pattern mining algorithms share a high-level logic of extending known k-patterns to generate $(k+1)$-patterns. Among the constraints presented in subsection 3.1.4, one always used in pattern mining is the support constraint. This means that only the patterns that are frequent with regards to a user-defined threshold are expected, in order to limit the number of output. This led to the emergence of two strategies for Sequential Pattern mining algorithms, that we describe in this section.

2.1 Apriori-like algorithms

The family of Apriori-like algorithms draws its name from the *Apriori* itemset mining algorithm (Agrawal & Srikant, 1994), from which they borrow the following two-step strategy:

- **Generating candidate patterns**. A candidate pattern is a pattern whose occurrences are not yet known. In terms of the constraints described in subsection 3.1.4, it means that it complies with any syntax and length constraints, but its support is still unknown. Generating a set of candidate patterns is a combinatorial task that can be done without reading the data. The anti-monotonicity property of the support (see subsection 3.1.5 for a definition) plays an important role in this step, allowing one to reduce the number of combinations based on previously discovered frequent patterns.
- **Checking candidate patterns**. Once a candidate set has been generated, the following step is to look for the occurrences of said candidates. This is where the data is read, and where the constraints targeting pattern occurrences are verified. At the end of this

step, each candidate pattern has either been encountered enough time to be considered frequent, or has been discarded as infrequent.

Taking advantage of the anti-monotonicity property during candidate generation, Apriori-like algorithms construct the candidate patterns by extending already discovered frequent patterns. From the list of frequent patterns of a given size n, candidates of size $n+1$ are generated. These candidates are then verified, and the frequent ones are combined to generate candidates of size $n+2$. The process is then repeated until no candidate can be generated or no new frequent pattern can be found. This leads to a general process where the two previously described steps are performed alternately. To determine the order in which the candidates will be generated and verified, two strategies can be found in the literature:

- **Breadth-first search.** Breadth-first algorithms prioritize the discovery of short patterns over longer ones. This means that all candidates of size n are generated and checked, before moving on to size $n+1$, and so on. When considering a tree representation of the patterns present in a dataset, this amounts to building the tree level by level (Figure 3.5).
- **Depth-first search.** Depth-first algorithms prioritize extending already discovered patterns. This means that when a pattern of size n is found to be frequent, it is extended to generate candidates of size $n+1$ that are then verified. This continues for as long as possible, before going back to searching for other patterns of size n. When considering a tree-representation of the patterns, this amounts to focusing on the branches rather than the levels (Figure 3.5).

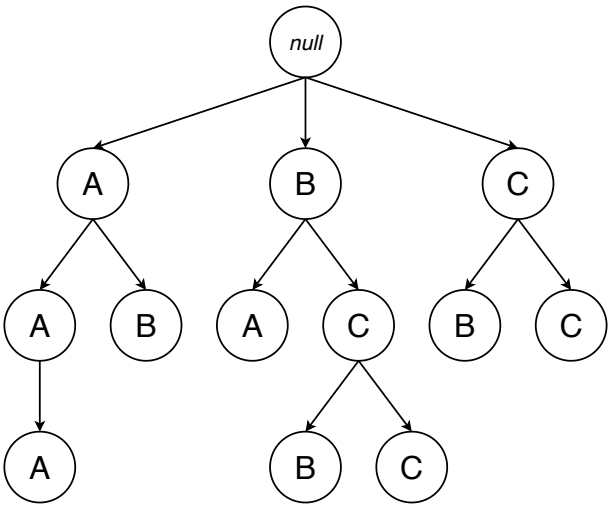


Figure 3.5: Tree-representation of a set of patterns. *The following patterns are included: <A>, <B>, <C>, <A;A>, <A;B>, <B;A>, <B;C>, <C;B>, <C;C>, <A;A;A>, <B;C;B>, <B;C;C>.*

Besides these two strategies to determine in which order the patterns will be processed, the difference between two algorithms can also be found in the format they expect for their input data. Again, two categories can be encountered, that are illustrated in table 3.2:

- **Horizontal data format.** In this case, the events that compose the dataset are ordered according to the sequence they belong to, and then by their transaction time. One could describe this data format as presenting a set of sequences and their content (see table 3.2a).
- **Vertical data format.** In this case, each event type is associated with the sequences in which it is found. One could describe this data format as presenting a set of event types and the sequences they are part of (see table 3.2b).

Table 3.2: Horizontal and vertical data format. *The same data is shown in both formats.*

(a) Horizontal data format. Sequences contain (Time, EventType) pairs.

Sequence Id	Sequence
1	(2, A) (5, A) (10, B)
2	(1, C) (10, B) (15, C)
3	(4, B) (6, A)
4	(1, B) (11, C) (12, A)

(b) Vertical data format. Occurrences contain (SequenceId, Time) pairs.

Event Type	Occurrences
A	(1, 2) (1, 5) (3, 6) (4, 12)
B	(1, 10) (2, 10) (3, 4) (4, 1)
C	(2, 1) (2, 15) (4, 11)

Table 3.3 presents an overview of some of the available Apriori-like Sequential Pattern mining algorithms. A brief description of these algorithms is available in appendix A.

2.2 Pattern growth algorithms

While it has its advantages, the method used by Apriori-like algorithms presents two major downsides. First, even with the addition of constraints and properties such as anti-monotonicity, the combinatorial nature of the candidate generation step means that the number of candidates increases exponentially with the number of event types in the data. More importantly, since candidate generation is done without reading the data, some of the candidates are absent from the data, leading to unnecessary work during the verification step. Secondly, performing multiple passes over the data is more and more costly as the size of the dataset increases.

The *pattern growth* (or *frequent pattern growth*) approach has been introduced to remedy these downsides in the following ways:

- Pattern growth algorithms perform a single pass over the raw data that is used to build a condensed representation of its important elements (*i.e.* the frequent event types), which can take various forms. Some conserve the initial data structure, such as projected

Table 3.3: Non-exhaustive overview of existing Apriori-like Sequential Pattern mining algorithms. *Strategy can be either Breadth- (BFS) or Depth-first search (DFS). Data format can be either vertical (V) or horizontal (H). Pattern type can be either sequential patterns (Seq. patterns) or Episodes*

Algorithm	Reference	Pattern type	Strategy	Data format
AprioriAll	Agrawal and Srikant (1995)	Seq. patterns	BFS	H
AprioriSome	Agrawal and Srikant (1995)	Seq. patterns	BFS	H
DynamicSome	Agrawal and Srikant (1995)	Seq. patterns	BFS	H
GSP	Srikant and Agrawal (1996)	Seq. patterns	BFS	H
PSP	Masseglia, Cathala, and Poncelet (1998)	Seq. patterns	BFS	H
SPIRIT	Garofalakis, Rastogi, and Shim (1999)	Seq. patterns	BFS	H
MFS	M. Zhang, Kao, Yip, and Cheung (2001)	Seq. patterns	BFS	H
SPADE	Zaki (2001)	Seq. patterns	Both	V
SPAM	Ayres, Flannick, Gehrke, and Yiu (2002)	Seq. patterns	DFS	V
CCSM	Orlando, Perego, and Silvestri (2004)	Seq. patterns	BFS	V
MSPS	Luo and Chung (2004)	Seq. patterns	BFS	H
LAPIN-SPAM	Yang and Kitsuregawa (2005)	Seq. patterns	DFS	V
IBM	Savary and Zeitouni (2005)	Seq. patterns	BFS	V
WinEpi	Mannila, Toivonen, and Verkamo (1995)	Episodes	BFS	H
MinEpi	Mannila and Toivonen (1996)	Episodes	BFS	H

databases (Han, Pei, Mortazavi-Asl, et al., 2000; Pei et al., 2001), while others differ from it, for example FP-trees (Han, Pei, & Yin, 2000) . Pattern occurrences are then obtained through multiple passes over this condensed representation, which is more efficient than working on the raw data.

- To address the problem of generating (and thus verifying) theoretical candidates that are not present in the data, pattern growth algorithms build their candidates from the condensed data representations. This ensures that any candidate exists in the data, thus making the search for its occurrences meaningful.

The main drawback of pattern growth algorithms is that one needs to have enough memory to store the condensed data representation. Note that most pattern growth algorithms use a depth-first search strategy (Fournier-Viger et al., 2017).

2.2.1 Illustration with PrefixSpan (Pei et al., 2001)

To illustrate the concept of a condensed data representation, the following example considers the projected databases used in the PrefixSpan algorithm, since it is the most commonly encountered pattern growth algorithm to mine sequential patterns. We use the example data that Pei et al. provided in their original paper, presented in table 3.4.

Table 3.4: An example of a sequence database (Pei et al., 2001).

SequenceId	Sequence
1	$\langle a(abc)(ac)d(cf)\rangle$
2	$\langle (ad)c(bc)(ae)\rangle$
3	$\langle (ef)(ab)(df)cb\rangle$
4	$\langle eg(af)cbc\rangle$

Step 1 – PrefixSpan starts with a scan of the database during which it finds the frequent patterns of size 1. In our example, this leads to the following patterns, where $\langle x\rangle : 3$ would mean that the pattern containing event type x has an associated support of 3:

$$\langle a\rangle : 4,\ \langle b\rangle : 4,\ \langle c\rangle : 4,\ \langle d\rangle : 3,\ \langle e\rangle : 3,\ \langle f\rangle : 3.$$

Step 2 – From the list of frequent patterns of size 1, the search space can be divided according to the different prefixes. In our example, we would have six: patterns having prefix $\langle a\rangle$, patterns having prefix $\langle b\rangle$ ... patterns having prefix $\langle f\rangle$.

Step 3 – A pass over the database is performed to build the *projected databases* with regards to the identified prefixes. A sequence α' is said to be a projection of sequence α with regards to prefix β if (1) α' is a subsequence of α, (2) β is a prefix of α' and (3) no other sequence α'' that matches the first two conditions longer than α' exist. Table 3.5 present the projected databases for our example.

***Table 3.5: Projected databases for prefixes** $\langle a\rangle$, $\langle b\rangle$, $\langle c\rangle$, $\langle d\rangle$, $\langle e\rangle$ **and** $\langle f\rangle$. $(\_b)$ indicates that the last item of the prefix is part of the first elements of the sequence.*

Prefix	Projected sequences			
	1'	2'	3'	4'
$\langle a\rangle$	$\langle (abc)(ac)d(cf)\rangle$	$\langle (\_d)c(bc)(ae)\rangle$	$\langle (\_b)(df)cb\rangle$	$\langle (\_f)cbc\rangle$
$\langle b\rangle$	$\langle (\_c)(ac)d(cf)\rangle$	$\langle (\_c)(ae)\rangle$	$\langle (df)cb\rangle$	$\langle c\rangle$
$\langle c\rangle$	$\langle (ac)d(cf)\rangle$	$\langle (bc)(ae)\rangle$	$\langle b\rangle$	$\langle bc\rangle$
$\langle d\rangle$	$\langle (cf)\rangle$	$\langle c(bc)(ae)\rangle$	$\langle (\_f)cb\rangle$	
$\langle e\rangle$			$\langle (\_f)(ab)(df)cb\rangle$	$\langle (af)cbc\rangle$
$\langle f\rangle$			$\langle (ab)(df)cb\rangle$	$\langle cbc\rangle$

The projected databases are then scanned to look for the frequent patterns of size 2. In our example, prefix $\langle a\rangle$ would yield the following patterns:

$$\langle aa \rangle : 2,\ \langle ab \rangle : 4,\ \langle (ab) \rangle : 2,\ \langle ac \rangle : 4,\ \langle ad \rangle : 2,\ \langle af \rangle : 2$$

The procedure is then repeated recursively for all frequent patterns as long as new ones are discovered. ■

Besides PrefixSpan, references to other pattern growth algorithms can be found in table 3.6.

Table 3.6: Non-exhaustive overview of existing pattern growth pattern mining algorithms. *Strategy can be either Breadth- (BFS) or Depth-first search (DFS).*

Algorithm	Reference	Pattern type	Strategy
FreeSpan	Han, Pei, Mortazavi-Asl, et al. (2000)	Sequential patterns	DFS
PrefixSpan	Pei et al. (2001)	Sequential patterns	DFS
SLPMiner	Seno and Karypis (2002)	Sequential patterns	DFS
PROWL	Huang, Chang, and Lin (2004)	Episodes	DFS

2.3 Implementations

When looking at existing software, various implementations of pattern mining algorithms can be found, either independently or integrated into larger systems. While they all offer an API, only some of them provide a graphical user interface. In particular, we can mention the following solutions:

- Weka[2], an open-source machine learning software. Some pattern mining algorithms are implemented, such as GSP. Both a command line and a graphical user interface are provided.
- Knime[3], an open-source data analysis software. Existing modules make the pattern mining algorithms implemented in Weka available to use within a Knime analysis process.
- SPMF[4] (Fournier-Viger et al., 2016), an open-source data mining library offering numerous Java implementations of algorithms, including GSP, SPADE, SPAM, LAPIN-SPAM and PrefixSpan for sequential patterns and Minepi for episodes. A graphical user interface is provided for some of the algorithms.
- R packages[5] extend the basic features of the R programming language. The arulesSequences[6] package provides an interface to a C++ implementation of SPADE. There also

[2] https://www.cs.waikato.ac.nz/ml/weka/
[3] https://www.knime.com/knime-software
[4] https://www.philippe-fournier-viger.com/spmf/
[5] https://www.r-project.org/
[6] https://cran.r-project.org/web/packages/arulesSequences/index.html

exists the TraMineR[7] (Gabadinho, Ritschard, Mueller, & Studer, 2011) package for sequence analysis, although it is specially designed for social sciences.

- The MLlib[8] machine learning library for Spark, which offers Scala and Java implementations of the PrefixSpan (Pei et al., 2001) algorithm.
- The proprietary softwares SAS Enterprise Miner[9] and IBM SPSS[10] both offer a sequential pattern mining procedure without indicating the underlying algorithm.

All of these solutions offer algorithms for itemset and association rule mining. When considering sequential data, they exclusively offer sequential pattern mining, with the notable exception of SPMF that also provides episode mining since June 2018. We were not able to find any other implementation for episode mining. This can be explained by the fact that there exist many ways to count the occurrences of episodes in a sequence, none of them being better than the others (see subsection 3.1.5). In fact, episode mining is not as standardized as itemset mining or sequential pattern mining, which does not favor its integration to existing data science software solutions, or its adoption by analysts. However, episode mining is better suited than sequential pattern mining when taking into account the temporal dimension, which in our opinion makes it a more useful technique to explore sequential data.

3 Conclusion – Towards Progressive Pattern Mining

Sequential Pattern mining is the focus of an active, long-lasting research community. Although the available literature on the matter is large, episode-related techniques and algorithms are only a minor part of it compared to mining sequential patterns over multiple sequences. This disparity is also present when considering existing implementations of Sequential Pattern mining algorithms.

Sequential Pattern mining algorithms output large quantities of results, and adding constraints to the mining process has been proven an efficient way to tune it to target specific patterns. Additionally, counting the occurrences of a Sequential Pattern can be done in several ways, none of them being strictly better than the other. As such, the task of extracting patterns remains highly dependent on the use case it is applied to.

Based on our review of the Progressive Visual Analytics and Sequential Pattern mining literature, we approached the use of Progressive Pattern Mining to explore sequential data from two complementary angles: Sequential Pattern mining within Progressive Visual Analytics, and

[7] http://traminer.unige.ch/
[8] https://spark.apache.org/docs/1.5.0/mllib-frequent-pattern-mining.html
[9] https://www.sas.com/fr_fr/software/enterprise-miner.html
[10] https://www.ibm.com/products/spss-statistics

progressiveness in Sequential Pattern mining. The first is related to the current state of pattern mining in the Progressive Visual Analytics literature, while the second is about the ways in which one could create and interact with a progressive pattern mining algorithm.

3.1 Sequential Pattern mining within Progressive Visual Analytics

As explained in chapter 2, pattern mining was used in Stolper et al. (2014)'s seminal work to illustrate their definition of the Progressive Visual Analytics paradigm. As such, they were more focused on the progressive aspects of their system rather than on pattern mining. The fact that very few subsequent works (Servan-Schreiber, Riondato, & Zgraggen, 2018) have continued to investigate pattern mining in Progressive Visual Analytics makes it a relatively underdeveloped aspect of the progressive paradigm.

All existing pattern mining contributions in Progressive Visual Analytics have in common the fact that they do not provide a way to explore the discovered patterns in the context of the data, effectively turning *data exploration* into *pattern exploration* (Servan-Schreiber et al., 2018; Stolper et al., 2014). While this can be relevant to discover trends and higher-level knowledge, it doesn't allow one to explore the finer information through individual pattern occurrences. In addition, even though patterns are a close representation of the data they are found in, they still are statistical constructs, whose properties differ from the ones of the events present in the data. While the pattern mining community has proposed the notion of episode mining that provides more information on individual occurrences, this has not been adapted to Progressive Visual Analytics. We summarize this in our first challenge for Progressive Pattern Mining:

Challenge PPM1

Investigate progressive pattern mining as a tool for data exploration and not only for pattern exploration.

3.2 Progressiveness in Sequential Pattern mining

In the pattern mining community, algorithms are considered atomic processes that take some data as input, process it according to their parameters and return an output. In order to use such algorithms in Progressive Visual Analytics processes, it is necessary to change this approach if we want to be able to provide the interactions that are at the core of the paradigm. Although existing works from the visualization community have provided some suggestion to introduce interaction within a progressive process, they do not explain how to effectively make an algorithm progressive, and pattern mining has never been their main focus (Badam et al., 2017; Fekete & Primet, 2016; E. G. Hetzler et al., 2005; Mühlbacher et al., 2014; Stolper et

al., 2014). As such, they did not provide a detailed analysis of interactions with a progressive pattern mining algorithm, which leads to our second challenge for Progressive Pattern Mining:

Challenge PPM2

Investigate the ways to make a pattern mining algorithm progressive, the relevant interactions when using such algorithm and how they should be implemented.

PART II

Propositions

Reviewing the literature on both Progressive Visual Analytics and pattern mining led us to identify four challenges for Progressive Pattern Mining, either directly tied to Progressive Visual Analytics or specific to pattern mining in a progressive context.

Challenge PVA1

Clarify what “interaction” means in the context of Progressive Visual Analytics, as well as investigate the role of the algorithm in the process.

Challenge PVA2

Investigate the consequences of interactions between the human and the algorithm on the analysis process.

Challenge PPM1

Investigate progressive pattern mining as a tool for data exploration and not only for pattern exploration.

Challenge PPM2

Investigate the ways to make a pattern mining algorithm progressive, the relevant interactions when using such algorithm and how they should be implemented.

In this part, we present the work we conducted during this PhD to tackle these challenges. We address them by proposing six contributions described as follows, of which we consider four to be major ones.

Major contributions

C1. A study on interaction within a progressive analysis process. This contribution targets our PVA1 and PVA2 challenges, and encompasses the following elements:

- **C1.1.** A framework describing the actions an analyst can perform when interacting with a progressive algorithm;
- **C1.2.** A model of how these actions impact a generic data analysis system;
- **C1.3.** An evaluation of the impact of "progressiveness" on an algorithm's performances.

C2. A clarification of the process of Progressive Visual Analytics, with regards to the role of the algorithm in the process. This contribution targets our PVA1 challenge, and encompasses the following elements:

- **C2.1.** A clear definition for the notion of *steering*;
- **C2.2.** An updated definition of Progressive Visual Analytics, that offers a more precise vision of the interactions that can take place within Progressive Visual Analytics;
- **C2.3.** A model of the Progressive Visual Analytics process, based on a classical Visual Analytics model (Sacha et al., 2014).

C3. PPMT, a progressive pattern mining system. This contribution targets our PPM1 and PPM2 challenges, and encompasses the following elements:

- **C3.1.** The PPMT system itself, that supports the analysis tasks we identified (C5), implements all the actions from our framework (C1.1), and follows our guidelines (C6);
- **C3.2.** A review of PPMT's compliance with existing recommendations for the design of Progressive Visual Analytics systems.

C4. A user experiment focused on the impact of the interactions between the analyst and the algorithm on a progressive analysis process. This contribution targets our PVA2 challenge.

Secondary contributions

C5. A data and task model for an analyst exploring temporal data with patterns, specialized from the general model of Andrienko and Andrienko (2006). This contribution targets our PPM2 challenge.

C6. A set of five guidelines for the design of a progressive algorithm for pattern mining in sequences. This contribution targets our PPM1 challenge.

Outline of this part

Rather than being based on our challenges or on our contributions, the content of this part is organized in a thematic way, as presented in figure 3.7. We open with our theoretical contributions in chapters 4 and 5, the former being focused on our Progressive Visual Analytics challenges (PVA1, PVA2) while the latter addresses our Progressive Pattern Mining challenges (PPM1, PPM2). Chapter 6 relates our practical contributions towards challenges PPM1, PPM2 and PVA2. Finally, chapter 7 contains our experimental contribution towards challenge PVA2.

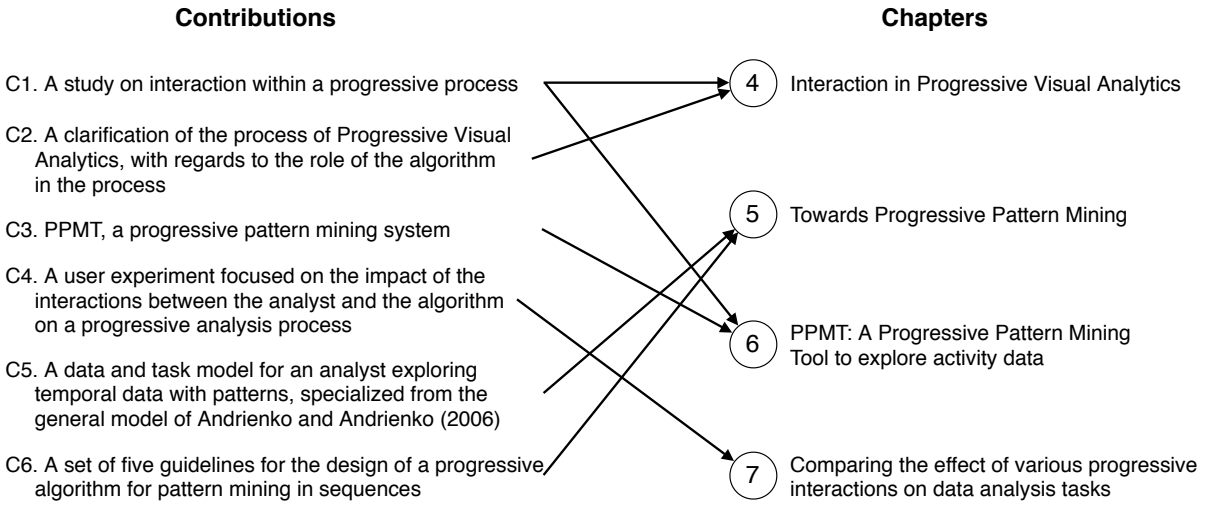
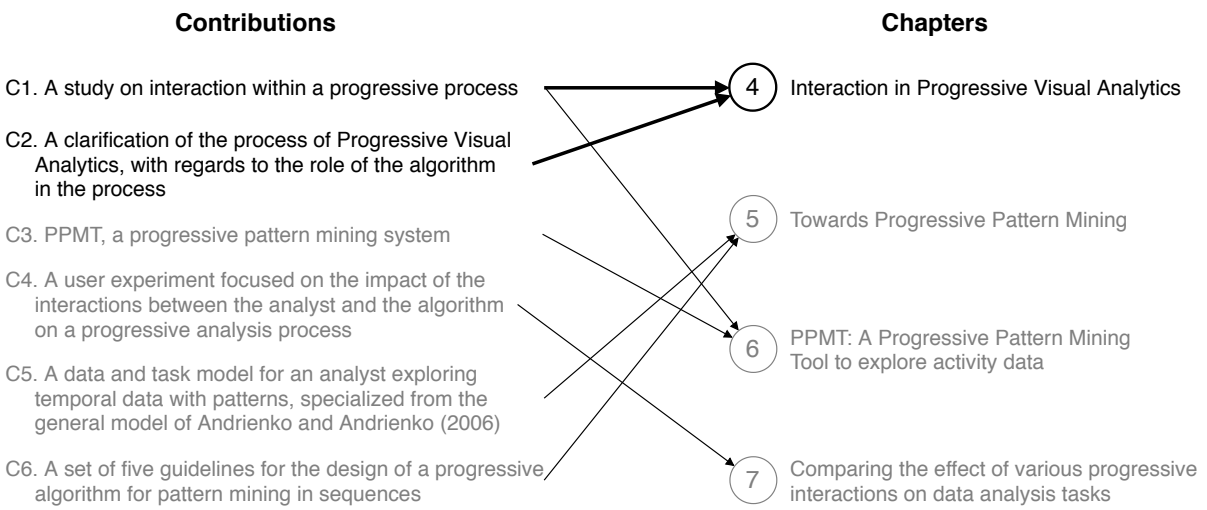


Figure 3.7: Relation between contributions and chapters.

CHAPTER 4

INTERACTIONS IN PROGRESSIVE VISUAL ANALYTICS

As evidenced in our review of the literature, interaction within a progressive workflow can take place in many ways (Badam et al., 2017; Kim et al., 2017; Pezzotti et al., 2017; Schulz et al., 2016; Stolper et al., 2014; Williams & Munzner, 2004). It can be considered either between the analyst and the user interface or between the analyst and the progressive algorithm running in the background, both aspects presenting interesting research questions. In our work, we chose to focus on the second aspect. In this chapter, we relate the theoretical work we conducted on the matter of interactions in Progressive Visual Analytics, which constitutes our first two major contributions.



In section 4.1, we relate our exploration of the notion of interaction between an analyst and a progressive algorithm. We propose a framework of actions an analyst can perform during such interaction (C1.1), and a model of how these actions impact a generic data analysis system (C1.2). The section concludes with a review of existing Progressive Visual Analytics systems seen through the lens of our framework.

Section 4.2 is focused on our work on the concepts of Progressive Visual Analytics. We

first propose a clarification of the meaning of the term *steering* (C2.1), that we then leverage to propose a definition of a Progressive Visual Analytics system (C2.2). Finally, we introduce a model of the Progressive Visual Analytics process, based on an existing model of Visual Analytics (C2.3).

In section 4.3, we discuss some indicators that could be useful to an analyst, to help them guide their analysis.

1 A framework of possible interactions with an algorithm in Progressive Visual Analytics

One of the conclusions we drew from our review of the Progressive Visual Analytics literature was that existing systems have been developed without a concerted view regarding the interactions they provided. This is both due to Progressive Visual Analytics being a recent paradigm that is still developing, and to the fact that each system targets specific data types and use cases. A drawback of this situation is that from one work to another, the same term can be used to designate different actions, which makes comparing existing systems more complicated than it needs to be. The main example of this phenomenon is the term *steering*, systematically used to describe largely different actions, such as prioritizing parts of the data (Pezzotti et al., 2017; Williams & Munzner, 2004), prioritizing parts of the results (Stolper et al., 2014), changing parameter values (Badam et al., 2017) or altering the results (Kim et al., 2017). This semantic confusion has also been noticed by Badam et al. (2017) in the following statement:

> "*[steering] encompasses multiple types of operations, from filtering data if the analyst wants to focus on specific values, to tuning algorithm parameters and behavior dynamically*". (Badam et al., 2017)

At the same time, existing efforts towards a classification of interactions between an analyst and an ongoing computation by Mühlbacher et al. (2014) fail to efficiently describe the variety of actions in existing Progressive Visual Analytics systems, especially when considering the *control* aspect of their model. For example, when considering the actions of changing the data that will be processed by the algorithm or changing the values for the algorithm's parameters, Mühlbacher et al. would consider them to be of the same type (*control over the execution*). Moreover, steering is not clearly located in Mühlbacher et al.'s model, the closest notion being what is called *prioritization*[1], considered a borderline case between execution and result control.

[1] "While the final result [...] is not affected, the purpose is to alter the sequence of intermediate results in order to generate presumably more interesting ones earlier."

1.1 Interactions between an analyst and an algorithm

Since the dimensions used in Mühlbacher et al.'s model are too broad when considering interaction with a progressive algorithm, we decided to formulate our framework in terms of actions and their impact on the Progressive Visual Analytics system at hand. Taking inspiration from Sacha et al. (2014)'s model of the Visual Analytics process (see subsection 2.1.4), we represent said system with the following three elements:

- **Data:** the data that is being explored by the analyst. Depending on the system, it can be the raw data, or a representation of it built for the future needs of the analysis. This can be assimilated to the *data* component from Sacha et al.
- **Algorithm:** The progressive algorithm that processes the previously mentioned data. While not a fully-fledged node in Sacha et al.'s process (it is a transition from the *data* to the *model*), our work targets the interactions with the algorithm. Therefore, we decided to make it a proper part of the system.
- **View:** The visual representations available to the analyst to explore the data, it corresponds to the *visualization* Visual Analytics component from Sacha et al. However, while they emphasized the visual aspect of the process, here we designate everything that is presented to the analyst. This potentially includes both graphical representations and textual information, hence our choice of the term *view*.

We designed our framework based on two factors, the part of the system that is targeted and the way it is done. This gives us the following four actions:

(A1) **Modifying the data** encompasses the different changes that can be done on a dataset, such as adding or removing attributes, data points, or changing their attribute values. These actions are primordial since most of the time in data analysis is spent preprocessing the data (S. Zhang, Zhang, & Yang, 2003). Here, we refer to these actions being performed while the algorithm is running, rather than during the preprocessing step of the analytics process.

(A2) **Constraining the algorithm on data** is a control on the algorithm expressed on the data. For example the analyst from Williams and Munzner (2004)'s system can guide the MDS layout process towards an interesting region of the dataset, the next MDS run taking place over this sub-dataset only.

(A3) **Constraining the algorithm on results** is a control on the algorithm expressed on the (intermediate) results. For example the analyst of Kim et al. (2017)'s system can move

points in the MDS layout so as to influence the MDS execution (e.g. to leave a local optimum). The algorithm will be impacted only in the computations related to the coordinates of the user-modified points.

(A4) Changing parameter values consists in changing the values of the algorithm parameters, that can be original parameters of the analysis algorithm (e.g. number of clusters in k-means, perplexity in t-SNE), or parameters introduced by the system's progressiveness (e.g. data chunk size in Schulz et al. (2016) and Badam et al. (2017)).

These actions are part of the analysis' exploration loop, and target the different parts of the process as illustrated in figure 4.1. *Modify data* acts on the *Data* layer, thus impacting the *Algorithm*'s subsequent execution, which in turns provides new information to the *View*. *Change parameter values*, *constrain the algorithm on data* and *constrain the algorithm on results* target the *Algorithm* layer in different ways, impacting the future states of the *View*. The *Explore* action, while not part of our framework, refers to the Visual Analytics' "Explore view" action, where the analyst is gathering information without actively interacting with the underlying layers of the system (*Data* and *Algorithm*).

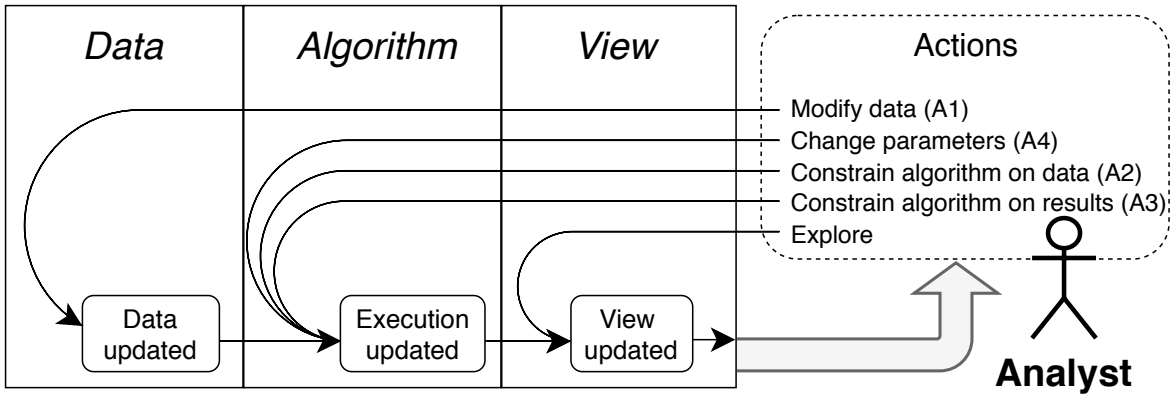


Figure 4.1: Interaction cycle between the analyst and the progressive system. *Thin arrows represent the repercussions of the analyst's actions on the system's layers. Observing the repercussions on the view leads the analyst to perform additional actions.*

1.2 Progressive Visual Analytics systems seen through our framework

With our framework in mind, we looked back at the existing systems in the Progressive Visual Analytics literature in order to describe the various ways in which they handle their interactions between the analyst and the algorithm. We provide here a quick reminder of the interactions these systems allow with their algorithms (see subsection 2.2.3 for a more detailed presentation). For the sake of comparison, we also include our own progressive system, PPMT, which

is presented in more details in chapter 6.

- Badam et al. (2017)'s system is based on successive runs of t-SNE and k-means algorithms over increasingly larger subsets of the data. Analysts are able to change the parameter values, with the new ones taking effect from the next run. While their algorithm is not progressive, the way the system works simulates a progressive analysis environment from the analyst's point of view.
- Kim et al. (2017) propose to manually alter MDS and t-SNE algorithms' output (i.e change the position of points in the low-dimensional space), either permanently or temporarily, to guide the remaining computation until convergence is reached. Merging or splitting selected k-means clusters permanently or temporarily is also possible.
- Pezzotti et al. (2017) use a t-SNE version with approximated computations. Analysts can select a region in the low-dimensional space to decrease the approximation level for the corresponding data points, thus increasing representation precision. They can also add, modify or remove data points on the fly. If they add or remove data attributes, then the algorithm is restarted.
- Schulz et al. (2016)'s similarity search system allows analysts to restart the procedure with new parameters.
- Stolper et al. (2014) allow ProgressiveInsights' users to steer a frequent pattern mining algorithm by prioritizing or excluding patterns with a given prefix, as well as to restart the algorithm over a subset of the data, or with new parameter values.
- Williams and Munzner (2004)'s MDSteer allows selecting a region in the MDS projection plane to focus the computing power on it, *i.e.* on laying out the data points that are likely to be projected into that region.
- Raveneau, Blanchard, and Prié (2018)'s PPMT allows analysts to prioritize the algorithm on a pattern prefix, a data sequence, or a time interval. The algorithm restarts if the analysts change the parameters or modify the data.

In addition to these interactions with their algorithm, these Progressive Visual Analytics systems present incremental visualizations of the results. They support various interactions allowing to explore these visualizations, such as zooming, filtering, ordering and highlighting elements... Additional control may be available, such as tuning density map's rendering parameters (Pezzotti et al., 2017; Schulz et al., 2016) or altering the visualization's evolution speed (Badam et al., 2017; Raveneau et al., 2018; Schulz et al., 2016). However, since they are not related to the algorithm itself, we do not consider these interactions here.

Table 4.1 describes existing Progressive Visual Analytics systems with regards to our framework, showing that most of these systems only support a few action types, while Stolper et al.'s system and our own implement three or more action types. The table also highlights how these systems handle their interactions with the algorithm: either by forcing a *restart*, which stops the algorithmic process, or by what we call the system's *progressive strategy*, *i.e.* how the system handles interaction in a progressive way, which implies that the past computations and results are for the most part not invalidated. We can observe that, except for action A1 *Modify Data* in Pezzotti et al. (2017)'s system, the same choice is made across all systems: actions A2 and A3 *Constrain on Data and Results* are performed through the system's progressive strategy, while actions A1 *Modify Data* and A4 *Change parameter* lead to restarting the algorithm. Additionally, when actions A2 and A3 are available in a system, their respective authors' descriptions involve the words "steer" or "steering".

Table 4.1: Analyst's actions on running algorithms in existing Progressive Visual Analytics systems, and how they are handled. *An empty cell means that the action is not supported by the system. R means that the action leads to restarting the algorithm. A check mark ✓ means that the action is performed via the system's progressive strategy.*

Interaction with:	Data		Results	Parameters
User's action:	A1. Modify	A2. Constrain	A3. Constrain	A4. Change
Badam et al.				R
Kim et al.			✓	
Pezzotti et al.	✓ and R[2]	✓		
Schulz et al.				R
Stolper et al.	R		✓	R
Williams et al.		✓		
Raveneau et al.	R	✓	✓	R

2 An updated definition of Progressive Visual Analytics

Based on the previous observation, we propose to define *steering* as the union of actions A2 and A3 *Constrain the algorithm on Data and Results*, which are at the core of the Progressive Visual Analytics paradigm:

Steering [an algorithm] – An analyst steers the algorithm in a Progressive Visual Analytics system, when 1/ they express a constraint on a subset of the data or of the results, and 2/ they demand that the algorithm satisfies the constraint during the next execution iteration(s), *i.e.* without restarting.

[2]The algorithm restarts only when adding or removing data attributes.

Here, the term “iteration” refers to a pass through the most high-level loop of an algorithm. When considering the systems from table 4.1, these are a data exploration loop (Badam et al.; Schulz et al.; Williams and Munzner), a result space exploration loop (Raveneau et al.; Stolper et al.) or a model refining loop (Kim et al.; Pezzotti et al.).

Some constraints stand only for the next iteration, while others stand for all the future iterations. This has been well defined by Kim et al. (2017) with what they call “soft” and “hard” replacements. However, there exists another type of persistence, in which the constraint stands only as long as undiscovered results that satisfy it remain. This type of persistence can be seen with algorithms exploring a data space or a result space. In Stolper et al. (2014)’s pattern mining system, as well as in our own, the analyst can for example steer the algorithm towards patterns that have a given prefix (A3), thus applying the constraint for several iterations. When satisfying patterns can no longer be found, the system relaxes the constraint and goes back to the rest of the patterns.

The literature sometimes presents systems as Progressive Visual Analytics ones, even though they do not offer ways to interact with the algorithm. Oftentimes, these are systems that provide the analyst with incremental visualizations, such as E. G. Hetzler et al. (2005), Brandes and Pich (2007), Fisher et al. (2012) or Zgraggen et al. (2017). To clarify the notion of what is a Progressive Visual Analytics system and what is not, we propose the following definition:

Progressive Visual Analytics system – A Progressive Visual Analytics system is the combination of three necessary aspects:

- A *progressive algorithm*, returning intermediate results during its execution while offering control over the remaining computations. When possible, the results should be of increasing quality;
- A *progressive visualization* of both the continuous output from the algorithm and the state of the running computation;
- *Interaction means*, that provide the analyst with steering control over the remaining algorithm’s computation (at least actions A2 and A3).

From our framework and our definition of a Progressive Visual Analytics system, we propose a model of the Progressive Visual Analytics process inspired by the existing one for Visual Analytics that we described in section 2.1.4, by Sacha et al. (2014). Our model is illustrated in figure 4.3, while figure 4.2 shows the one by Sacha et al.

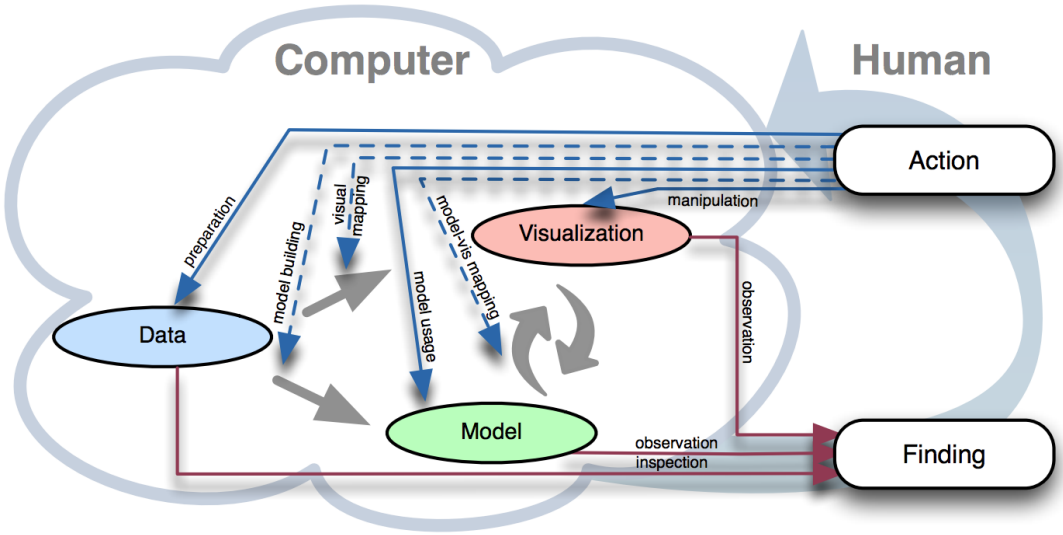


Figure 4.2: Knowledge generation process in Visual Analytics (Sacha et al., 2014). *Human actions are the blue arrows, which can lead to Visual Analytics components (filled arrows) or to the mappings between them (dotted arrows). Red arrows represent the human cognition paths, when generating findings by observing the system.*

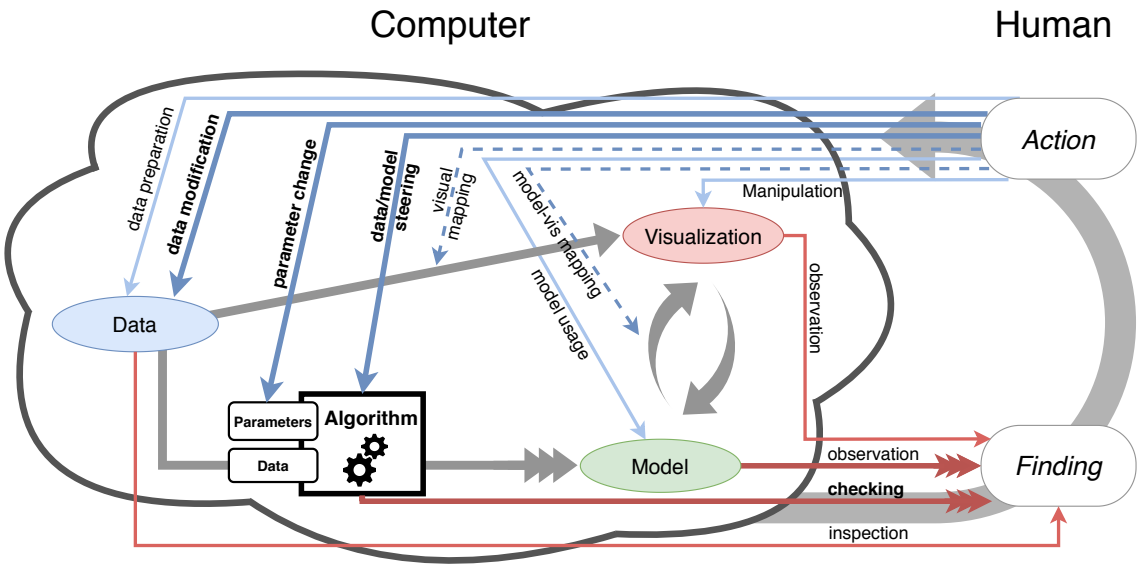


Figure 4.3: Model of the Progressive Visual Analytics process. *The human can act on the analysis' components (continuous blue arrows) or on the mappings between them (dotted blue arrows). He can observe the system's reactions (red arrows) to discover findings. Triple-headed arrows represent the elements communicated in a progressive way by the system. Elements differing from Sacha et al. (2014)'s Visual Analytics model are in* ***bold****.*

The overall process is similar due to both Visual Analytics and Progressive Visual Analytics being structured around an exploration loop involving the human and the system. However, the *Algorithm* has been raised from an unlabeled arrow linking *data* and *model* to a first class element of the process. It takes its input in two ways, the *Data* and its *parameters*. The analyst is able to receive feedback on the algorithm's execution by *checking* its state while it runs. In addition to this, the four actions from our framework are represented. *Data modification* (A1) complements the *data preparation* inherited from Visual Analytics (now limited to data preprocessing before the exploration), while *constrain on data/result* (A2/A3) and *parameter change* (A4) have replaced the *model building* action from Sacha et al. Finally, triple headed arrows indicate the parts of the process that benefit from the intermediate results offered by the Progressive Visual Analytics paradigm.

3 Indicators to guide the analysis

During algorithm execution, the analyst can either decide to keep exploring incoming results, to steer the algorithm in potentially fruitful directions (A2, A3), to modify the data by incorporating results obtained so far (A1), or to change the parameters (A4). From their point of view, one main choice is to decide whether the current results will be lost or not, *i.e.* if the analysis' current algorithmic session will be changed or continued. The first case is related to a lack of success in the exploration process, both intermediate and future results being deemed uninteresting. In the second case the focus is on continuity of the current process with slight shifts and guiding, keeping past computations and results. The decision is also taken with regards to the time it may take to get new results and to finish the action, as well as to the perceived quality of the results if the action is taken.

3.1 Indicators for the analyst

We propose a set of predictive indicators that a Progressive Visual Analytics system could present for each available action, at each step of the current execution, to help the analyst choose the next action to perform:

I1: the expected time before the next result update;

I2: the expected time to finish processing the consequences of the action. Depending on the way it is implemented and the action considered, this could target the moment the action is complete, or the moment it has provided a sufficient result;

I3: the expected quality from forthcoming results;

I4: the continuity of the results, *i.e.* how much the current results will be changed.

These categories of indicators are rather general, and apply to any action (A1 to A4). However their form may differ depending on the action at hand, and on the considered algorithm. The expected time before the next result (I1) update may be influenced by memory load. The expected quality (I3) can either address the quality of the subset of results that are to be expected from steering the algorithm, or the general quality of all the results after a parameter change. Continuity of results (I4) apply for instance on images (data projections) for MDS or t-SNE, and on sets of patterns and occurrences for pattern mining. The expected time to finish an action (I2) is based on different characteristics in pattern mining (size of search space) and with k-means (model stability). Using representations of the confidence bounds as done by Fisher et al. (2012) could be a way to deal with the inherent uncertainty when displaying such indicators.

Some of these indicator categories are related to requirements for Progressive Visual Analytics systems (see subsection 2.2.4.3). Here we refer to Badam et al. (2017) since they proposed the most exhaustive requirement compilation. The expected time before the next result update (I1) is related to the R13 requirement ("Support two modes: on-demand refresh and constant update") and the associated update speed, since the indicator could provide guidance in whether or not the "on-demand refresh" mode will be needed. For example, let us consider the case of an analyst that wants to explore the current state of a data visualization in order to investigate some hypothesis. If the estimated time before the next update is short, such as a couple of seconds, they might want to use the "on-demand refresh" mode to work with a stable visualization. On the other hand, if the next update is not to be expected before a few minutes, they can explore the visualization using the "constant update" mode without risk of being disturbed by a visualization update during the process. R10-11 ("provide both absolute and relative progress of the execution") and R6-7 ("provide aggregated information and display uncertainty") are concerned with execution time (or more generally process progress) and result quality. However they focus on information delivery at present time, while our I2 and I3 indicators emphasize the benefits of a predictive estimation to assist decision-making. To our knowledge the predictive I4 indicator has not been considered in previous works.

Though approximate, these predictive indicators should prove useful during the unfolding of progressive analytics processes. As in any visual analytics system, the Progressive Visual Analytics analyst should be able to proceed with trial and errors, especially considering that intermediate results address the main disadvantage of trial and errors, that is waiting to see whether the current try leads to better outcome or not. This stresses the need for undo-redo functionalities in Progressive Visual Analytics system, a topic that has been mentioned by Stolper et al. (2014), and a potential new requirement for Progressive Visual Analytics systems.

3.2 Indicators in existing systems

Since they are broad enough to apply to any action from our framework, implementations of these indicators can vary greatly from one system to another. When considering existing Progressive Visual Analytics systems, we did not found any implementations available, although some of the systems provide estimates of the current result quality. Examples are displaying a measure of projection error (Badam et al., 2017; Williams & Munzner, 2004), offering different levels of approximation (Pezzotti et al., 2017), introducing convergence measures (Kim et al., 2017) or measuring the execution progress (Badam et al., 2017; Raveneau et al., 2018). If they were estimates of the future results, these features would have satisfied indicator *I3*. It is important to note that such indicators are highly dependent on the algorithm in use and on the system's inner workings. For example, an estimate of the remaining time before the next result (*I1*) can be affected by the system's current memory load. As such, the presented value could fluctuate depending on the creation or termination of other unrelated demanding processes.

4 Conclusion

Our review of the literature highlighted the fact that existing works have proposed different ways to interact with their algorithms, which makes defining the notion of "interaction" in Progressive Visual Analytics complicated. This is most clearly evidenced with regards to the idea of *steering*, as noted by Badam et al. (2017). In this chapter, we presented our work towards clarifying what *interaction* can mean within this paradigm. Our main contribution on this matter is a framework containing four kinds of interactions, that we believe covers all possible cases. We were able to look back at the existing literature through its lens, which led us to propose a clear definition for *progressive systems* and for the notion of *steering the algorithm*. Evaluating a framework is a complicated matter, but considering that we successfully used ours to both describe existing systems and generate new knowledge, its usefulness appears to be validated.

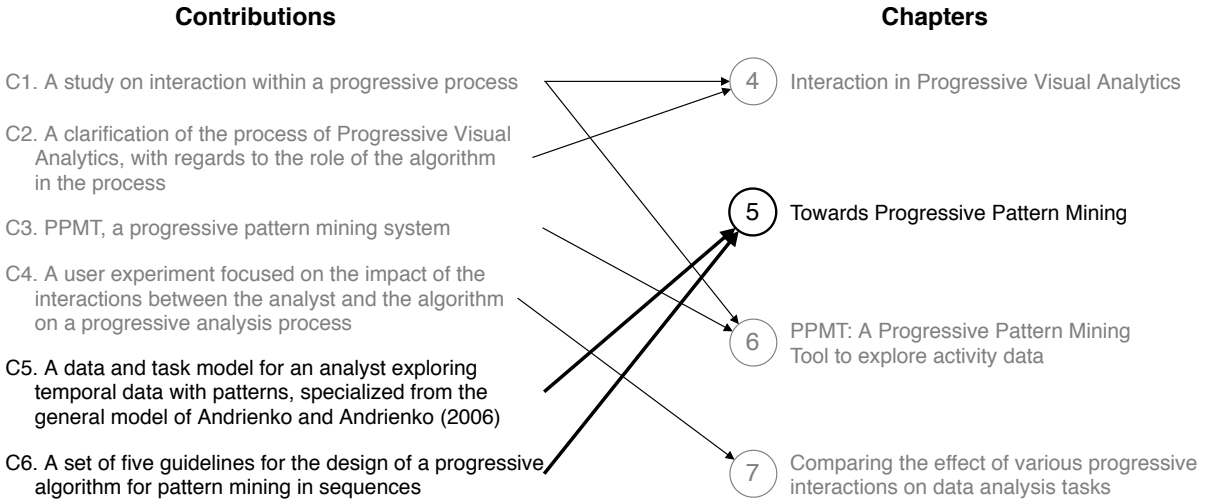
Having a clearer vision of interactions allowed us to identify several ways in which they can impact the intermediate result inherent to progressive algorithms, which will be useful for the design of future systems. We also leveraged this knowledge to propose a visual model of the Progressive Visual Analytics workflow, along with a list of indicators that could be useful to an analyst when available in progressive tools.

These theoretical contributions, while able to stand by themselves, are an important basis for the practical work we present in chapter 6. They allowed us to design and implement an efficient progressive pattern mining tool, and should be useful to the community. On the other hand, the work we conducted on indicators to guide an analyst is only a start. As such, further investigation and experimentation are needed to provide a complete perspective on this matter, with regards to their feasibility and how they can impact the data analysis process.

CHAPTER 5

TOWARDS PROGRESSIVE PATTERN MINING

As evidenced in the state of the art, adding progressiveness to a process is not a trivial change. In this chapter, we relate the theoretical work we conducted on the matter of Progressive Pattern Mining. Our main interest was directed at how Progressive Pattern Mining algorithms can be developed, which begs the question of what analysts may want to use them for. We focused on how progressiveness can impact the defining aspects of sequential pattern mining[1], especially with regards to the analysis tasks that are usually performed using sequential patterns. This constitutes our two secondary contributions.



Section 5.1 is about our investigation of the analysis tasks that can be performed when using a Progressive Pattern Mining algorithm. We relate our reasoning for choosing the existing model from Andrienko and Andrienko (2006) as a basis. We then provide a detailed presentation of their model, before introducing our own version, specialized for the use of pattern mining techniques (C5).

[1] In this chapter, the term "sequential pattern" refers to the general notion of "patterns within sequences" (see the **Note** on Sequential Patterns in chapter 3, page 54)

In section 5.2, we build upon our model and our other works to propose a set of guidelines for the design of a Progressive Pattern Mining algorithm (C6).

1 Analysis tasks performed with patterns

In this section, we present our work on the tasks that an analyst using pattern mining techniques may want to perform. We relate our reasoning for choosing Andrienko and Andrienko (2006)'s model among all the ones we presented in our review of the literature (see subsection 2.1.5.3) and offer a more detailed description of their original model, before presenting a specialized version that is specific to data exploration tasks involving sequential pattern mining.

1.1 Choosing a task model

To investigate the tasks performed with pattern mining, we were looking for a model being both exhaustive and built with temporal data in mind. The exhaustiveness (with regards to the tasks it encompasses) was a direct requirement from our goal of considering all the tasks that can be performed with a non-progressive algorithm, and filtered out all models focused on a specific analysis technique, such as interacting with a visualization. The requirement for a model suited to temporal data came from the nature of sequential patterns, so as to give us the ability to deal with the specificities of the temporal dimension. For these reasons, we decided to build on the model proposed by Andrienko and Andrienko (2006). They propose a general high-level model designed to express the tasks an analyst may perform on temporal data that we specialize for the specific case of working with sequential patterns. In addition to the rapid presentation we provided in subsection 2.1.5.3, the next two subsections offer a more in-depth presentation of their model, followed by our specialization.

1.2 Andrienko and Andrienko (2006): data model

In their work, Andrienko and Andrienko consider that a dataset is composed of multiple *data records*, each of these records being made of the same set of *variables*, that take their *values* from the variable's *domain*. They distinguish two kinds of variables:

- The *referrers* are the variables that describe the context in which the data records were collected. Their values are called *references*.
- The *attributes* are the variables that represent measurements, observations and calculations performed in the context described by the referrers. Their values are called *characteristics*.

For every combination of values of the referrers, there is at most one combination of values for the attributes. Thus, the data representation can be seen as a data function f from $R = R_1 \times R_2 \times \ldots R_p$ to $C = C_1 \times C_2 \times \ldots C_q$ where R_i is the value domain of the ith referrer and C_i is the value domain of the ith attribute (*i.e.* the set of possible characteristics). Figure 5.1 illustrates this model.

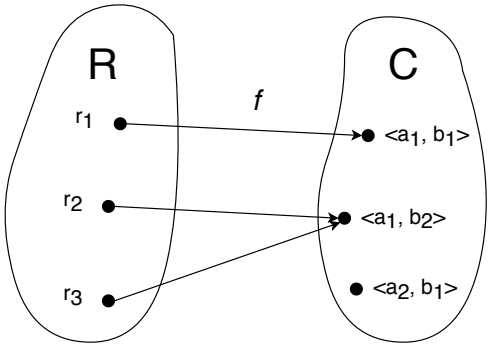


Figure 5.1: The data model of Andrienko and Andrienko (2006). *R is the set of references (only one referrer in this example), and C is the set of characteristics (two attributes a and b in this example). The transition function f links the references to the corresponding characteristics.*

Andrienko and Andrienko provide the example of a dataset containing records of the population numbers in municipalities of a country over the years. In this case, each record has several variables: *municipality*, whose domain is existing municipality names, *year*, *number of persons* and *number of unemployed persons* whose domains are positive integer numbers, and *employment rate*, whose domain is a percentage. Here, *municipality* and *year* are the referrers, that provide the context in which the measurements took place, and the other elements are attributes, that are measured each year. As such, the data representation can be seen as a function from *Municipality* $\times$ *Year* to *Number of persons* $\times$ *Number of unemployed persons* $\times$ *Employment rate*.

1.3 Andrienko and Andrienko (2006): task model

By "task", Andrienko and Andrienko refer to an analyst looking for some information. Each task contains some known parts (that drive the search), some unknown parts (that need to be searched for) and a target, which is the information the analyst is looking for. Depending on the task, the target can either be identical to the unknown part, or only a subset of it. Based on the previous data model, Andrienko and Andrienko identify eleven task categories, organized as presented in figure 5.2. They divide all possible tasks between two categories, the *elementary tasks* and the *synoptic tasks*, although they note that elementary tasks only play a marginal role in exploratory data analysis, due to their extreme simplicity.

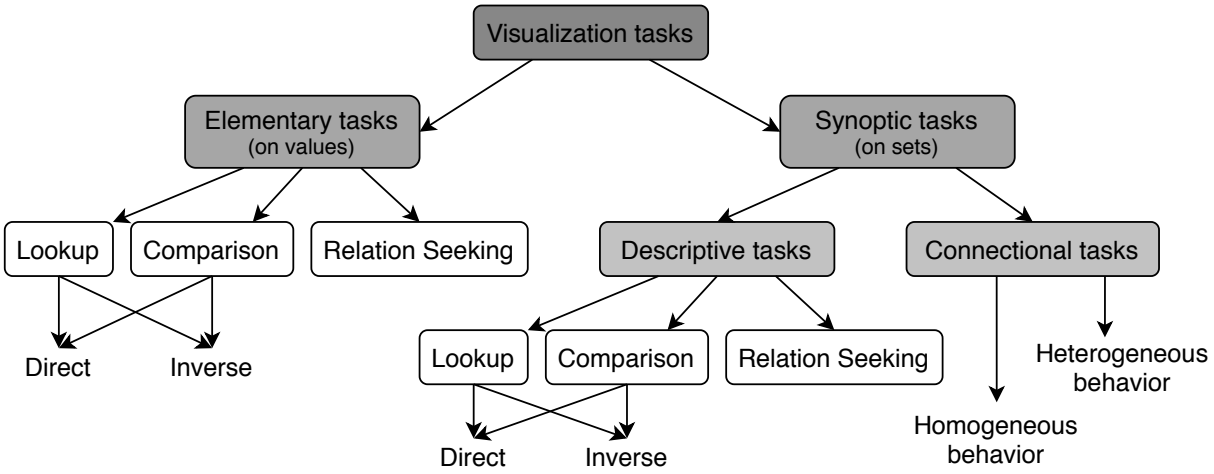


Figure 5.2: The task model of Andrienko and Andrienko (2006). *Illustration by Aigner, Miksch, Schumann, and Tominski (2011).*

1.3.1 Elementary tasks

Elementary tasks refer to individual elements of data, either *references* or *characteristics*. Their target can either be one or more *characteristics*, or one or more *references*. However, if the task involves more than one reference, each of them must be considered individually. They divide these tasks in three categories, that we illustrate using the dataset presented in the *data model* subsection.

Lookup tasks involve a search for information, and are divided in two categories. *Direct lookup tasks* designate tasks where the references are known and characteristics are the target. An example of such task could be "What is the *number of unemployed persons* in the *municipality* X in *year* Y?". *Inverse lookup tasks* represent the cases where the characteristics are known and one is looking for some or all of the corresponding references. An example of such task could be "Which *municipalities* have an *employment rate* of 80% in *year* X?".

Comparison tasks involve questions about the relations that exist between references or between characteristics, where the relation is the target. For Andrienko and Andrienko, a "comparison" is "[the] identification of the kind of relation existing between two or more elements of some set and, whenever permitted by the properties of the set, numerical specification of this relation on the basis of distances or ratios between the elements". Comparison tasks involve performing a lookup task before comparing the result of the lookup with either another lookup's result or a given value. Depending on whether it is a direct or an inverse lookup, the comparison will respectively be a *direct comparison* or an *inverse comparison*. An example of a direct comparison task could be "How does the *unemployment rate* of *municipality* X compare to that of *municipality* Y in *year* Z?" (underlying direct lookups are needed to find the *unemployment*

rate of both *municipalities*, that can then be compared). An example of an inverse comparison task could be "How did the number of *municipalities* having an *employment rate* below 80% evolve between *year* X and *year* Y?" (underlying inverse lookups are needed to find the lists of *municipalities*, whose sizes can then be compared).

Relation seeking tasks also involve questions about the relations that exist between references or between characteristics, but where the relation is already known. The target of the task is therefore part of either the references or the characteristics. In other words, the analyst is looking for *occurrences* of the specified relation. An example of such task could be "In which *municipalities* did the *employment rate* increase from *year* X to *year* Y?" (the "increase of *employment rate* from *year* X to *year* Y" relation is known, and one wants to find *municipalities* in which it occurs).

1.3.2 Synoptic tasks

To describe these tasks, Andrienko and Andrienko introduce the notion of "behaviors" of the data function f (Andrienko and Andrienko (2006), p. 83), a general term that encompasses notions such as distributions, variations, trends... With regard to the data model, a behavior is an abstract configuration of *characteristics* corresponding to a given *reference* set. Figure 5.3 provides an illustration of this notion, using the following inventory dataset:

Product	Time	Quantity
A	January	1000
A	February	900
A	March	1200
A	April	1000
A	May	1400

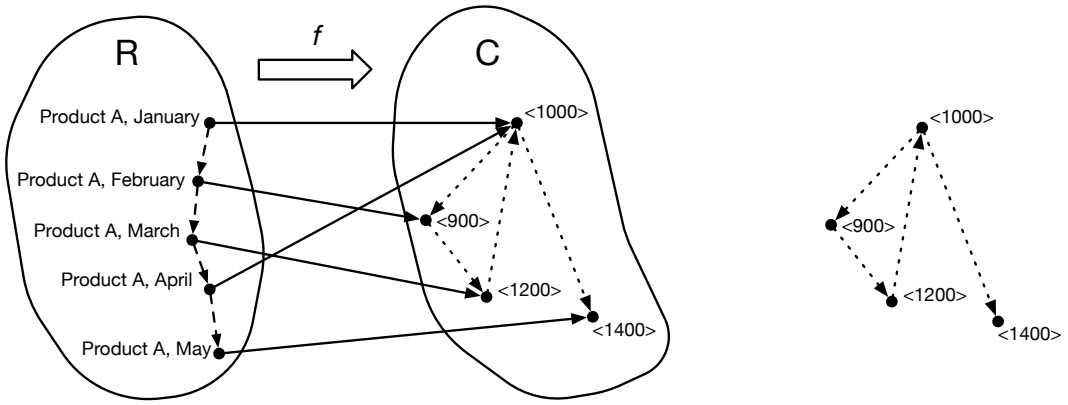


Figure 5.3: Abstract representation of a behavior. *R is the set of references (Product and Time), and C is the set of characteristics (Quantity). On the right, an abstract representation of the behavior of the data function f over R, considering that the references are ordered by their temporal dimension.*

Another important notion when considering synoptic tasks is that of a *structure*[2]. Given a behavior, a structure is an interpretation of the behavior by the observer that reflects its essential features "*in a substantially shorter and simpler way than specifying every reference and the corresponding characteristics*" (Andrienko and Andrienko (2006), p. 85). Andrienko and Andrienko describe the relation between a behavior and its structures as the structure *approximating* the behavior. Essentially, a behavior can be described as a data template while a structure is the result of an observation of said template of the data. As such, the same behavior can lead to several structures, depending on the analyst knowledge, the precision required... If we consider the behavior from figure 5.3, structures such as "an overall increase" or "a succession of decrease then increase" could be observed. Note however that behaviors and structures are not necessarily about the temporal dimension, but may be related to any other referrer, such as the spatial dimension for example.

In essence, synoptic tasks deal with sets as a whole, and as such involve subsets of the *references* (possibly all of them), behaviors, and relations between them. Synoptic tasks are divided in two sub-categories, the *descriptive* and *connectional* tasks (see figure 5.2). Descriptive tasks (lookup, comparison and relation seeking) echo the analysts' actions described for the *elementary tasks*, but applied to behaviors (as such, they can be considered as behavior characterization tasks), while connectional tasks are specific to the synoptic category.

Synoptic lookup tasks are similar to elementary lookup tasks, in that they are divided between the *direct* and *inverse* lookup tasks. *Direct synoptic lookup* tasks involve specifying a set of references and a behavior, in order to find structures that approximate the behavior of the characteristics. An example of such task could be "What is the evolution of the *employment rate* for *municipality* X over time?" (The references are a *municipality* and a set of values for the *year* referrer, and we want to describe the evolution of the associated characteristics). On the other hand, *inverse synoptic lookup* tasks involve specifying a structure in order to find sets of references with the behavior over those corresponding to the structure. An example of such task could be "Find the time intervals in which the *employment rate* increased" (the structure is "increase of the *employment rate*", over the *year* referrer).

Synoptic comparison tasks deal with relations between behaviors, and as such between the structures approximating these behaviors. This means determining how two behaviors are related, *i.e.* how they are similar and how they are different. *Direct synoptic comparison* tasks target the relation between behaviors, while *inverse synoptic comparison* tasks refers to the tasks where one is trying to determine the relations between sets of references corresponding to specific behaviors or structures, or determining relations between reference sets and individual references.

[2] In their work, Andrienko and Andrienko use the term "pattern". To avoid any confusion between this and the pattern mining context our work belongs to, we use the term "structure" instead.

Synoptic relation seeking tasks involve finding occurrences of specific relations between behaviors, and determining the corresponding references sets. The authors state that said relations can be *same*, *different*, *opposite* or even more precise cases of similarities and differences.

Synoptic connectional tasks refer to the cases when an analyst is trying to find connection structures that approximate some mutual behavior, either between different phenomena or between parts of the same phenomenon. A distinction is made between homogeneous and heterogeneous behaviors, with both being subdivided the same way elementary tasks and synoptic descriptive tasks are (lookup, comparison and relation seeking).

1.3.3 Formal description of the tasks and completeness of the model

Andrienko and Andrienko provide a formal description of the analyst's tasks using mathematical functions. Considering the representation in figure 5.1, where R is the set of references and C the set of characteristics, any data subset can be expressed with a notation of the form $f(x) = y$, with x being a subset of R and y a subset of C. Since a task is a search for information under certain constraints, its formal representation takes the aforementioned form, augmented with the missing parts that constitute the search target. These are located at the start between a question mark and a colon, as illustrated in formula 5.1, where one is looking for the attributes (y) corresponding to the given referrer (r).

$$?y : f(r) = y \tag{5.1}$$

Additional symbols are introduced when needed to represent behaviors, structures and relations. Formula 5.2 illustrates the behavior comparison task where two behaviors (β_1 and β_2) are specified and one is looking for 1/ the structures (s_1 and s_2) that approximate ($\approx$) these behaviors ($\beta_1 \approx s_1; \beta_2 \approx s_2$), and 2/ the relation ($\lambda$) between the structures.

$$?s_1, s_2, \lambda : \beta_1 \approx s_1; \beta_2 \approx s_2; s_1 \lambda s_2 \tag{5.2}$$

Since the different task categories of their model have been identified by enumerating all the possible ways to introduce unknown elements in the general formulation of the task, Andrienko and Andrienko are confident in saying that the model is complete with regards to the different types of analysis tasks.

1.4 Data model for sequential patterns

The first step in specializing Andrienko and Andrienko's model was to adapt the data model used to express the various task categories. From our description in chapter 3, a sequential pattern mining algorithm's first purpose is to search for occurrences of patterns within sequences. In order to allow our specialized model to represent tasks involving both the pattern itself and specific occurrences of the pattern, we identify three referrers and one "main" attribute:

- Referrers: *sequence*, *time* and *pattern*
- Attribute: *occurrence*

Values for the *sequences*, *patterns* and *occurrences* can be seen as objects, having an ID associated to a set of properties. This allows one to either reference specific values (using their IDs) or describe some combination of properties. Possible examples could be sequences of a given length, patterns satisfying a given syntax or interestingness for the analyst, and occurrences of a specific duration.

The last referrer, *time*, behaves differently. It is a scale (either continuous or discrete) that can also have associated properties (day/night, weekday or not, season...). Although our work in this section only takes into account instantaneous events, we do not see any meaningful change if ranges were used for the *time* referrer.

Following Andrienko and Andrienko's methodology, we provide the following formal notation for our data model for pattern mining tasks:

$$d \left| \begin{array}{ccc} \mathcal{P} \times \mathcal{S} \times \mathcal{T} & \rightarrow & \mathcal{O} \\ (p, s, t) & \mapsto & o \end{array} \right. \quad \text{if } o \text{ is an occurrence of } p \text{ in } s \text{ at time } t$$[3]

In the above notation, $\mathcal{T}$ is the time referrer, $\mathcal{S}$ is the set of data sequences, $\mathcal{P}$ is the set of all the patterns generated by the mining algorithm, and $\mathcal{O}$ is the set of all the occurrences identified by the mining algorithm. Note that this representation encompasses not only the patterns but also the whole dataset since data events can be seen as patterns of size 1. Therefore, the representation can be used to formulate questions about both the dataset and the patterns that can be found inside. Figure 5.4 provides an illustration of this notation, with the d function being used to describe the three occurrences of pattern `< B ; C >`.

[3] Here, t can be the start of the occurrence, its end, or any timestamp in between

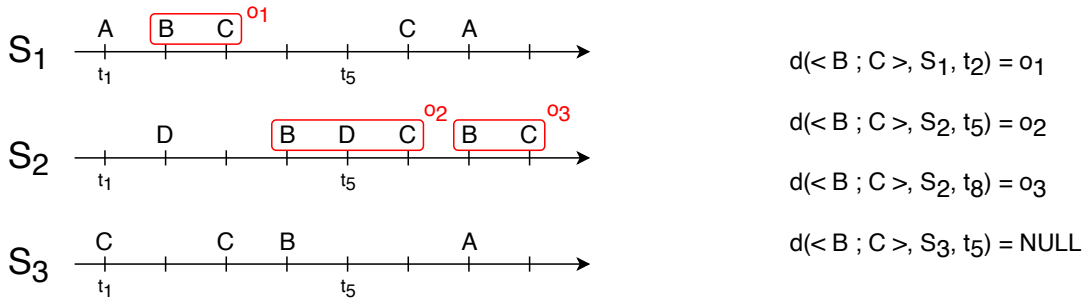


Figure 5.4: Example of a small sequence dataset (left). *Examples of instantiation for the data function* d *are given for the occurrencs of pattern* $< B ; C >$ *(right). A fourth example is present, describing an absence of occurrence.*

1.5 Task model for patterns

Since Andrienko and Andrienko's task model is exhaustive with regards to the way they formulate data analysis tasks, we focused on verifying that all 11 categories are applicable to pattern mining. While they technically are, due to the fact that we can derive them from our data model which is a specialization of Andrienko and Andrienko's, we wanted to make sure that every task category is relevant for a pattern mining driven data exploration. We ended up with the examples provided in table 5.1.

As illustrated by table 5.1, all 11 task categories are valid when considering pattern mining tasks. However, a strong dichotomy between the referrers is highlighted by some of the categories, when a distribution over the *patterns* is involved. The *time* and *sequence* referrers are respectively a metric space and a population of elements that can reasonably be considered independent. As such, they allow one to consider distributions over their domains. Typical examples include observing some phenomenon *over a given time-period*, or searching for *the sequences that verify a given property*. The same cannot be said of the *pattern* referrer, due to the fact that patterns are obviously not independent, since the larger ones are built from the shorter ones. As such, some inclusion relations occur among them when considering their syntax, introducing a bias into any distribution built on the *pattern* referrer. This mainly affects synoptic tasks, which by essence involve distributions (behaviors and structures) over the referrers. Based on this observation, we claim that the synoptic tasks **involving a distribution over patterns** are not rigorously correct in the general case, and that any conclusion based on their result may be misleading in practice. This is especially true if the pattern population is not narrowly selected to avoid inclusion relations among patterns, which are a consequence of their combinatorial nature. It is however important to note that synoptic tasks remain valid for distributions over the other referrers (*time* and *sequences*).

Table 5.1: *Examples of tasks on sequential patterns*

Elementary tasks		
Lookup	**Direct**	• What is the duration of the occurrence of pattern *AB* that happens at time *45* in sequence S_1? • What is the length of the occurrence of pattern *BD* that happens at time *3* in sequence S_2?
	Inverse	• In which sequences does pattern *BCD* occur? • Which patterns occur in sequence S_1 at time *3*?
Comparison	**Direct**	• What is the relation between the duration of the occurrences of pattern *DAB* that happen respectively at times t_1 and t_2 in sequence *S*?
	Inverse	• Compare the sequence that has the longest occurrence of pattern *ADB* with the sequence that has the shortest occurrence of the same pattern.
Relation seeking		• Which patterns have twice the number of occurrences of pattern *BCD*?
Synoptic tasks		
Lookup	**Direct**	• Are the occurrences of pattern *ABC* uniformly distributed over the sequences?
	Inverse	• In sequence *15*, during which time intervals does the pattern *ABC* occur regularly each 3 days?
Comparison	**Direct**	• Compare the temporal distributions of the occurrences of pattern *ABC* between sequences *1* and *2*.
	Inverse	• For the sequences of the *control group*, compare the periods in which *ABC* collapses with the periods in which *ADE* rises.
Relation seeking		• In which sequences is the temporal distribution of *ABC* occurrences identical to the one in sequence *5*?
Connectional		• Is the activity in sequence *45* influencing the activity in sequence *71*?

1.6 Leveraging our task model

Being exhaustive in its categorization of analysis tasks, our model can be used as a guide during the design of a temporal data analysis system using pattern mining. First, the data model can be leveraged as a basis for the data structure design. Secondly, the task model can be seen as a cartography of tasks, from which one can choose the features that need to be offered by the system. For the same reason, our model can also be used when one needs to choose an analysis tool to perform a specific task. In this case, the model's cartography of tasks is used as a reference frame to compare several systems, in order to decide which solution is the most appropriate to perform the analysis. However, this presupposes the availability of a description of several systems in terms of the categories of our task model. Such description should ideally be proposed by the system's authors, or by users that are sufficiently familiar

with the system's features and inner workings. In this thesis, we propose such a description of PPMT, our analysis tool, in subsection 6.2.2. We also used our task model to formulate guidelines for the design of Progressive Pattern Mining algorithms that were then implemented in PPMT.

2 Guidelines for Progressive Pattern Mining algorithms

The literature already contains some requirements and guidelines for the design of efficient Progressive Visual Analytics systems, as presented in subsection 2.2.4.3. Instead, we focused on providing guidelines specific to progressive pattern mining algorithms, based on the pattern-oriented task model described in the previous section and on our experience in working with patterns. We end up with the following 5 guidelines, with guidelines G1, G2 and G4 being directly based on our pattern-oriented model.

Extract episodes (G1)

Our pattern-oriented task model shows that occurrences play a key role when exploring temporal data with pattern mining techniques. However, the majority of existing sequential pattern mining approaches (Mooney & Roddick, 2013) adopt the vision from Agrawal and Srikant (1995), which focuses on the sequences in which a pattern is found rather than where the pattern occurs in these sequences. To be able to leverage the individual occurrences of patterns, using episode mining as introduced in Mannila et al. (1995)'s work seems more suited to the exploration of a temporal dataset. Doing so, the analyst is able to explore and analyze the context in which pattern occurrences are discovered.

When adapting an existing algorithm, one needs to consider the fact that episode mining deals with a single sequence. As such, one needs to either adapt an episode mining algorithm to deal with several sequences (by adding the occurrences found in each sequence), or adapt a sequential pattern mining algorithm to extract all the occurrences of a pattern that can be found in each sequence. While the first solution is easier to implement, it should be noted that most open source pattern mining implementations extract sequential patterns rather than episodes.

Save occurrences (G2)

The output of a pattern mining task is a set of frequent patterns and their number of occurrences. To be able to explore the context in which these occurrences are found, and to be able to perform the different categories of tasks we evidenced in our model, one needs to save the pattern occurrences. It should however be noted that this is likely to largely increase the memory needs of the algorithm.

Use a breadth-first Apriori-like strategy (G3)

Progressive Visual Analytics algorithms rely on intermediate results to provide the analyst with early and constant feedback during the computation. This aspect of the paradigm makes Apriori-like strategies better candidates than pattern growth ones for a progressive pattern mining algorithms. This comes from the fact that Apriori-like algorithms start generating candidates and verifying them (thus potentially discovering frequent patterns) right from the start of the computation, whereas pattern growth algorithms first need to build their condensed representation of the data before considering extracting patterns from it. In addition to this recommendation, Stolper et al. (2014) already identified that breadth-first strategies are more interesting than depth-first ones when dealing with progressive pattern mining. Their reasoning is that, since short patterns are the building blocks of the longer ones, it is more useful to focus the early stages of the computation on their extraction, thus providing the analyst with a better overview of what can be expected later on.

Offer steering on patterns, sequences and time (G4)

The data model presented in the previous section makes the referrers the keys to access the corresponding attributes. As such, an ideal progressive pattern mining algorithm steering should be based on referrer values. In practice, this means that such an algorithm should support steering its execution on patterns, sequences, time periods, or any combination of these dimensions.

Steering on sequences or time induces a constraint on pattern occurrences. This constraint reduces the number of valid occurrences, which then reduces the number of generated patterns thanks to the minimum support threshold constraint. Steering on patterns is more direct since the compliance check does not impose to read the data. Having a pattern's syntax is enough information to know whether the pattern complies with the steering target, and thus decide if its occurrences should be extracted or not.

Provide information on the algorithm activity (G5)

Existing guidelines for Progressive Visual Analytics systems already highlight the fact that an analyst needs information about what an algorithm has already done, what it is doing and what remains to be done, in order to efficiently interact with it. When considering pattern mining algorithms, important information are the currently explored pattern size, the number of candidates of this size (and how many have already been checked), and whether steering is occurring or not. In case of an ongoing steering, knowing its target is also an important information. Beyond these core elements, other information can be useful, for example an estimate of the speed at which the algorithm is running its search for frequent patterns.

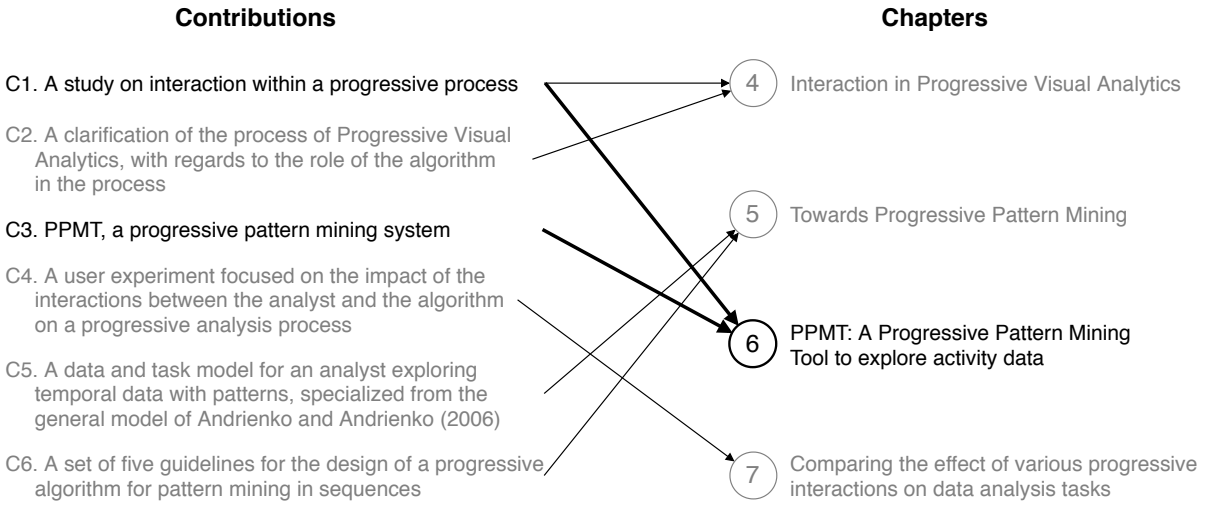
3 Conclusion

In this chapter, we presented Andrienko and Andrienko (2006)'s data analysis task model, and specialized it for the case where said analysis is performed using sequential pattern mining techniques. We evidenced an important difference between the *pattern* and the other two referrers of our data model (*sequence* and *time*), in that considering distributions over the patterns can be misleading. This had consequences on the design of PPMT, presented in the next chapter, where we prioritized *time* and *sequences* to structure our visualizations. In a second section, we introduced five guidelines for the design of progressive pattern mining algorithms, to be used alongside existing recommendations for the design of progressive systems.

CHAPTER 6

PPMT: A PROGRESSIVE PATTERN MINING TOOL TO EXPLORE ACTIVITY DATA

In this chapter, we present our practical work towards implementing a system that allows an analyst to explore activity data, using pattern mining[1] techniques within the paradigm of Progressive Visual Analytics. This Progressive Pattern Mining Tool, which we refer to by its acronym PPMT, constitutes our third major contribution. Since it heavily relies on the algorithm we implemented in PPMT, this chapter also contains the last component of our first major contribution.



The first five sections of this chapter form our description of PPMT (C3.1). Section 6.1 describes our design process and focuses on the main design choices we made. Section 6.2 details the features of PPMT, while section 6.3 contains descriptions of the various widgets that compose its user interface. Sections 6.4 and 6.5 are dedicated to the internal aspects of PPMT, respectively detailing its architecture (both theoretical and our implementation) and describing

[1] As explained in section 6.2, PPMT only supports episode mining. As such, in this chapter the terms "pattern" and "sequential pattern" refer to the notion of *episodes*, as described in subsection 3.1.3

the pattern mining algorithm at work.

In section 6.6, we relate two evaluations we conducted on PPMT. The first is a review of existing recommendations for Progressive Visual Analytics systems, and how PPMT fares with regards to implementing them (C3.2). The second evaluation is focused on the Progressive Pattern Mining algorithm, in order to study the impact of progressiveness on its performances (C1.3).

PPMT is free open-source software, distributed under the GPLv3 license at https://github.com/VRaveneau/PPMT. An instance of PPMT is available at http://ppmt.univ-nantes.fr/ppmt. It contains the *coconotes* dataset we used during our work, that is described in subsection 6.1.3. While the version of PPMT provided here should work in all modern browsers, we recommend the use of Firefox.

Note: In this chapter, we provide images of PPMT being used during the analysis of human activity data. As such, each data sequence corresponds to a single user, and we use the term "user" to refer to what is called a "sequence" in the literature and in our data model presented in chapter 5. For the sake of clarity, we use the term "analyst" to refer to the person using PPMT.

1 Design process

In this section we describe our design process and offer a retrospective presentation of the design choices we made.

1.1 Organization of the process

Building the current version of PPMT was a two-year process that involved several iterations, during which we alternated between design and development sessions, based on the feedback we gathered. Most of the time, this feedback came from our own use of the tool or from members of our team, including the research engineer who collected the dataset we were using and was interested in its exploration. One notable exception was at the start of 2018, where we conducted a user experiment to receive more feedback on data exploration in a progressive context. This led us to refine some parts of the user interface, mainly adding feedback about the current state of the algorithm. More information about this experiment is available in appendix B. Screenshots of older iterations of PPMT are available in appendix C.

1.2 Main design choices

Since we were interested in how one can leverage the Progressive Visual Analytics paradigm to explore activity data using pattern mining, we experimented with the three following topics: the progressive exploration of activity data, the design of progressive pattern mining algorithms, and the interactions with a progressive pattern mining algorithm. The following subsections relate the design decisions we took during this process.

1.2.1 Provide a browser-based interface

While going for a native application would have given us more freedom to develop it, we decided to create a browser-based interface. The main reason was that it would make our tool easily accessible from most computers, without needing to install any particular software. In retrospect, we still stand by this decision, even though it limited us to a single thread due to limitations on shared memory that were introduced in all major browsers in early 2018, in reaction to the Meltdown and Spectre security issues[2][3][4]. Enabling multi-threading through the use of web workers would have therefore required duplicating the data, thus greatly increasing the browser memory requirements.

1.2.2 Use a client-server architecture

This second design decision ties into the first one, and the fact that we were dealing with activity datasets, that usually contain a large amount of events. Since pattern mining on large datasets can have significant requirements in terms of processing power and memory consumption, being able to delegate this task to a remote server allows the client to focus its resources on the visualization and exploration process.

1.2.3 Focus on interactions rather than visualization

In a progressive context, extensive work could have been conducted on both the *visualization* and the *interaction with the algorithm* aspects of the system. We were more interested in exploring the possible interactions between the analyst and the algorithm, which led us to focus our research on this matter. Doing so, we relied on classical visualizations (essentially timeline-based, see section 6.3), that we adapted to work in a progressive context.

[2] https://meltdownattack.com/, archived on 2020/01/03
[3] https://webkit.org/blog/8048/what-spectre-and-meltdown-mean-for-webkit/, archived on 2019/11/14
[4] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SharedArrayBuffer, archived on 2019/11/23

1.2.4 Adapt an existing pattern mining algorithm

When considering the pattern mining algorithm that would run within our system, we had the choice between creating a progressive version of an existing one and designing our own algorithm from the ground up. We decided to adapt an existing algorithm, which was faster than starting from scratch, thus allowing us to start experimenting with our data more rapidly. In addition to being faster, this solution also had the benefit to provide us with a reference algorithm that we could use as a comparison for metrics such as execution speed and memory requirements. The way in which we adapted the algorithm is presented in detail in section 6.5.

Retrospectively, we still believe this was the right choice to make in order to be able to start experimenting with a working algorithm. However, working within the logic of the non-progressive implementation made the task of adding the ability to steer the algorithm complicated at times.

1.2.5 Communicate frequent patterns when all occurrences are discovered

Even though any algorithm has to provide intermediate results to be considered progressive, the rate at which these results are provided can depend on the task at hand and the needs of the analysis. In PPMT, we made the choice to output a frequent pattern when all its occurrences have been discovered. When designing our progressive pattern mining algorithm, we used an existing implementation that outputted frequent patterns at the end of each pattern size extraction. On large datasets, it had the drawback of presenting hundreds (if not more) of patterns at once, which can be daunting for the analyst. Presenting batches of frequent patterns (*i.e.* outputting them when n frequent patterns have been discovered) could have been possible, but determining the size of the batches felt rather arbitrary.

Going down to the individual patterns, we could also present a pattern as soon as enough occurrences have been found to make it frequent, and then update its number of occurrences. However, waiting for the complete discovery of the occurrences allowed us to have less information to update, thus increasing the analyst's confidence in the displayed numbers since they know they will not change. An additional argument in favor of our choice is the fact that a single candidate can be processed rather quickly. As such, showing a number of occurrences for a fraction of a second and then updating it can be confusing. While sufficiently large datasets might justify changing our decision, the amount of data needed to do so is most probably well beyond PPMT's data management capabilities.

1.2.6 Extract every occurrence, even during a steering

When implementing steering the computation towards a time period or a specific sequence, we had to choose between two options: extracting occurrences only from the designated steer-

ing target, or within the entire dataset. While extracting within the steering target seems more intuitive, this solution leads to some questions. First, how to handle the minimum support threshold? When dealing with small steering targets (with regards to the complete dataset), keeping it "as is" reduces (or even removes) the ability to discover frequent patterns, which is not suitable. One could use a fraction of the support threshold, based on the proportion of data considered, but this only works under the assumption that the data (and thus the pattern occurrences) is uniformly distributed, which is not the case in almost any real dataset. Other questions arose with regards to the output of the mining. For patterns that are frequent, the system needs to communicate the fact that additional occurrences might exist. Similarly, patterns found to be non-frequent with regards to the steering target might end up being frequent over the entire dataset.

In PPMT, we opted for the second option. During a steering, the algorithm searches for a candidate's occurrences within the steering target. If no occurrences are found, the candidate is discarded. However, if at least one occurrence is found, the entire dataset is searched for occurrences. From the analyst's point of view, the speed at which a single candidate is processed makes looking for occurrences over the entire data only marginally longer than over a subset of the data. However, the main benefit of this choice is that it allows us to keep a constant support threshold and present complete information to the analyst.

1.2.7 Storing frequent patterns and their occurrences on the server

Choosing a client-server architecture raised the question of where frequent patterns and their occurrences should be stored. Having the complete information stored on the client side would have put a massive memory load on the analyst's computer, while only having it on the server would have required a lot of information transfer between the two parts.

To solve this matter, we consider this information as the combination of two parts, the *syntax data*, which is a collection of pattern syntax, and the *occurrence data*, which contains the details of the pattern occurrences. The first is more general, requires less memory space, and can be considered an access point to the other, while the second is more detailed and represents most of the memory requirements. For these reasons, we decided to keep the complete information on the server, while only providing the syntax data to the client. If the analyst is interested in specific patterns, selecting them will trigger a request for the associated occurrences that will be sent from the server and stored locally.

1.3 The *coconotes* dataset

During the development of PPMT, we mostly used the *coconotes* dataset. Originating from a past project of our research team, it contains the anonymized activity data of students using

COCoNotes[5], a platform that offers video-based education resources. Figure 6.1 illustrates the video player interface. The left part of the screen consists of two synchronized players, for the video as well as the supporting slides when available. On the right, the "plan" tab presents the structure of the video, each slide representing a chapter, allowing one to easily navigate its content. At any time when watching a video, one can create text annotations to add remarks, questions or useful additional information that will be linked to the video timestamp at which they are created. These annotations can then be kept private, or shared publicly with other users of the platform, and are presented in additional tabs in the right part of the screen.

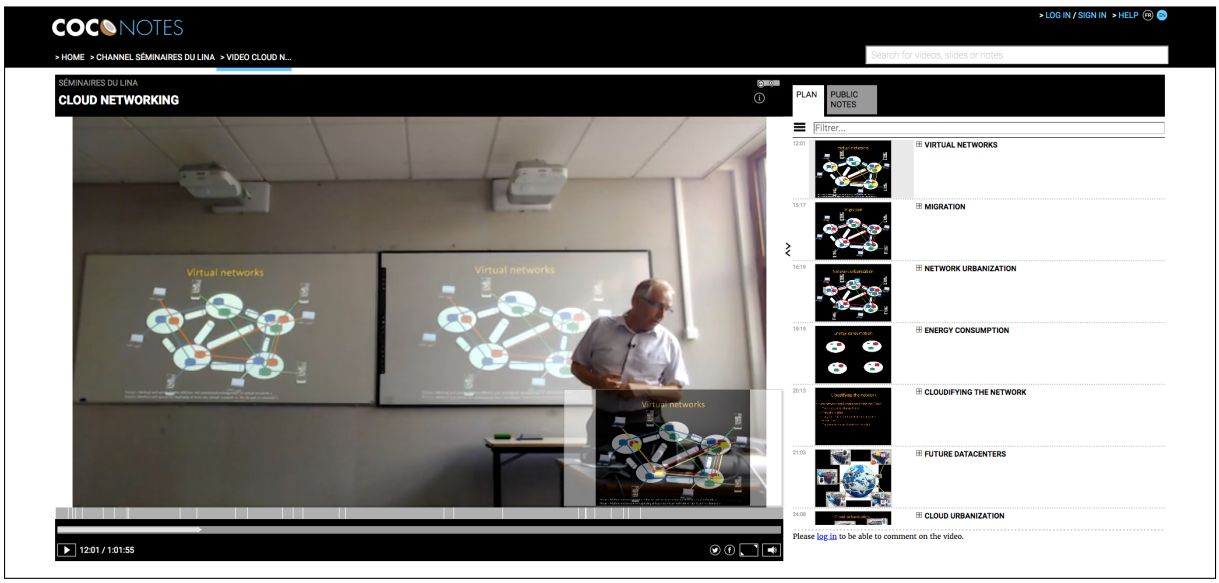


Figure 6.1: COCoNotes's video annotation interface.

The dataset itself spans a four months period from September 2016 to January 2017, representing 201.000 events, 32 event types and 211 users. Event types range from login and navigation actions to interactions with the video player (play, pause, change volume, move to a specific timestamp...), and annotation management (create, edit, share...) events. Event types are grouped into several categories, depending on which part of COCoNotes' user interface triggers them and whether they are recorded by the client (high level) or the server (low level). Figure 6.2 shows the number of events in each user trace, and figure 6.3 the distribution of the duration of the users' traces.

[5] http://coconotes.comin-ocw.org/

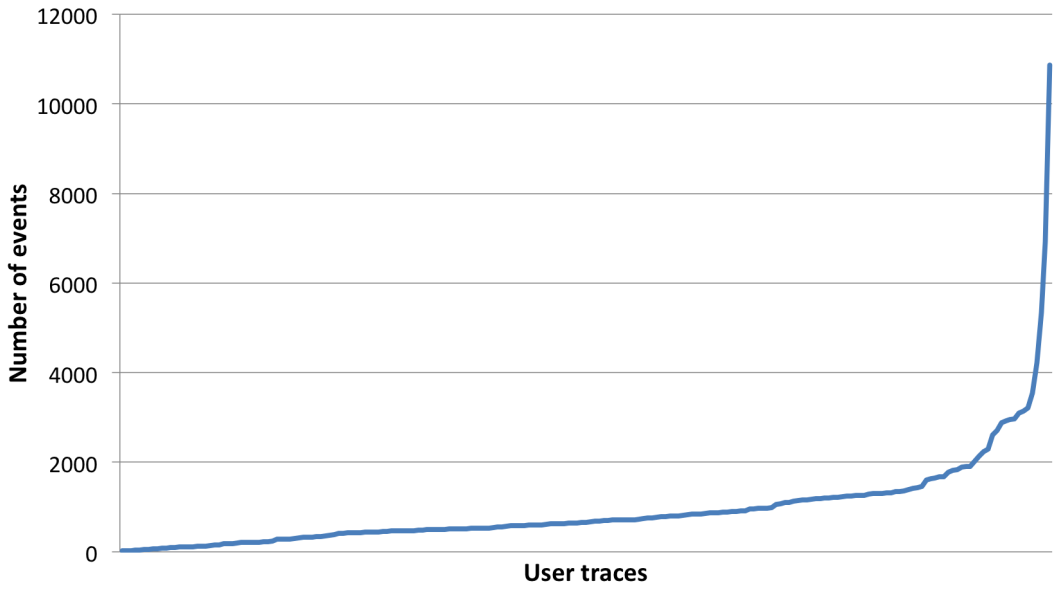


Figure 6.2: Number of events in each of the 211 user traces.

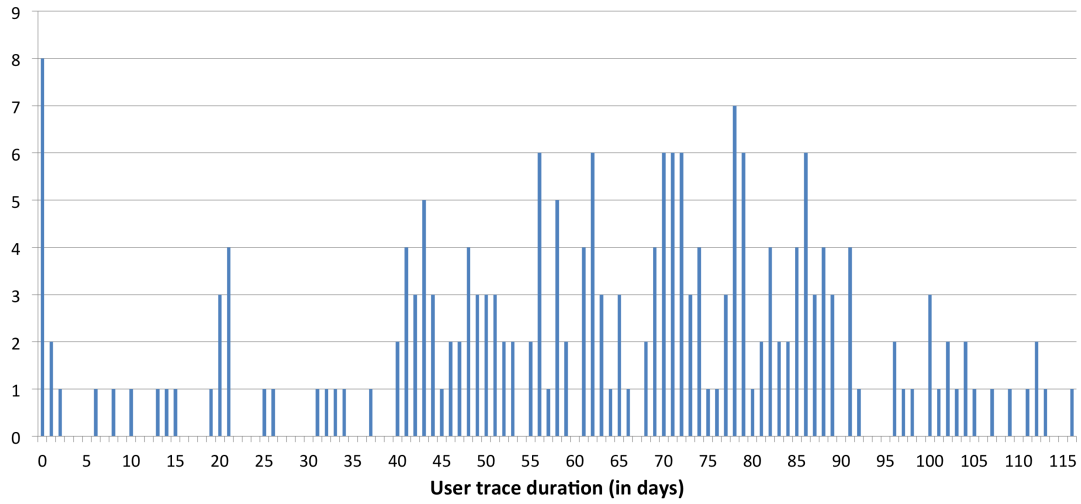


Figure 6.3: Distribution of the user traces' duration (in days).

2 Features

In this section, we introduce the features of PPMT. On a broad scale, our aim was to provide a tool to allow one to explore activity data with the assistance of a progressive pattern mining algorithm. To reach this goal, we implemented usual visual data exploration features, alongside the various components from our interaction framework introduced in section 4.1.

2.1 Technical features

Exploring raw data. Perhaps the most basic feature of PPMT is to provide access to the raw content of the dataset, independently of any algorithmic processing. The analyst has access to both aggregated information (number of events, users...) and detailed visualization of the data within timeline representations. In addition to the individual events, PPMT also represents the activity sessions within the dataset.

Extracting and exploring episodes from activity data. PPMT provides a progressive pattern mining algorithm that assists the analyst during the data analysis process. This algorithm extracts frequent episodes and their occurrences from the dataset, according to the parameters specified by the analyst. These episodes offer an abstraction of the dataset's content that the human can use while exploring data. In order to do so, PPMT allows one to visually explore the context in which occurrences are found.

Interacting with the pattern mining algorithm. Each of the four interactions we introduced in our framework presented in section 4.1 is available in PPMT. Besides restarting the algorithm with different parameter values (A4), the analyst is able to steer the algorithm in three different ways. Steering towards a time period or towards a specific user will prioritize patterns that occur within these specific subsets of the dataset (A2), while steering towards an already discovered pattern will prioritize the mining of patterns that have it as a prefix (A3). When using PPMT, an analyst can apply modifications to the dataset to integrate previous findings into the data (A1). Doing so restarts the algorithm on the new data, and any modification can be undone if needed. The current version of PPMT supports three kinds of data modification: removing specific event types, removing specific users, and creating a new event type from a pattern. In this last case, events that are involved in the occurrences of the pattern are replaced by events of the newly created type.

2.2 Supported analysis tasks

In subsection 5.1.5, we proposed a model for the tasks an analyst can perform when working with pattern mining techniques. In its current version, PPMT's support of these tasks is focused on the elementary ones, as these are the most basic steps that can be performed when exploring a dataset, on which synoptic tasks rely, making their implementation mandatory.

Direct lookup tasks can be performed through the lists of event types, users and patterns, as well as the focus view, by selecting the appropriate values for each referrer and exploring the resulting visualizations.

Inverse lookup tasks are mostly supported, depending on the missing referrers. If the

sequence or *time* is unspecified, inverse lookups can be performed the same way than direct ones, by selecting patterns to display their occurrences within the visualizations. However, if the *pattern* referrer is unspecified, there is no efficient way to obtain the task's target, since selecting some users and a time-period will not provide the patterns one is interested in. A simple way to remedy this would be to change the behavior of the pattern list, so that it is automatically filtered to only display patterns that have occurrences in the part of the dataset that is shown in the visualizations.

Direct comparison tasks are mostly achieved through visual comparison, using the data visualizations. However, should one be interested in comparing according to the information displayed in the various lists (such as the support of the patterns), it is possible to leverage the filtering and sorting options to do so more efficiently.

Inverse comparison tasks, as they rely on inverse lookups, present the same feasibility. If the required inverse lookup involves an unspecified *sequence* or *time* referrer, one can perform it and visually compare the visualizations. However, if the *pattern* referrer is unspecified, the task's target cannot be easily obtained.

Relation seeking tasks are for the most part not supported by PPMT, since no way of specifying a relation (either between characteristics or referrers) are provided. However, should the relation be one that can be materialized through PPMT's user interface, then one can look for its occurrences within the visualizations. An example of such case could be looking for sequences that start before a specific sequence, achievable by sorting the sequence list by start time.

While the above paragraphs address how one can perform each type of task on the available data, the ability to steer the computation provides an efficient mean of making sure that said available data is as comprehensive as possible. This is especially true for *direct* tasks (lookup or comparison), since each referrer can be expressed via a dedicated steering target: sequences (steering on users), patterns (steering on a prefix) and time (steering on a time period).

With regards to the **synoptic** tasks, PPMT does not provide specific assistance. To do so would require a way for the analyst to be able to either specify structures (and then have PPMT output the corresponding behaviors) or select some data or pattern occurrences that would be used to construct a behavior (and then have PPMT identify underlying structures). However, one can still leverage the different visualizations to perform theses tasks "by hand".

3 User interface

In this section we present the user interface of PPMT, composed of several linked widgets that are organized in three columns (figure 6.4). The left column contains information about the

Figure 6.4: PPMT's interface. *The left column is about the dataset. The right column is about the pattern mining and its output. These elements can be visualized in the graphical views in the middle column.*

selected dataset (**A**) and the actions that have been performed during its exploration (**B**). The column on the right contains information about the progressive pattern mining algorithm, its current state (**G**) and the frequent patterns that have been discovered (**H**). Finally, the middle column contains timeline-based graphical representations of the data and selected patterns (**C**, **D**, **E**, **F**). The following subsections will focus on the different widgets that compose this interface, introducing them in more details. Illustrations throughout this section showcase data from the *coconotes* dataset, described in subsection 6.1.3.

3.1 Dataset-related panels

3.1.1 Trace information

This widget is the first tab in the left column, and is illustrated in figure 6.5. In its upper part, it provides aggregate information about the dataset that is currently being explored. It displays the total numbers of events, event types, users and sessions, along with the timestamps of the first and last events. In most cases, this information will not be frequently needed by the analyst during their data exploration. It can however provide some context in the early stages of the work, or when investigating some hypothesis.

The lower part of the tab contains a history of the events related to the analyst's actions

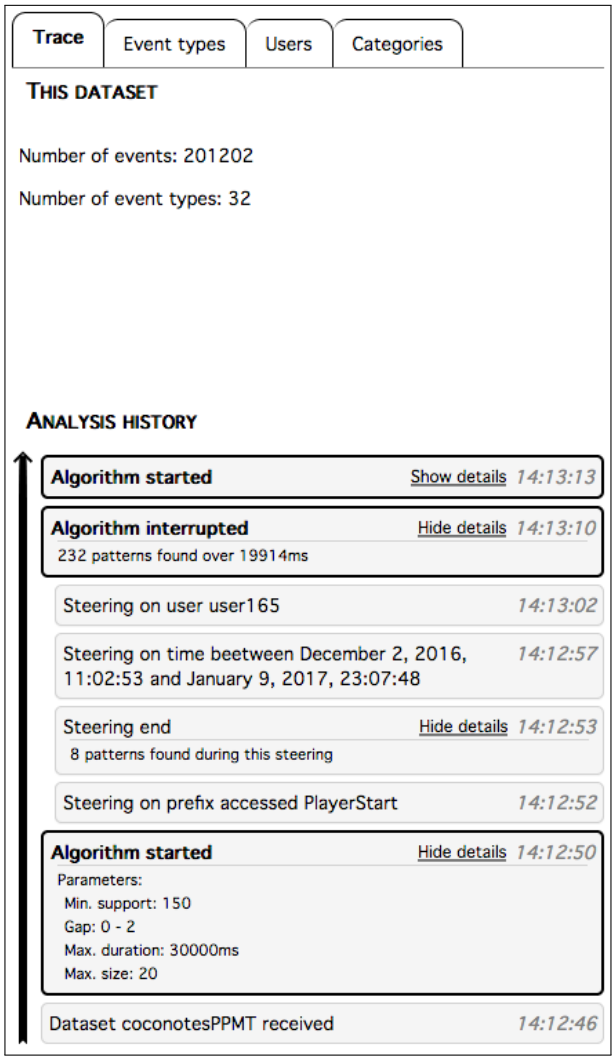


Figure 6.5: Information about the dataset and the analysis history.

since she started exploring the current dataset. Each event is a card with a title, a timestamp and a description that will depend on the kind of event. The history always starts with the `Dataset [X] received` event indicating that the client has loaded the complete dataset. From this, the history will log the following events:

- `Algorithm started`, with the description containing the value of each parameter.
- Algorithm ended, with the description containing the number of patterns found and the time it took. Two different events can be encountered:
  - `Algorithm completed`, if the algorithm ran to completion;
  - `Algorithm interrupted`, if it was interrupted by a restart.
- `Steering on [...]`, indicating the start of a steering, with the title indicating the type of

steering and its target.

- `Steering end`, with the description indicating the number of patterns discovered during the steering.
- Data modification, with a different event for each type of modification:
  - `Event type created: [...]`, with the title indicating the name of the new event and the description containing the former pattern that served to create it.
  - `[X] users removed`, with the title indicating the number of users removed and the description containing the list of these user IDs.
  - `[X] event types removed`, with the title indicating the number of event types removed and the description containing the list of these event types.
- `Reset the dataset`, when data modifications are reverted to go back to the initial dataset.

To allow an easier understanding of the exploration history, algorithm start and end are bolded (both the title text and the card border), and events happening during the algorithm execution are indented. The analysis is also able to show or hide each event description, which provides a more condensed history when this information is not needed. If the dataset has been modified, a button is available at the top of the history to revert these changes.

3.1.2 Event type information

This second tab in the left column is illustrated in figure 6.6 and contains a list of all the event types that exist in the dataset. Each event type has a name, the symbol used to represent it, a support (the number of times it is encountered in the data), a category (which defines the symbol's color) and the number of users that have events of this type in their data. When available, a description is also provided for each event type, that the analyst can hide to provide a more compact list. The list can be sorted according to each column by clicking on their headers, with decreasing support value being the default ordering.

The analyst can click on a row to toggle on or off the highlight of the corresponding event type in the central visualizations. Highlighted rows are bolded. While hovering over a row, its background changes to highlight it and a contextual button appears over it on the right, giving the analyst access to three options: removing all occurrences of this event type from the dataset, removing all highlighted event types, and removing all non-highlighted event types. Clicking the button performs the first option. Using any of the options will display a modal window asking for the analyst's confirmation, while warning that this will lead to the restart of the algorithm (figure 6.7).

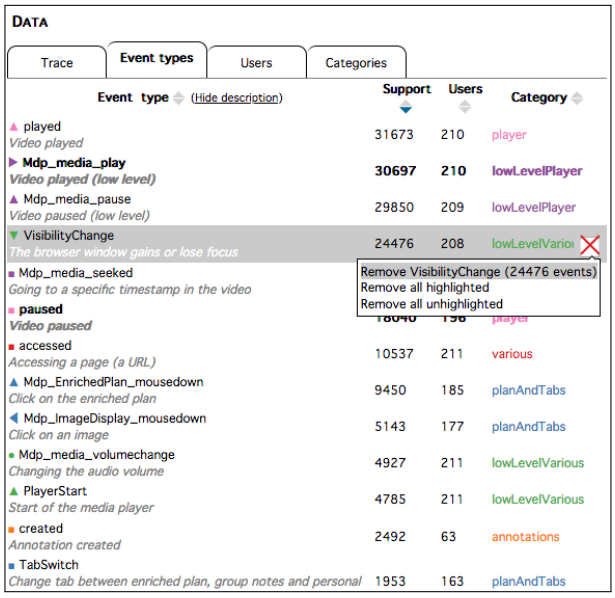


Figure 6.6: The list of event types.

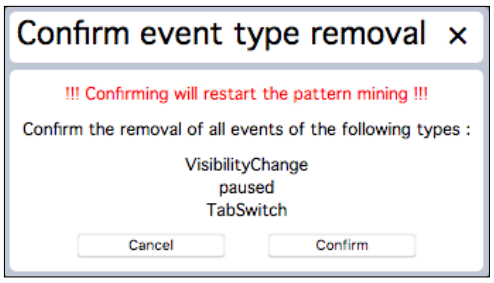


Figure 6.7: The modal confirmation window before removing several event types from the data.

3.1.3 Users information

This third tab in the left column is illustrated in figure 6.8 and contains a list of all the users that exist in the dataset. Each user has an ID, is associated to a number of events, and to the date of the first and last of these events, the duration between these two dates and the number of sessions during this time-period. Contrary to the other information, the sessions are not directly present in the data but derived from it. For a given user, we define a session as *a series of consecutive events of any size, related to the same user, in which the duration that separates two consecutive events is no longer than an analyst-defined value*. Within PPMT, this value is set at 30 minutes when exploring the COCoNotes dataset, based on existing work on reading sessions in online courses (Sadallah, Encelle, Maredj, & Prié, 2015). The list can be sorted according to each column by clicking on their headers, with decreasing number of events being the default ordering.

Similarly to the event types list, the analyst can click on a row to toggle on or off the highlight

of the corresponding user in the central visualizations. Highlighted rows are bolded. While hovering over a row, its background changes to highlight it and two contextual buttons appears over it on the right. The first allows the analyst to start steering the algorithm on this user, while the second button provides access to three options: removing all events of this user, removing all highlighted users, and removing all non-highlighted users. Clicking the button performs the first option. Using any of the options will display a modal window asking for the analyst's confirmation, while highlighting the fact that this will lead to the restart of the algorithm (figure 6.9).

At the top of this widget is a text field that can be used to filter the list. Highlighted users are always displayed regardless of the current filter, but non-highlighted users are only displayed if their ID contains the current filter value. While the analyst is typing into the field, a few auto-completion suggestions are provided, that can either be clicked or navigated using the keyboard.

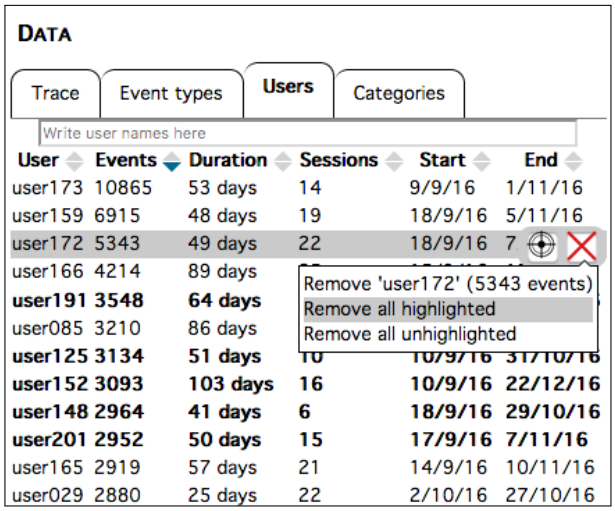


Figure 6.8: The user list. *Here, users* `user191`*,* `user125`*,* `user152`*,* `user148` *and* `user201` *have been selected.*

Figure 6.9: The modal confirmation window before removing several users from the data.

3.1.4 Categories information

This fourth tab in the left column is illustrated in figure 6.10. It contains a list of all the categories of event types present in the dataset, along with the color associated to them. Each color has two variations, the second one being a faded version of the first. When no particular highlight is being used, the first variation is used in the graphical representations. When a user and/or an event type highlight is occurring, highlighted events use the first variation while non-highlighted ones use the second.

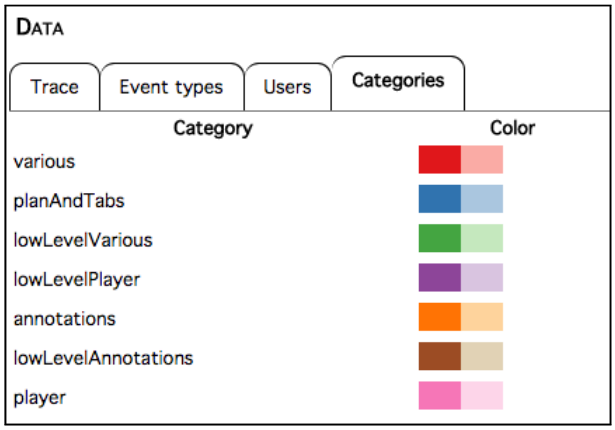


Figure 6.10: The list of event type categories.

3.2 Algorithm and pattern-oriented panels

3.2.1 Default algorithm view

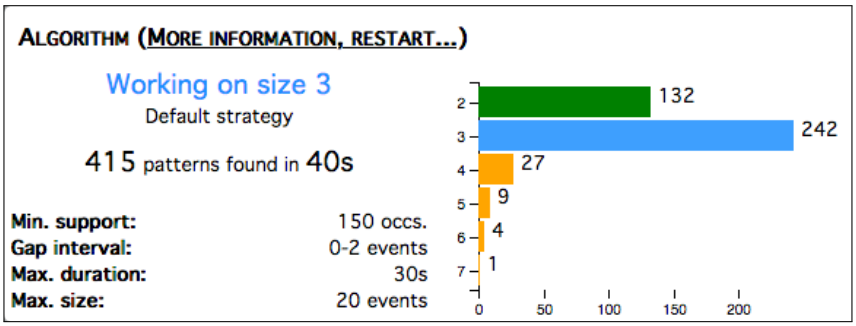


Figure 6.11: The default algorithm view.

Located at the top of the right column, this widget provides information about the pattern mining algorithm, and is illustrated in figure 6.11. In its left half, it displays its current status, the number of patterns discovered, the execution time and the value for the various parameters of the algorithm. The status display is itself made of two information, the current activity and the

current strategy. Here, *activity* refers to what pattern size is being investigated, while *strategy* can either be "default strategy" or an indication of a steering target ("steering on [...]"). The right half contains a bar chart that presents the number of frequent patterns for each pattern size. The bar chart uses the following color code:

- A green bar indicates that all the patterns of the corresponding size have been extracted.
- An orange bar indicates that the corresponding size has been investigated during a steering. It will be colored orange until its completion under the default strategy.
- A blue bar indicates that the corresponding size is the current target of the algorithm, whether a steering is occurring or not.

The same color code applies to the text that indicates the algorithm's status, which can take the following values:

- `completed`.
- `Steering on` X, with X being a description of a steering target (either a syntax prefix, a time period bounds or a user id).
- `Working on size` n, with n being the currently investigated pattern size.

While this provides a condensed view of the current algorithm state, the analyst can click anywhere in this area to open the extended algorithm view described in the next paragraph.

3.2.2 Extended algorithm view

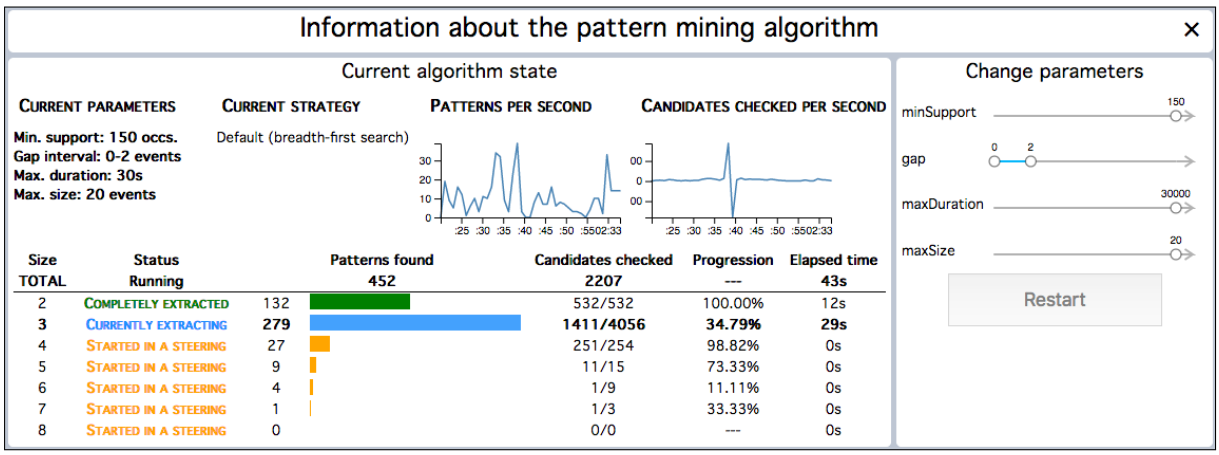


Figure 6.12: The extended algorithm view.

Since it is not always available, this widget is displayed in a modal panel, on top on the rest of PPMT's interface. As shown in figure 6.12, all the information found in the default algorithm view is also presented here. Two graphs are giving indications about the algorithm's speed during the last minute, in the form of respectively the number of frequent patterns found per second and the number of candidates checked per second. The bar chart is integrated into a table in order to provide additional information. For each pattern size, rather than just showing the number of frequent patterns, the table includes the number of candidates that need to be checked and that have already been checked, an indication of progression (with regards to the number of candidates) and the time the algorithm has spent investigating it. The color code described in the previous paragraph is also used in this extended view. On the right side of the widget, the analyst can restart the algorithm with new values for the parameters, which can be set using four sliders. Clicking anywhere outside the modal panel will close it, allowing the analyst to go back to the main interface.

3.2.3 Pattern list

Located in the bottom part of the right column, this panel is illustrated in figure 6.13. It consists of two parts, the list of frequent patterns itself, and the filters that can be applied to it.

The list. The list contains all the frequent patterns with size > 2 that have been discovered and that pass every filter the analyst have specified. Each row presents a pattern with its name (the event types that compose its syntax, along with the corresponding list of symbols), its support (number of occurrences), the number of users that include the pattern in their data, and its size. The list can be sorted according to each column by clicking on their headers, with increasing size being the default ordering. This analyst is also able to hide the patterns' name, showing only the corresponding sequence of symbols. This provides a more compact list, useful when considering long patterns or patterns containing event types with a long name.

The analyst can click on a row to toggle on or off the highlight of the corresponding pattern in the central visualizations. The text in highlighted rows is bolded. While hovering over a row, its background changes to highlight it and three contextual buttons appear over it on the right (see the 4th pattern from the top in figure 6.13). The first one highlights all the users whose data present the pattern, or de-highlights them if they are all already highlighted. The second button starts a steering whose target will be patterns that have the selected pattern as a prefix. Finally, the third button is used to modify the dataset by creating a new event type from the occurrences of the pattern. When the analyst clicks on the button, a modal panel appears, allowing the analyst to select the name and description of the new event type (figure 6.14). If they so choose, the analyst can also remove all occurrences of the events types that compose the selected pattern, rather than just the ones involved in its occurrences.

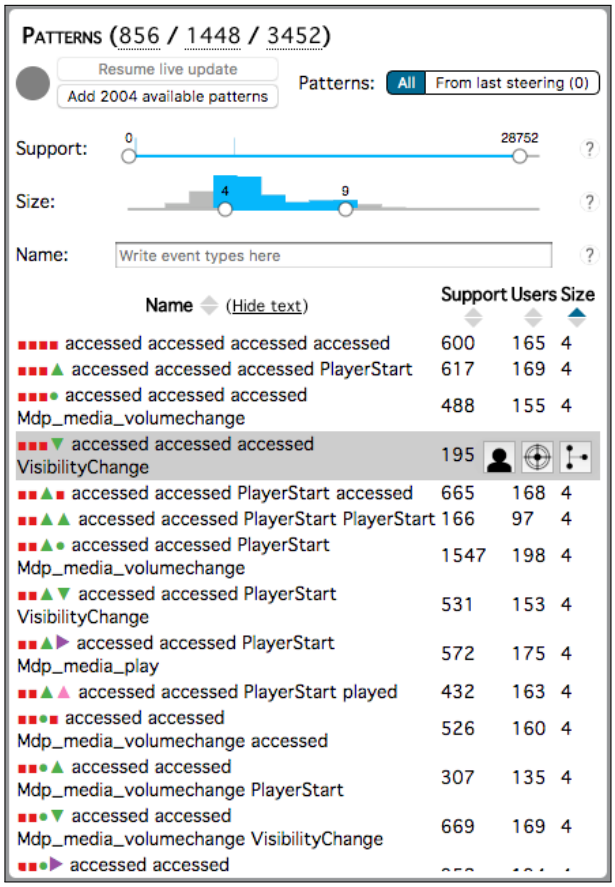


Figure 6.13: The list of frequent patterns and the associated filters. *Here, the 4th row is being hovered over.*

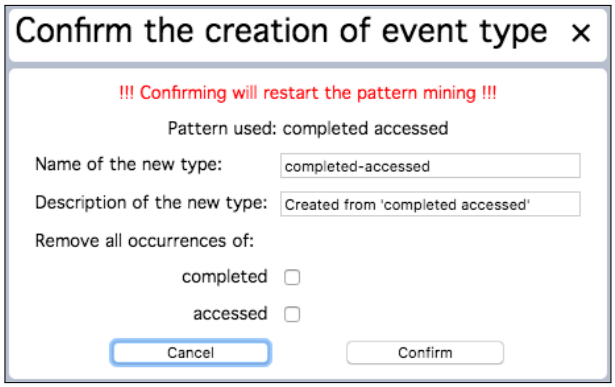


Figure 6.14: The event type creation modal panel.

The filters. Several filtering options are available above the pattern list, allowing the analyst to more easily access the patterns that are of interest to her. Two sliders are provided to filter based on the support and size of the patterns. Both display a bar chart of the current pattern

frequent patterns based on their respective attribute, that is colored to highlight which patterns are actually used to build the list. Each slider allows selecting both a minimum and a maximum value, and their range is updated if needed when new patterns are discovered. Under these sliders, a text field allows the analyst to filter the list according to the syntax of the patterns, to only include the ones that include the entered text. Auto completion is offered and suggests event type names.

While the previously described filters are to be expected from any tool, the top part of the widget offers other filtering options directly tied to the progressive nature of the algorithm. On the right, the analyst can switch between considering all patterns or just the ones discovered during the last steering (or during the current one if on is happening). Hovering over the switch displays a tooltip indicating the target of said steering. On the left, a button allows to stop and resume the live update of the pattern list, which will determine whether newly discovered patterns will be instantly added to the list or not. When the live update is stopped, a button appears, indicating how many patterns are currently waiting to be added. Clicking on this button adds them to the list (provided they comply with the other filters) without resuming the live update. If the analyst goes back to a live update of the list, any “awaiting” pattern is added. A circle serves as an additional indication of the live update status, being grey when it is paused and a pulsing green when active.

At the very top of the panel, three numbers provide an indication about the amount of patterns the analyst is dealing with. From left to right, these are respectively the number of patterns that pass the current combination of filters (*i.e.* number of rows in the list), the number of patterns taken into account to create the pattern list and the number of discovered patterns. Hovering over them provides this information to the analyst. In practice, the last two numbers are identical if the live update is on. If it is paused, the difference between them is the number of patterns waiting to be included in the list.

3.2.4 Selected pattern list

Located between the algorithm state and the pattern list, this panel is presented in figure 6.15. It provides the same features than the frequent pattern list (without the filter part), but can only contain up to five patterns that have been selected by the analyst. If it contains at least one, a “Reset” link is available at the top of the panel to clear the pattern selection.

Similarly to the complete pattern list, hovering over a row changes its background changes and three contextual buttons appears over it on the right (see the 2nd pattern from the bottom in figure 6.15). The first one highlights all the users whose data present the pattern, or de-highlights them if they are all already highlighted. The second button starts a steering whose target will be patterns that have the selected pattern as a prefix. Finally, the third button is used to modify the dataset by creating a new event type from the occurrences of the pattern. When

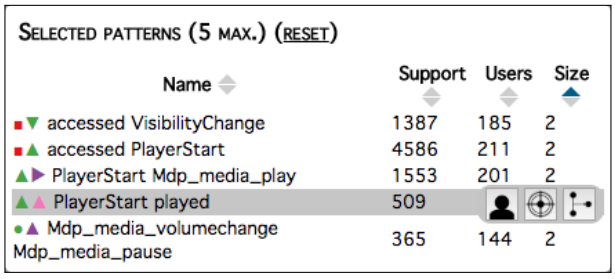


Figure 6.15: The list of selected patterns.

the analyst clicks on the button, a modal panel appears, allowing the analyst to select the name and description of the new event type (figure 6.14). If they so choose, the analyst can also remove all occurrences of the events types that compose the selected pattern, rather than just the ones involved in its occurrences.

3.3 Visualization-oriented panels

The three panels that compose the central part of PPMT's interface offer synchronized timelines that allow the analyst to explore various aspects of the data. While the following paragraphs will focus on each of these timelines individually, it is important to keep in mind that an important part of their usefulness during the analysis comes from their synchronicity. It is implemented through what we call the "focus", a subpart of the time-period covered by the dataset, which can be set in the "overview" visualization or by panning and zooming in the timelines. Both the "all users" and "per user" timelines only cover the time-period defined by the focus.

3.3.1 Summary of highlights and selections

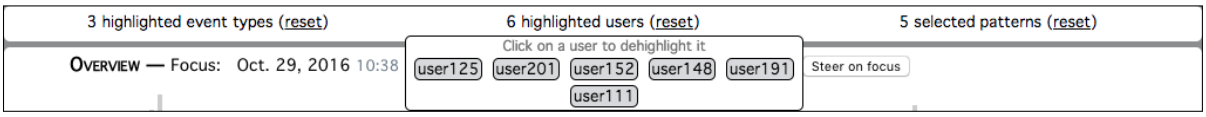


Figure 6.16: Selections and highlights summary.

The top part of the central column displays the number of highlighted event types and users, along with the number of selected patterns, as shown in figure 6.16. Links are also available to reset these highlights and selections, rather than doing it element by element. When hovering over these numbers, a tooltip displays the detailed list of elements. The analyst can click on these to de-highlight or unselect specific items from this display.

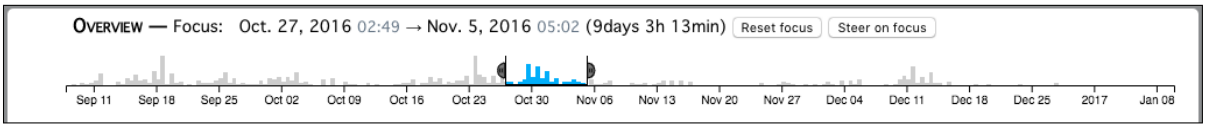


Figure 6.17: The dataset overview.

3.3.2 Overview

Located at the top of the central column, this widget provides an overview of the dataset. As illustrated in figure 6.17, it provides a timeline that spans the entire dataset. On top of it is a selector that the analyst can manipulate (either moving it or resizing it using two handles) to designate the *focus*, *i.e.* the portion of the timeline that will be displayed in the other two visualizations. By default, the focus is on the entire data. The timeline itself displays a bar chart of the number of event types, whose granularity will be automatically adjusted based on the time span of the focus. Above the timeline, the bounds of the focus are displayed alongside the corresponding time-span. Two buttons are also available, one to reset the focus on the whole dataset, the other to start steering the algorithm on the current focus.

3.3.3 All users visualization

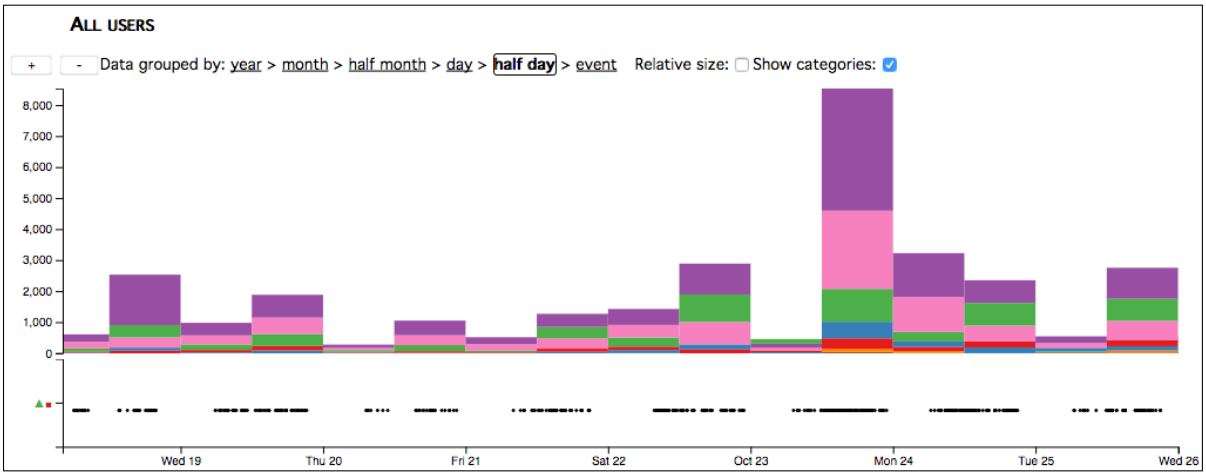


Figure 6.18: Visualizing aggregated events from the whole dataset.

Located in the central part of the middle column, this panel is illustrated in figures 6.18 and 6.19. From top to bottom, it contains the following three parts: a set of controls, a timeline displaying events and a timeline displaying the occurrences of the selected patterns. The analyst can adjust the focus by panning and zooming over the timelines, by using the "+" and "-" control buttons or by clicking the links to the different granularity levels.

Depending on the time period covered by the focus, the events-displaying timeline switches

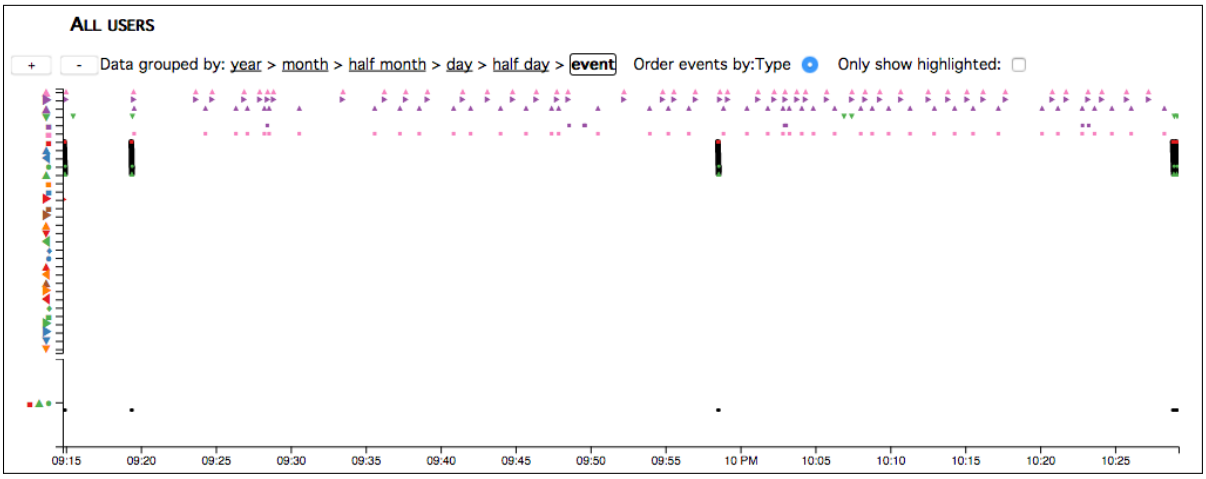


Figure 6.19: Visualizing individual events from the whole dataset.

between two modes: an aggregated representation (figure 6.18) and a per-event representation (figure 6.19). Depending on the dataset, up to five granularity levels are available for the aggregated representation: per year, per month, per half month, per day and per half day. In this mode, the representation is a bar chart that reflects the number of events in each aggregate. An option is available in the control area to turn the bars into stacked bars of the different event type categories. If this option is activated, another option appears that allows the analyst to switch the Y-axis between an absolute and a relative scale. In the per-event mode, the Y-axis is dedicated to the event types, following the ordering of the list in the corresponding panel, and the events are represented in the timeline by their symbol. When patterns are selected, black lines connect the events that compose their occurrences. If at least one event type is highlighted, an option is available in the control area to only display the events of a highlighted type. This allows the analyst to more easily focus on events of interest when needed.

Under the main timeline, a smaller timeline is dedicated to display the occurrences of the selected patterns. The Y-axis indicates the patterns, with each occurrence being represented with a horizontal black line going from its start to its end. Contrary to the ones in the main timeline, these black lines are visible even in aggregated mode to help spot the occurrences within the data. As such, this visualization can be considered an overview of the selected patterns' occurrences.

3.3.4 Per user visualization

Located at the bottom of the middle column, this timeline presents information about the sessions of each user, as illustrated in figure 6.20. The Y-axis consist of a list of users, following the user list panel's ordering, and the sessions are displayed as rectangles that span the corresponding time period. When patterns are selected by the analyst, the sessions in which their

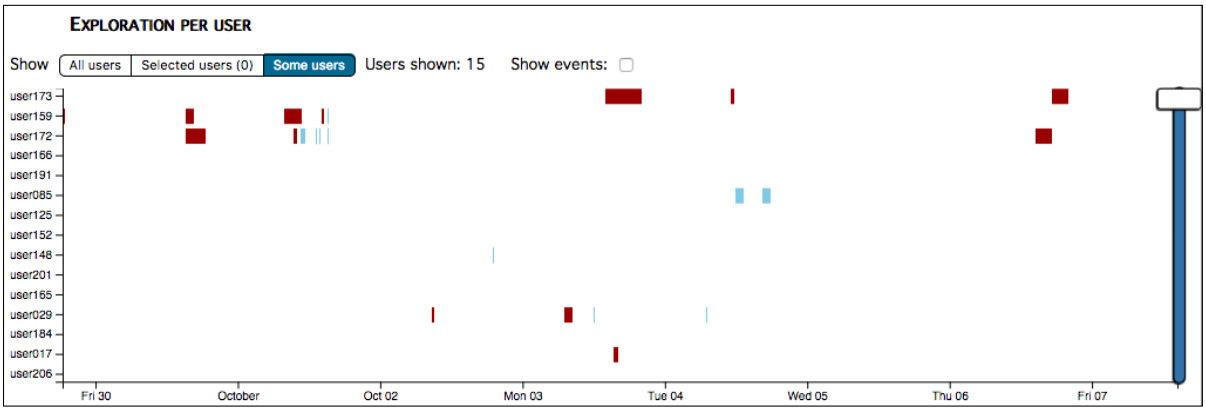


Figure 6.20: Visualizing user sessions.

occurrences are found are colored in red to distinguish them from the others, colored in blue. Highlighted users are also made easier to spot by having the corresponding rows having a light grey background (figures 6.21 and 6.23). To explore the content of a particular session, the analyst can click on the associated rectangle to align the focus on the corresponding time period.

Depending on what the analyst is interested in, a toggle located right above the timeline offers three different modes. “All users” (figure 6.21) displays every user, which will offer an overview of the sessions repartition at the cost of any finer detail in any dataset having at least 50 users. “Selected users” (figure 6.22) is similar to the previous option, but restricted to the currently highlighted users. In this mode, the grey row background is removed since it would be present for every row. Finally, “Some users” (figure 6.23) displays only fifteen users at a time, with a slider on the right allowing the analyst to browse the entire list. In this mode, the analyst is able to display the individual events (either all of them or only the highlighted ones) over the sessions to explore their content.

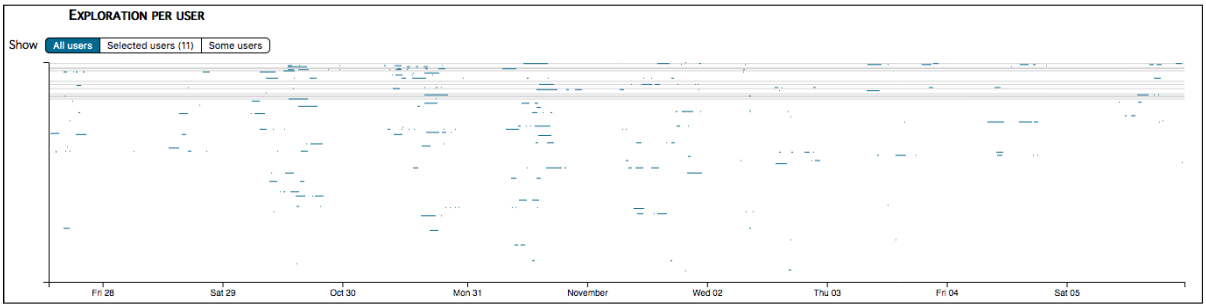


Figure 6.21: “All users” view.

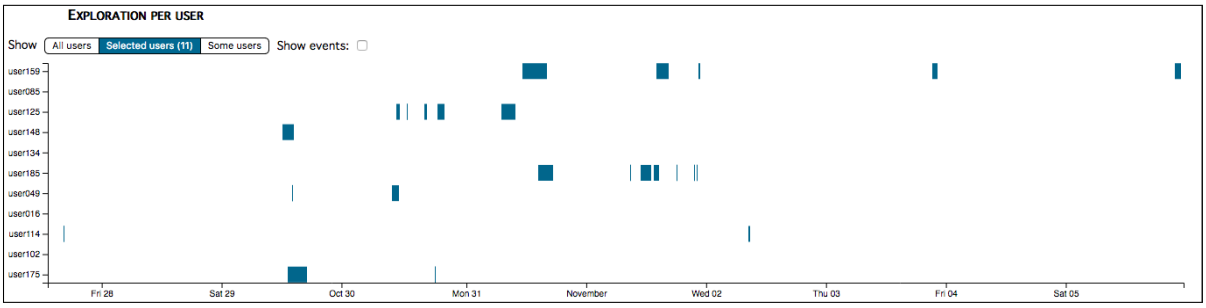


Figure 6.22: "Selected users" view.

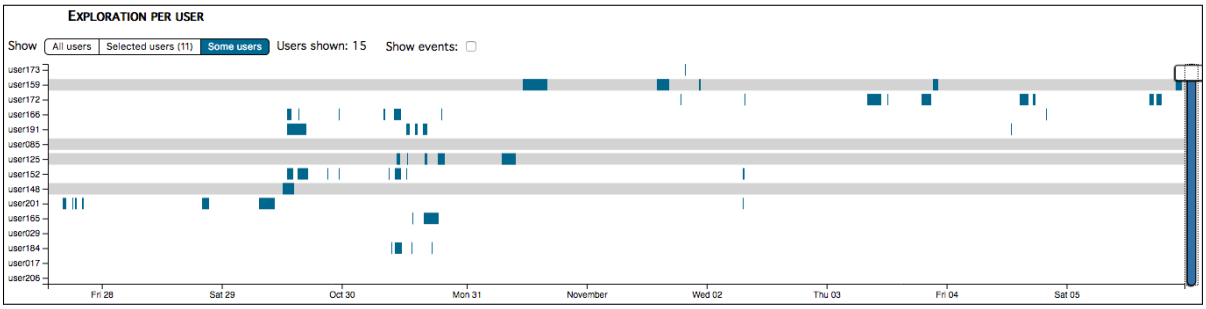
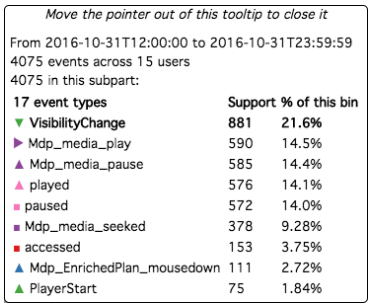


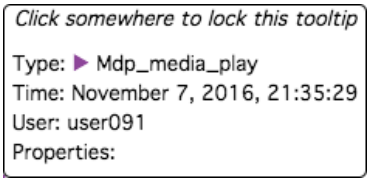
Figure 6.23: "Some users" view.

3.3.5 Tooltips

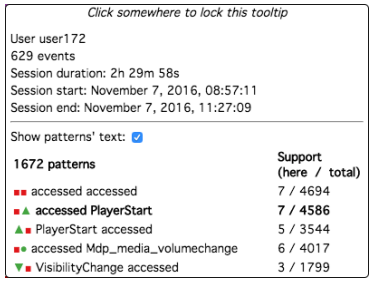
When hovering over certain elements of the "all users" and "per user" visualizations, the analyst has access to a tooltip that provides additional information about various elements. These tooltips are illustrated in figure 6.24.



(a) Tooltip for an event bin



(b) Tooltip for an event



(c) Tooltip for a user session

Figure 6.24: Tooltips available in PPMT.

By default, the tooltip moves with the mouse cursor, however the analyst can click to prevent the tooltip from moving, making it possible to interact with its content. In the "all users" visualization, tooltips contain information about a specific bin in aggregate mode or about a single event in event mode. In the "per user" visualization, tooltips present information about

the session and the patterns it contains.

3.4 Dataset selection

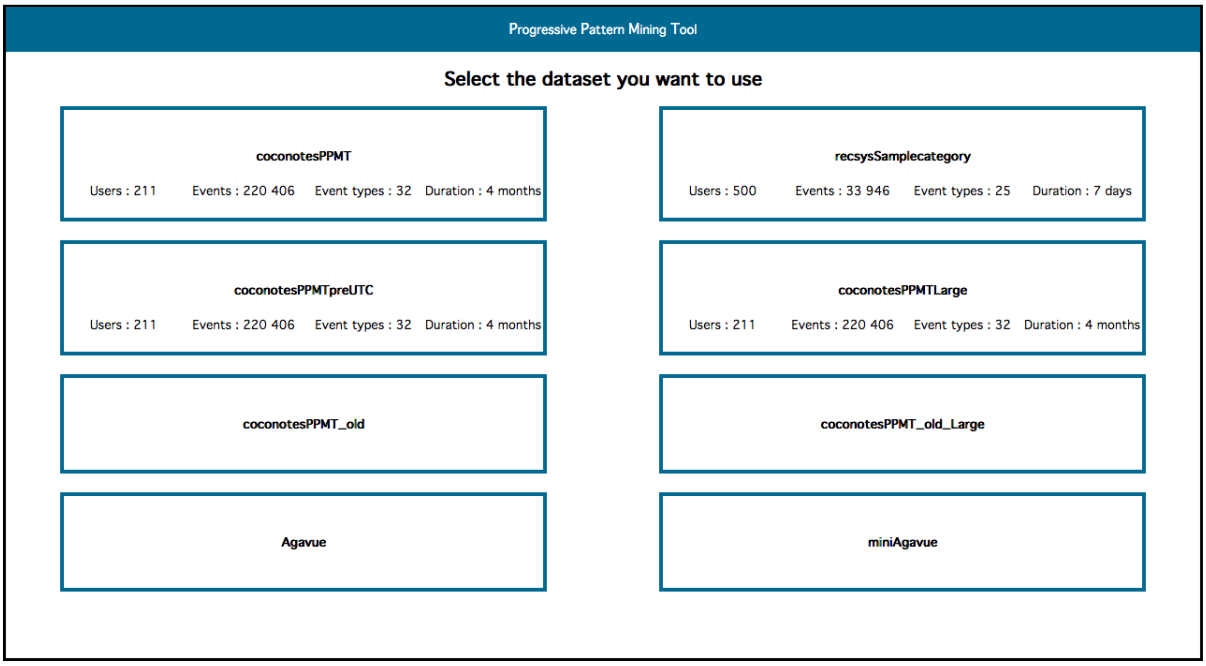


Figure 6.25: The dataset selection page.

Contrary to all the previous elements that are part of the same page, the dataset selection illustrated in figure 6.25 is a different page. While not needed for the data analysis work performed with PPMT, it is the first page an analyst sees when starting to use PPMT in its current implementation. For every dataset on the server, it displays a card that present at least the dataset name. When available, additional information can include the number of users, events and event types of the dataset, along with the duration between its first and last events.

4 Architecture

In this section, we present the architecture on which PPMT is built. We first introduce the logic within PPMT, before focusing on the actual implementation.

4.1 Logical architecture

Our first work on the architecture of a progressive pattern mining system designed to explore activity data led us to the design presented in figure 6.26. In this workflow, interactions traces are processed by the progressive pattern mining module to extract patterns, that are sent to

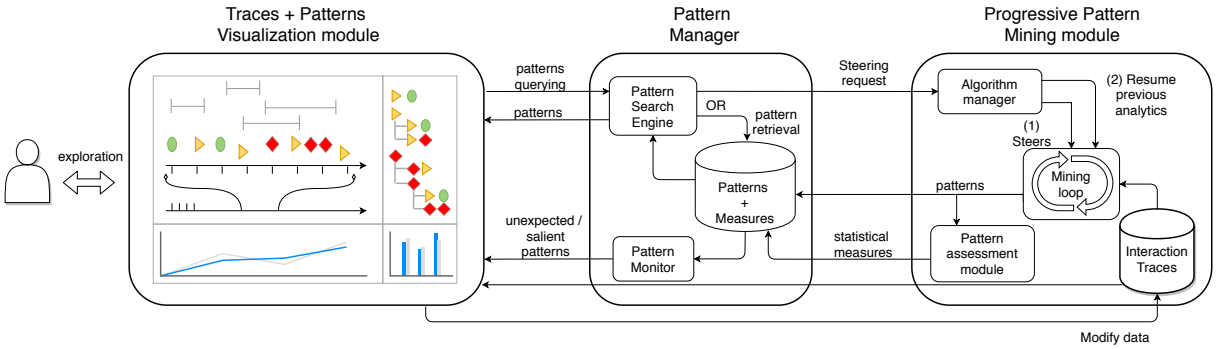


Figure 6.26: Our logical architecture for a progressive pattern mining system.

a pattern manager for storage and to a pattern assessment module to derive statistical measures, also stored in the pattern manager. While the algorithm keeps running and outputting patterns, the analyst explores visual representations of the data, patterns and statistical measures. When necessary, the analyst queries additional patterns from the pattern manager. Such requests are handled by a pattern search engine, that will produce an answer either by retrieving already discovered patterns relevant to the query or by steering the pattern mining algorithm to prioritize the analyst's request. Independently from this queries, a part of the pattern manager we call the pattern monitor processes the discovered patterns to push additional patterns to the visualization. The analyst can also request modifications of the data. This process is supposed to assist the data exploration by identifying unexpected or salient patterns, based on their statistical characteristics and the analyst's analysis behavior. This model is an improvement over our earlier work on a theoretical architecture for a progressive pattern mining system (Raveneau, Blanchard, & Prié, 2016), in which the behavior within the Progressive Pattern Mining module was less detailed. This improvement comes from our practical experience implementing PPMT.

4.2 Implemented architecture

4.2.1 Progressive Visual Analytics architecture

PPMT is built on a client-server architecture, and the implementation for the most part follows our logical architecture, as illustrated in figure 6.27. Due to our focus on interactions between the analyst and the algorithm, two elements of the logical model are missing from PPMT: the *pattern assessment module* and the *pattern monitor*. This allowed us to focus more heavily on the various ways one could steer the algorithm. As explained in our design choices (see subsection 6.1.2), the visualization and the pattern mining are clearly separated by using a web-based client in charge of the visualization, while the server performs the progressive pat-

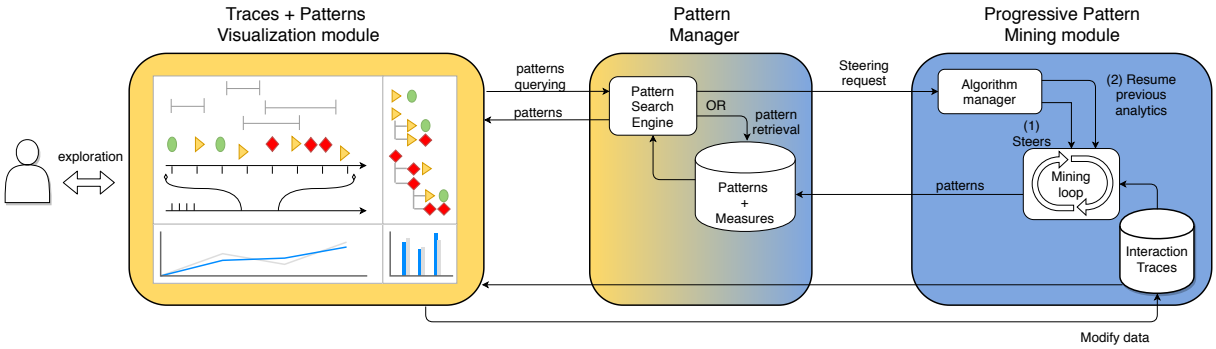


Figure 6.27: PPMT's logical architecture. *Yellow sections are part of the client; blue ones are part of the server.*

tern mining and provides the data, that is stored on both side. While the Visualization and Progressive Pattern Mining modules are faithful to our theoretical logical model, the pattern manager, on the other hand, is not as clearly defined. The list of frequent patterns is stored on both sides, with the server storing the complete occurrence data, while the client only knows the occurrences of the patterns requested by the analyst.

4.2.2 Server architecture

While the previous sub subsection details the architecture that a single Progressive Visual Analytics process relies upon (*i.e.* having a single client communicate with the server), it ignores some components of PPMT's server-side that allow it to concurrently handle several clients. The first is the global *dataset manager*, which manages the copy of the data available to every client. To reduce the memory needs, only one copy of each available dataset exists, that can be read by several clients. It is only when one of them wants to modify the data that a copy of the dataset is created, on which the requested modifications are performed. The second component is the *client manager*, whose role is to manage the various instances of the previously described Progressive Visual Analytics architecture, one for each client. An illustration of these two components' role is available in figure 6.28.

4.2.3 Communicating with the algorithm

With the exception of the initial data transfer to the client that uses an HTTP request, every communication between the client and the server is done using websockets, allowing both sides to initiate a communication when needed.

When the server receives a message, it dispatches it to one of two modules, depending on the content. Requests to modify the data (or to undo a modification) are sent to the *dataset manager* module, in order to create (or update) a dedicated copy of the dataset to perform

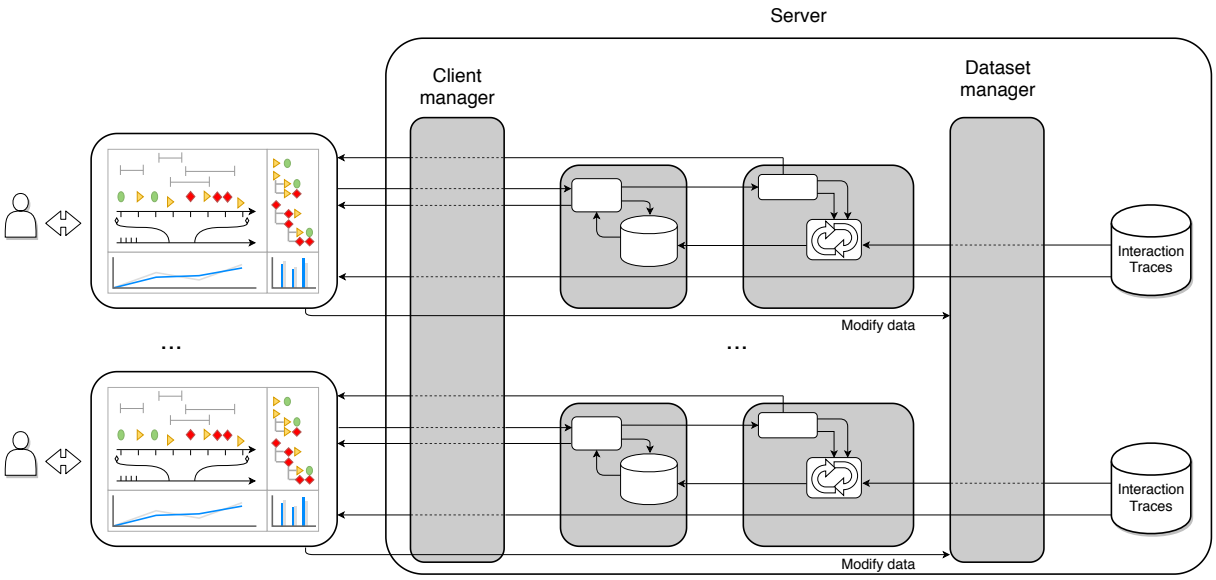


Figure 6.28: PPMT's server architecture.

the requested modifications. Other requests will consist of communication between the analyst and the algorithm, and are sent via the *client manager* to the relevant *algorithm manager* that registers them (more on how steering is handled in subsection 6.5.2). Regularly, the algorithm will also send information about its progress (number of candidates generated and verified) to the algorithm manager that will then forward it to the client.

Once a valid dataset has been selected, the server starts sending the data to the client. When this transfer is complete, the pattern mining algorithm starts running with its default parameters, and any future communication from the client to the server will either be requests to interact with the algorithm or requests for the detailed occurrences of specific patterns. In the other direction, messages from the server side will contain newly discovered frequent patterns, information about the algorithm's state and progression, and answers to the client requests.

5 Progressive pattern mining in PPMT

In this section, we focus on the algorithm that performs the pattern mining task in PPMT. We first present its design and implementation process, detailing the modifications we made to turn an existing implementation of the GSP algorithm into a progressive episode mining algorithm. While previous sections introduced the various steering types that are available in PPMT, the second subsection offers a detailed presentation of their implementation within the algorithm. In the third subsection, we compare our algorithm with the one from Stolper et al. (2014) that they used to illustrate the paradigm of Progressive Visual Analytics.

5.1 Design and implementation of the algorithm

When designing PPMT, we decided not to develop a new algorithm from scratch, allowing us to use existing implementations. Following our G1 guideline ("Extract episodes rather than sequential patterns", see section 5.2), we were looking for an episode mining algorithm, but no implementation of such algorithm was available at the time (in 2017). This is no longer the case, some being available since June 2018 in the SPMF library[6] (Fournier-Viger et al., 2016). We were looking for a well-known algorithm, and GSP (Srikant & Agrawal, 1996) was a good candidate. It is one of the first sequential pattern mining algorithms to have been created, and has served as a basis for many of the following ones. It also uses fairly simple data structures, which makes future modifications easier. Finally, it uses a breadth-first Apriori-like strategy, thus fulfilling our G3 guideline. As for the existing implementation, we used the one from SPMF. The following paragraphs describe the modifications we made to the SPMF implementation to make it progressive and compliant with our guidelines.

Extracting episodes rather than sequential patterns, and removing event sets mining. Since GSP is a sequential pattern mining algorithm, we needed to change its behavior to extract episodes instead. While GSP only keeps track of the sequences in which a pattern is found, we modified this behavior to explore the entire sequences, saving every occurrence found during the process. Doing so, we followed our first two guidelines ("G1: extract episodes" and "G2: save occurrences"). In order to better leverage the temporal dimension of our data, we decided to focus on serial episodes (see subsection 3.1.3). To do so, we had to modify the GSP implementation to use a data representation in the form of sequences of events rather than sequences of event sets.

Using an absolute support measure. In the SPMF implementation, GSP determines if a pattern is frequent or not based on a relative support measure expressing the idea that the pattern is found in *at least X% of the sequences*. While this can be relevant when working with sequential patterns, it became meaningless after moving to episode mining. To solve this problem, we changed the support measure to use the total number of occurrences, regardless of the number of sequences containing them.

Providing intermediate results. The SPMF implementation of GSP provides a parameter that determines whether the discovered frequent patterns are returned at the end of the computation, or at the end of the mining of each pattern size. While this second option technically provides intermediate results, any sufficiently large dataset will have every output take time and

[6] http://www.philippe-fournier-viger.com/spmf/

present the analyst with large numbers of patterns. To make PPMT more interactive and less cluttered, we decided to output every frequent pattern when all of its occurrences are discovered.

Providing additional parameters. The GSP implementation only allows two parameters, the *minimum support (minsup)* threshold to consider a pattern as frequent, and the *maximum length (maxlen)* beyond which patterns contain too many events to be interesting. As explained previously, we turned *minsup* from a relative to an absolute value, and added the following parameters:

- *Minimum* and *maximum gap* (see subsection 3.1.4.2) control the number of events allowed to be found between events forming a pattern occurrence, while not being part of the pattern. This provides more control over what one can consider "noise" allowed in the pattern occurrences. This constraint is verified during the mining process, when the algorithm is working on identifying the occurrences of a candidate pattern.
- *Maximum duration* (see *span* in subsection 3.1.4.2) of a pattern occurrence, in milliseconds. This effectively acts as the time window constraint that is often found in episode mining. This constraint is verified during the mining process, once a potential occurrence that complies with the gap constraints has been found.

Adding steering vectors. To make our algorithm completely progressive, and compliant with our fourth guideline ("G4: Offer steering on patterns, sequences and time"), we added three ways in which an analyst could be able to steer its computation. As explained in sections 6.2 and 6.3, steering is possible over specific sequences, over specific time periods and towards specific pattern prefixes. The following subsection provides a more detailed description of their implementations.

Provide feedback during the computation. To comply with our fifth guideline ("G5: provide information on the algorithm activity"), we made the algorithm output some information about its inner workings in addition to the frequent patterns. This happens for the following events:

- Start of a new steering, along with its target.
- End of a steering, along with the number of frequent patterns that have been found during the steering.
- Start of the extraction of a new pattern size, along with the number of candidates that have been generated.
- Every time a given number of candidates have been explored. This provides information about the algorithm's progression, even if no new frequent patterns are discovered.

5.2 Steering the algorithm

In order to comply with our G4 guideline for progressive pattern mining algorithms, PPMT allows the analyst to steer the pattern extraction towards specific sequences, time periods and pattern syntaxes.

Within PPMT, steering towards a pattern syntax can be done by specifying an already discovered pattern, thus prioritizing the search for patterns that have it as a prefix. From an algorithm standpoint, it is the easiest steering form to implement. When a new candidate is selected, its syntax is compared to the prefix. If it complies with the steering target, the algorithm looks for occurrences as if no steering was happening. Otherwise, the pattern is discarded and the algorithm will go back to it once the steering is complete.

Steering towards a time period or a sequence is performed differently, since knowing the candidate syntax is no longer enough to determine whether it complies with the current target or not. Under such steering, verifying a candidate starts with the algorithm scanning the steering target for occurrences of the candidate pattern. If none is found, the candidate is discarded. If at least one occurrence is discovered, the algorithm continues the occurrence search over the entire data, as if no particular steering was occurring.

To illustrate this process, let's consider the sequences in figure 6.29, when a time steering targeting the time period between t_6 and t_{11} occurs at the start of the mining process. Using a maximum gap of 0 and a minimum support of 5, events of type A and B are both frequent, which leaves us with four candidates of size 2 to look for: <A;A>, <A;B>, <B;A> and <B;B>. The algorithms starts with candidate <A;A>, focusing on the steering target, and finds one occurrence (AA_1). Since the candidate is present within the steering target, the search for additional occurrences of the candidate is performed on the whole dataset, ending up with two occurrences. The candidate is thus declared infrequent, and the algorithm continues with the next candidate, <A;B>. Looking at the steering target, two occurrences of <A;B> are found (AB_2 and AB_5), triggering the search over the complete dataset that leads to five occurrences, thus validating <A;B> as a frequent pattern. The next candidate, <B;A>, is found two times within the steering target (BA_2 and BA_3), which starts the mining over the whole dataset. Since five occurrences of <B;A> are found, it is validated as a frequent pattern. The algorithms then continues with the last candidate, <B;B>, but does not find any occurrence within the steering target. The candidate is thus discarded for the current steering, regardless of how many occurrences exist in the data[7]. The next step is to generate candidates of size 3, and continue this process until no frequent patterns are found. At this point, the steering ends, and the algorithm goes back to check the candidates that have been discarded during the process. In this case, this means only the candidate <B;B>, since <A;A> has been considered infrequent after a search for occurrences over the complete data, not just within the steering target.

[7] In fact, five occurrences of <B;B> can be found in the complete data, making it a frequent pattern.

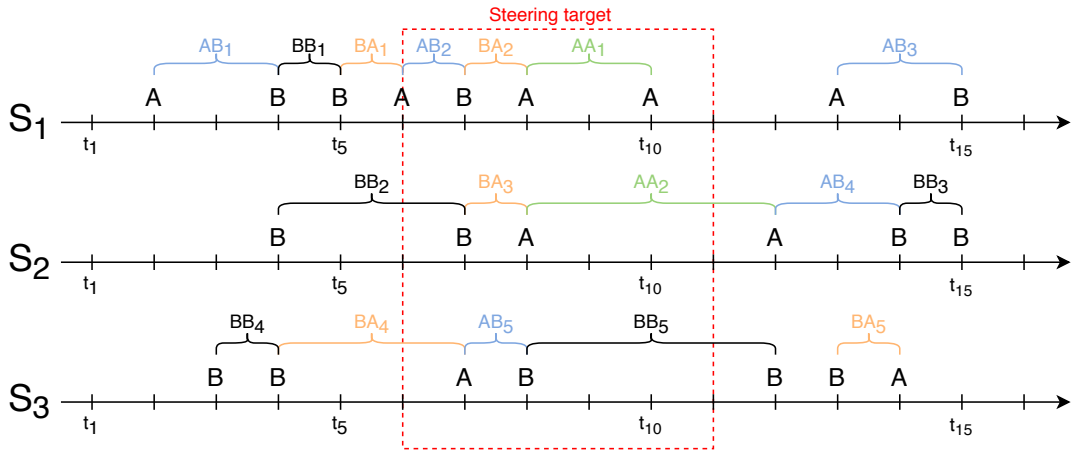


Figure 6.29: An example dataset containing 3 event sequences. *Occurrences of candidate patterns are indicated by accolades above the sequences.*

An important thing to note is that only one steering can be happening at a given time. We made this choice to work with simple steering options, even though one could consider steering the algorithm towards a combination of these criterions (for example, towards a prefix in a specific time period). In the current implementation, a steering request from the client is saved as a "pending" steering. Every time the mining algorithm is done extracting the occurrences of a candidate pattern, and before moving on to the next candidate, it checks if there is a pending steering. If it is indeed the case, it becomes the current steering, with its target being set accordingly. Should a steering be requested when one is already occurring, the previous one will be cancelled and the algorithm will start steering towards the new target.

We implement steering using the *algorithm manager* module (see subsection 6.4.2). Before the algorithm starts processing a new candidate, a verification is made to see if a steering has been requested since the previous check. If it is the case, the current state of the pattern search (the current size and the list of unverified candidates) is saved, and the candidate verification continues according to the required steering, as described in the previous paragraphs. During the steering, since our knowledge about frequent patterns of a given size is incomplete, candidates of size n are no longer obtained by combining frequent patterns of size $n-1$. Instead, candidates of size $n-1$ are combined with the frequent event types to generate the size n candidates. When the steering has completed, the previously saved state of the pattern search is loaded, information about the frequent patterns found during the steering is added to it, and the default strategy resumes from this point.

5.3 The algorithm

The progressive sequential pattern mining algorithm within PPMT uses the following logic:

```
initialize frequentSet with the frequent items types;
lastSizeCompleted ← 1;
lastFrequentSetCompleted ← frequentSet;
currentSize ← 1;
levelEndedWithNoCandidate ← false;
/* Loop until the default strategy (i.e. without steering) has completed for all pattern sizes */
while lastSizeCompleted < maxSize do
    noCandidateFound ← false;
    noFrequentCandidate ← false;
    steeringHasOccurred ← false;
    /* Main mining loop, each pass extracts frequent patterns of size currentSize */
    while frequentSet is not empty AND currentSize < maxSize do
        increment currentSize;
        notify the client that the mining of a new pattern size begins;
        if steeringIsOccurring then
            candidateSet ← generateCandidatesByCombination();
            steeringHasOccurred ← true;
        else
            candidateSet ← generateCandidatesFromFrequentSet();
        if candidateSet is empty then
            levelEndedWithNoCandidate ← true;
            noCandidateFound ← true;
            break;
        notify the client that candidates have been generated;
        empty frequentSet;
        /* Check every candidate to see if they are frequent or not */
        foreach candidate : candidateSet do
            notify the client that a new candidate is being checked;
            if candidate is already a known frequent pattern then
                /* Can happen if the pattern has been found during a previous steering */
                add candidate to frequentSet;
            else
                if steeringIsRequested then
                    steeringHasOccurred ← true;
                    steeringIsOccurring ← true;
                    notify the client that a steering starts;
                if steeringIsOccurring then
                    validCandidate ← checkCandidateWithinSteeringTarget();
                if validCandidate OR not steeringIsOccurring then
                    extract occurrences of candidate from the data;
                    if support of candidate ≥ minSupport then
                        add candidate to frequentSet;
                        save occurrences of candidate;
                        notify the client that candidate is a frequent pattern;
```

```
continuation of the first while loop
    continuation of the main mining loop
        /* Exit the main mining loop if no frequent pattern of size currentSize has been found   */
        if frequentSet is empty then
            noFrequentCandidate ← true;
            break;
        /* Only consider patterns of size currentSize as fully extracted if no steering has
           occurred                                                                              */
        if not steeringHasOccurred then
            lastSizeCompleted = currentSize;
            lastFrequentSetCompleted = frequentSet;
            notify the client that patterns of size currentSize have been fully mined;
    if noCandidateFound OR noFrequentCandidate then
        if steeringHasOccurred then
            /* The main mining loop ended and a steering occurred, we go back to the last pattern
               size completed with the default strategy                                          */
            currentSize ← lastSizeCompleted;
            frequentSet ← lastFrequentSetCompleted;
            notify the client that a steering has ended;
            steeringHasOccured ← false;
        else
            /* The main mining loop ended without a steering occurring, all the frequent patterns
               have been found                                                                   */
            notify the client that patterns of size currentSize have been fully mined;
            break;
```

Within this algorithm, we refer to three particular functions. During the default strategy, `generateCandidatesFromFrequentSet` generates new candidate patterns of size k by combining frequent patterns of size $k-1$, as done in the GSP algorithm (Srikant & Agrawal, 1996). This candidate generation process limits the number of candidates that will end up being infrequent, but requires that all patterns of size $k-1$ have been discovered. Since this is not guaranteed during a steering, `generateCandidatesByCombination` generates new candidates of size k by extending frequent patterns of size $k-1$ with the frequent event types. Finally, `checkCandidateWithinSteeringTarget` is used to verify that a given candidate complies with the current steering target. As explained in subsection 6.5.2, this can be done by reading the candidate syntax in the case of a steering on pattern, or by reading a subpart of the data in the case of a steering on sequence or time.

5.4 Comparison with existing Progressive Pattern Mining algorithms

5.4.1 Comparison with Stolper et al. (2014)'s algorithm

When considering the existing literature, Stolper et al. (2014)'s work is the first to propose a progressive pattern mining algorithm, and our algorithm share some similarities with it. Both use a breadth-first Apriori like mining strategy, even though we started from an existing breadth-first algorithm (GSP) while they modified SPAM, a depth-first algorithm. Both implementations allow the analyst to steer the computation towards a given pattern prefix, but Stolper et al. also provide the ability to forbid a specific prefix.

While Stolper et al. used their algorithm to illustrate the principles of Progressive Visual Analytics, our focus was on creating an algorithm that could be used within a system that supports not just pattern exploration, but also data exploration. This important distinction is the reason for the two main differences between our algorithm and Stolper et al.'s. First, we provide additional steering options, allowing one to target specific time periods and sequences, thus putting constraints on the data and not on the patterns. Secondly, we extract episodes rather than sequential patterns, thus providing more information about when the patterns occur, and the relevant context.

5.4.2 Comparison with Servan-Schreiber et al. (2018)'s algorithm

Servan-Schreiber et al. (2018) propose ProSecCo, a progressive frequent pattern mining algorithm. The algorithm processes data by chunks, returning an increasingly better approximation of the list of frequent patterns after each chunk. While our algorithm is derived from GSP, ProSecCo is not a modification of an existing non-progressive algorithm, even though it uses the non-progressive PrefixSpan algorithm (Pei et al., 2001) to handle the first chunk of data.

It should be noted that even though the authors describe ProSecCo as a *progressive* algorithm, they do not provide any way to interact with its execution. Furthermore, their implementation was only used to assess the algorithm's performances compared to other pattern mining algorithms, not to conduct an actual data analysis.

6 Evaluations

In order to conduct a thorough evaluation of our Progressive Visual Analytics system, we decided to focus on three different aspects. The first is an evaluations of PPMT's compliance with existing recommendations for building Progressive Visual Analytics systems, by reviewing each existing recommendation and discussing its implementation within our tool. The second is about the performances of the underlying progressive algorithm, by observing the impact of progressiveness on the algorithm's performances using different configurations. Finally, the third aspect is PPMT's usefulness to an analyst exploring a dataset, for which we conducted a user study. In this section, we relate the first two of these evaluations, while the last one is the focus of the next chapter.

6.1 Compliance with existing recommendations

In its current version, PPMT fully complies with 13 out of 18 of Badam et al. (2017)'s recommendations for the design of Progressive Visual Analytics systems. Of the remaining 5, 2 of them are partially implemented, and the last three are not, because we felt that they were not important for our use case. In the next paragraphs, we provide a more detailed explanation for

each of the recommendations, while table 6.1 provides an overview of PPMT's compliance with them.

Table 6.1: Compliance of PPMT with existing requirements for the design of Progressive Visual Analytics systems. *Requirements are either fully (✓), partially (✓) or not (×) implemented.*

Requirements	In PPMT
R1: Meaningful partial results	✓
R2: Structure-preserving intermediate results	✓
R3: Retaining cognitive workflow on updates	✓
R4: Minimized distraction during updates	✓
R5: Cues for new results	✓
R6: Aggregated information	✓
R7: Uncertainty	✓
R8: Provenance information on demand	✓
R9: Aliveness	✓
R10: Absolute progress	✓
R11: Relative progress	✓
R12: Full interactivity	✓
R13: Support two modes: constant update and on-demand refresh	✓
R14: Steer results	✓
R15: Cancellation	×
R16: Prioritization	✓
R17: Provide similarity anchors in complex visualizations	×
R18: Consistently offer quality measures	×

6.1.1 Fully implemented recommendations (13/18)

Meaningful (R1) and structure-preserving (R2) partial results. Working with patterns is enough to fulfill these two recommendations, since "results" are the patterns and their occurrences. This ensures a common structure since occurrences always are a series of event types. Each occurrence is also meaningful, in that it describes a piece of data that can be related to a behavior (in the sense of a sequence of actions).

Retaining cognitive workflow (R3) and minimizing distractions (R4) on updates. In PPMT, updates happen when a new frequent pattern is discovered and received by the client. More so, these updates only involve updating the display of the number of discovered patterns and adding the pattern to the lists of patterns, either on the side or in some of the tooltips, since only selected patterns have their occurrences displayed in the central timeline. As such, distractions are minimized with regards to most of the user interface. Should one be focusing on the patterns lists, it is possible to reduce these distractions even more in two ways: filtering the patterns to

only display the ones the analyst is interested in, and pausing the automatic addition of newly discovered patterns to the list.

Support two modes: on-demand refresh and constant update (R13). As explained in the previous paragraph, the analyst is able to toggle on or off the constant update of the pattern list. This allows her to choose between having the more up-to-date information or having a stable list, should she want to explore its content. When the constant update is toggled off, the number of patterns discovered but not added to the list is displayed, and the analyst can either include these patterns into the list or resume the constant update (which also adds the missing patterns into the list).

Cues for new results (R5) and aggregated information (R6). PPMT presents aggregated information about the frequent patterns in two ways. The first is by displaying the number of discovered patterns, either as a whole (in total and in the pattern list, both before and after the application of filters) or for each pattern size (labels in the bar chart in the top right). The second way of presenting aggregates are the bar charts themselves, mainly the one that represents the number of patterns of each size (both in the top right and above the *pattern size* slider). Another bar chart is available above the *pattern support* slider, however the reduced size of this element and the large scale it requires can quickly make it too small to be noticeable, depending on the dataset. When new results are available, these aggregates are updated to provide a cue to the analyst.

Uncertainty (R7). The only aspect of PPMT that presents the analyst with uncertain results is the bar chart that shows how many candidates have to be verified for a given pattern size. More specifically, when performing a steering, the candidates are generated within the bounds of its target. While this provides an indication of the remaining “in steering” work, additional candidates can be generated later, when the algorithm goes back to its default strategy (or during other steering requests). To communicate this information to the analyst, we use a color code in our bar chart that echoes the color of the algorithm’s current status display. For a given pattern size, green means that the information is complete and nothing more can be discovered, while orange means that the information presented comes from a steering, thus possibly being incomplete. Regardless of whether a steering is occurring or not, the blue color indicates the pattern size the algorithm is currently working on.

Aliveness (R9). The algorithm’s aliveness is communicated by displaying the elapsed time since the start of its execution, that will remain constant if the algorithm has ended. The ex-

panded algorithm state view also presents two graphs that respectively indicate the number of frequent patterns found per second and the number of candidates checked per second, thus providing the analyst with a rough estimate of the algorithm's "speed". However, these metrics need to be considered as cues rather than absolute truths, since not finding any pattern nor checking any candidate for a few seconds could indicate that one particular candidate takes several seconds to be checked (given sufficiently large data). Another major indicator of the algorithm's aliveness is the display of its current strategy, which can be either "default strategy" or "steering on *[steering target]*" if the algorithm is running, or "not running" when it is not.

Full interactivity (R12). PPMT provides full interactivity with the discovered patterns by allowing the analyst to sort, filter and highlight them. These options are also available for the underlying data that can be sorted, filtered and highlighted based on the *user* and *event type* properties.

Steer results (R14) and prioritization (R16). The analyst is able to interact with the algorithm in multiple ways. While it can be restarted using new parameter values, the most interesting interactions with regards to these recommendations are the steering options. The analyst can prioritize the pattern mining towards three types of target: a requested prefix for the pattern syntax, a time period where the patterns must be found, and a user sequence where the patterns must be found.

Provenance information on demand (R8). Frequent patterns discovered by the algorithm can either be discovered by the default strategy or during a steering. At any time, the analyst can see which of these two states the algorithm is in in the top right part, which provides a clue for the provenance of future patterns. Once a pattern is known, no distinction is made between "how" they were extracted. However, the analyst is able filter the list of patterns to only show patterns discovered during the current steering (or during the last one if the algorithm is in the default state). As explained in section 6.2, this exception comes from the reasoning that, since requesting a steering is a conscious action from the analyst, it can indicate an interest in the patterns that will result from it. As such, providing an easier access to these patterns is supposed to help the analyst with her current goals and interests.

6.1.2 Partially implemented recommendations (2/18)

Absolute (R10) and relative (R11) progress. The algorithm's absolute progress is communicated to the analyst in the form of the number of discovered patterns (both in total and for each pattern size) and a display the elapsed time since it started running. Relative progress

is provided in the extended algorithm view, where the number of checked candidates and the total number of candidates is displayed for each already started pattern size. While summing up these values could provide some insight about the remaining work the algorithm has to perform, it is possible for larger pattern sizes to yield frequent patterns. As such, the analyst has no way to know if she sees all the pattern sizes. Providing a relative progress of the overall algorithm would require building an estimate of either the number of candidates or of the remaining execution time, neither of which is easily done (if at all possible). The maximum number of patterns is known due to its combinatorial nature, but the real number of candidates is always far lower, making its display irrelevant and misleading. As for an estimate of the remaining time, the speed at which candidates are verified depends on the data and varies from candidate to candidate, which would most certainly lead to a incorrect estimate.

6.1.3 Recommendations not implemented (3/18)

Provide similarity anchors in complex visualizations (R17). Due to our focus on interactions between the algorithm and the analyst, we used timeline-based visualization to display the data and patterns. Our timelines are displayed in a vertical stack, and we made sure that the x-axis, that represents time, is always the same across all of them. Since these types of data visualization are not complex, we did not feel the need for similarity anchors that would allow one to follow data between different timelines beyond these synchronized axes.

Consistently offer quality measures (R18). Quality measures are supposed to provide the analyst information about the uncertainty that may occur in the data representations. However, PPMT represents information without uncertainty. All the data is available and displayed right from the start, and patterns are either unknown (thus not displayed) or known to be frequent once all of their occurrences have been discovered. As such, we did not feel the need for such quality measures.

Cancellation (R15). Being able to cancel the algorithm can be useful if one knows that additional results from the algorithm will not be needed, either because the already available information is enough for the task at hand, or because it is clear that future information will not be valuable. In such cases, cancelling the algorithm prevents the analyst from being disturbed or distracted by unnecessary visual updates. In PPMT, discovering more patterns do not clutter the visualization since they are not automatically displayed. Should one be bothered by their addition to the pattern list, it is possible to stop the constant update of the list by the click of a button, essentially cancelling the algorithm from the point of view of the analyst. Due to this, we did not feel that providing a way to cancel the algorithm was necessary.

6.2 Performances of a progressive pattern mining algorithm derived from an existing algorithm

Here we present the second evaluation we conducted, aiming at studying the impact of progressiveness on an algorithm. Since providing intermediate results and offering steering options is additional work compared to a non-progressive version of the algorithm, we were interested in what could be named the "cost of progressiveness" was. We first present our protocol, before presenting the results and discussing them.

6.2.1 Protocol

To explore the impact of progressiveness on an algorithm, we wanted to take advantage of the fact that our algorithm was derived from a non-progressive one. This provides us with a baseline, against which we can compare several strategies by introducing variations on the progressive aspects. While we expected the additional work induced by the progressive features, the main uncertainty was the impact that performing a steering would have on the computation. We ran and compared the following four configurations:

1. The GSP implementation from SPMF, only modified to extract episodes rather than sequential patterns
2. Our progressive algorithm derived from GSP, without performing any steering
3. Our progressive algorithm derived from GSP, while performing syntax steering
4. Our progressive algorithm derived from GSP, while performing time steering

In all configurations, we store both the frequent patterns and their occurrences. We decided not to add a configuration where our algorithm would perform user steering due to its similarity with the time steering one, as explained in subsection 6.5.2. In configurations 3 and 4, steering was performed on a given pattern or time-period, once for every pattern size, which amounts to up to 15 steering phases per execution. Considering that an execution takes between 3 and 20 minutes, we believe that it will allow us to highlight any impact that steering could have on the algorithm, while gaining insights about the performances between the different configurations.

We used the *coconotes* dataset, presented in subsection 6.1.3. It consists of traces of students using COCoNotes[8], totaling 201 000 events, 32 event types and 211 users over a four months period from September 2016 to January 2017. Regarding the parameter values, we kept a constant gap (*mingap* = 0, *maxgap* = 2), maximum size (*maxsize* = 20) and maximum duration (*maxduration* = 30s), but considered three scenarios by using different support thresholds:

[8] https://coconotes.comin-ocw.org/

1. *minsup* = 150, which led to 3492 frequent patterns
2. *minsup* = 50, which led to 21731 frequent patterns
3. *minsup* = 30, which led to 49121 frequent patterns

Each configuration ran each scenario 20 times on a single dedicated CPU core. We measured the CPU time and maximum memory usage needed to extract all the patterns from the dataset using the debug tools provided with SPMF's GSP implementation.

6.2.2 Results and discussion

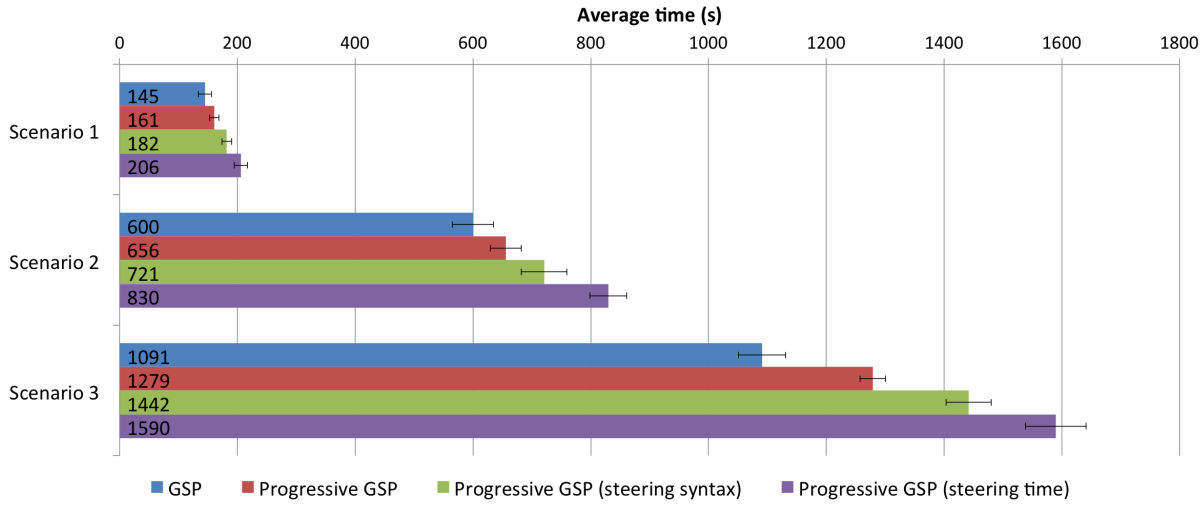


Figure 6.30: Results of the comparison between the different algorithms, regarding the CPU time necessary to discover every pattern.

Figure 6.30 presents the average time (in seconds) that each configuration needed to complete the different scenarios. Overall, the relation between the configurations is the same from one scenario to another, and was as we expected. The non-progressive algorithm is the fastest, followed by the progressive configuration without steering. This proximity can be explained by the fact that as long as no steering is performed, the progressive version is essentially the same as the non-progressive one, only adding a check for a pending steering request when switching to a new candidate. Having steering on syntax be faster than steering on time was also expected based on their respective implementation, as explained in subsection 6.5.2. A final remark on execution time is that while the difference is relatively small in scenario 1, the gap between the configurations increases when the parameters are relaxed to obtain more

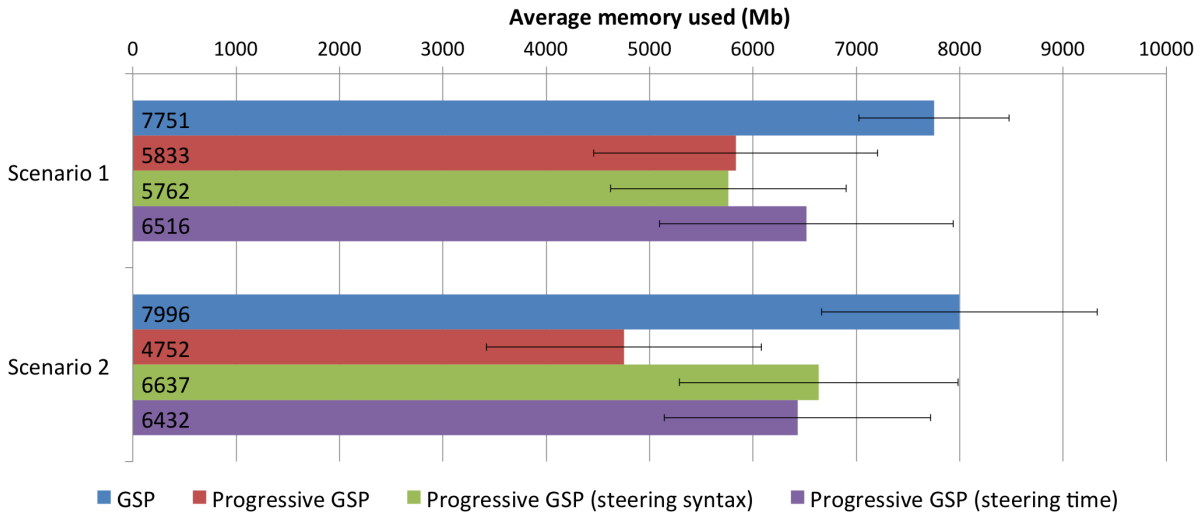


Figure 6.31: Results of the comparison between the different algorithms, regarding the maximum memory usage to discover every pattern. *Results are missing for the third configuration, due to a failure of the memory-measuring tools.*

patterns. As such, experimenting with even larger scenarios could be interesting, to see if our observations are still valid.

With regards to maximum memory usage, the results are presented in figure 6.31. Our main takeaway is that the non-progressive algorithm uses more memory, while the three steering variants do not show any meaningful variation. This was the result we expected, since the non-progressive algorithm has to maintain the full memory of occurrences until the end of its execution, while the progressive ones can remove parts of it once they have been outputted. However, when taking into account a complete data analysis system, patterns need to be memorized somewhere else to be leveraged. As such, while progressiveness can be more memory efficient if one only wants to output a list of pattern, an interactive environment will still require the necessary memory space in another part of the system.

This experiment allowed us to investigate the impact on the system's resources of a progressive algorithm. Results ended up confirming our predictions, highlighting the fact that the ability to steer the algorithm can have a rather significant impact memory wise. It should however be noted that we did not go out of our way to optimize our algorithm, and had to work within the limits of the existing implementation we used as a basis. As such, this experiment could benefit from being performed with a more carefully designed progressive algorithm.

7 Conclusion

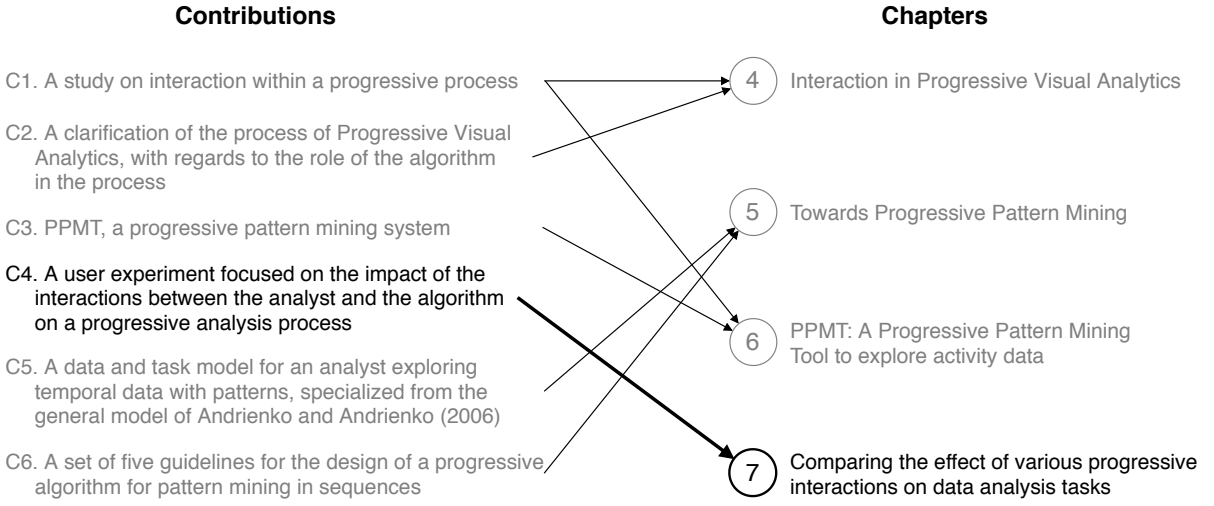
In this chapter, we presented the practical work we conducted while implementing PPMT, a Progressive Pattern Mining Tool designed to explore activity data. As explained in the first section about our design process, it covers a two-year timeframe and constitutes an important part of our study of interaction between an analyst and a progressive algorithm.

Implementing PPMT allowed us to put our theoretical propositions in practice and experiment with them. At the same time, it provided us with a first-hand experience about the design and implementation of a progressive algorithm. In its current version, it offers the usual features one can expect from a data visualization tool, such as being able to visually represent the dataset and explore it by panning and zooming, adapted to a context where additional information can be provided and requested through a progressive algorithm. Several ways to interact with the algorithm are implemented, that cover every case of the interaction framework we proposed in section 4.1. This allowed us to use it to conduct user studies that are described in more details in the next chapter, in order to receive feedback about our work and to observe how real users were interacting with such system.

CHAPTER 7

COMPARING THE EFFECT OF VARIOUS PROGRESSIVE INTERACTIONS ON DATA ANALYSIS TASKS

After the implementation of PPMT, the two evaluations presented in the previous chapter (see section 6.6) were intended to confirm its compliance with existing guidelines for the design of such system, and the effectiveness of the underlying Progressive Pattern Mining[1] algorithm. Here we present the third evaluation we conducted using PPMT, which targets the interestingness of the tool itself, from the point of view of a human analyst working with data. This constitutes our fourth major contribution.



Our aim was to study the interactions between the analyst and a progressive algorithm, since we concluded our review of the literature by highlighting the fact that no such review existed. In this evaluation, our focus was the interaction between the analyst and the algorithm,

[1] As explained in section 6.2, PPMT only supports episode mining, in accordance with our G1 guideline for Progressive Pattern Mining algorithms (see section 5.2). As such, in this chapter the terms “pattern” and “sequential pattern” refer to episodes

not between the analyst and the available visualization. For this reason, this experiment was designed with the framework for interactions we presented in subsection 4.1.1 in mind, and we observed the impact these actions can have on the analysis process.

The content of this chapter extends the results we provide in Raveneau et al. (2018), since the experiment was still ongoing at the time of our publication. The experiment protocol did not change, although our article is focused on a subpart of the questions and was based on the answers of the 14 participants we had at the time. We first present our protocol, before going over the results and discussing them.

1 Material and protocol

1.1 Experiment material

To conduct this experiment, we used two versions of PPMT:

1. The complete system, as described in the previous chapter;
2. The complete system stripped of its steering capabilities.

We used the *coconotes* dataset, described in subsection 6.1.3. It contains the anonymized activity data of students using the COCoNotes[2] video annotation platform. It spans a four months period from September 2016 to January 2017, totaling 201.000 events, 32 event types and 211 users. Event types range from login and navigation actions to interactions with the video player (play, pause, change volume, move to a specific timestamp...), and annotation management (create, edit, share...) events.

1.2 Experiment protocol

Evaluating PPMT's ability to support an exploratory data analysis process would have required several people interested in exploring the dataset over a long period of time. Since such people were not available, and considering that an exploratory analysis alternates between exploration phases and verification of hypotheses, we decided to focus on this second aspect of the data analysis. To do so, we selected specific questions that an analyst may end up asking themselves during an analysis, and asked participants to answer them.

We recruited 23 participants, 18 men and 5 women, between 20 and 48 years old ($\overline{age} = 25.2$; $\sigma = 7.05$), that were all familiar with data analysis. We distributed them between two groups:

- The *steering* group, having 11 members, used the first version of PPMT;

[2] http://coconotes.comin-ocw.org/

- The *non-steering* group, having 12 members, used the second version of PPMT.

A participant's session lasted about an hour. They signed a consent form, were given some reminders about pattern mining along with a presentation of PPMT's features and user interface. They then had to use PPMT to answer the following two batches of questions.

- First batch, 5 questions:

  Q1.1 How many occurrences of the pattern "`paused played VisibilityChange`" are present in the data?

  Q1.2 How many users have the pattern "`paused created played`" in their traces?

  Q1.3 How many patterns are discovered for user *user049* for the 9:02 to 9:11am session of October 5th?

  Q1.4 Give two patterns with size > 2, present both in the first session of user *user153* and in the first session of user *user177* (September 18th, from 9:38am to 11:15am).

  Q1.5 We are interested in the events "`X`" that can follow or precede the pattern `M` = "`Mdp_media_play played`". How many "`X M`" or "`M X`" patterns are there?

- Second batch, 3 questions:

  Q2.1 How many occurrences of the pattern "`paused played Mdp_media_pause paused`" are present in the data?

  Q2.2 How many patterns are discovered for user *user049* in his first session on October 2nd (from 12:14 to 14:03)?

  Q2.3 We are interested in the events "`X`" that can follow or precede the pattern `M` = "`PlayerStart Mdp_media_volumechange`". How many "`X M`" or "`M X`" is there?

Questions had to be answered in order, and the algorithm and user interface (filters and selections) were reset between each question, allowing us to analyze each question's answers independently. Participants were not able to perform any *modifying data* action during the first batch, the feature being explained between the two batches. After answering these questions, the participants had to rate 16 affirmations on a 5-points Likert scale. These were targeting their overall feedback (3 affirmations), progressive pattern extraction (4 affirmations), data modification (5 affirmations) and their perception of the algorithm (4 affirmations). Members of the *steering* group had 7 more affirmations to rate that were about the action of steering the algorithm. For every participant, their session concluded with a discussion where they were asked about their use of PPMT: why they did (or did not) use the *steering* or *modifying data* actions, and what they were expecting from it. They were also able to freely comment on their experience.

Each question targets a specific action from our framework, which allows us to compare the results between the two batches of questions for a given targeted action, as shown in table 7.1. We designed questions 1.1, 1.2 and 2.1 to encourage *constraining the algorithm on results* (pattern syntax), *i.e.* action A3. Questions 1.3, 1.4 and 2.2 were targeting *constraining the algorithm on data* (a time period), *i.e.* action A2. Question 2.3 was designed to encourage *modifying data*, *i.e.* action A1, with question 1.5 being its counterpart when this option is not available. Notice that *constraining the algorithm on results* would only bring half of the expected results to these two questions, since PPMT's constraining on results only deals with prefixes, thus only helping with the "`M X`" part of the questions.

Table 7.1: Correspondence between the two batches of questions. *The last column indicates the action we were targeting with each question.*

Batch 1	Batch 2	Targeted action
Q1.1	Q2.1	Constraining on results (A3)
Q1.2		
Q1.3	Q2.2	Constraining on data (A2)
Q1.4		
Q1.5	Q2.3	Modifying data (A1)

Overall, when combining the two version of PPMT and the two batches of questions, we ended up with four configurations regarding the various actions participants could do, as presented in table 7.2. Since the experiment was already long, and packed with a lot of information to process for the participants, we restricted the scope of some of the interactions. *Constraining the algorithm on data* was limited to targeting a time period, and *modifying data* was limited to the creation of new event types from frequent patterns. We decided not to include the ability to *change the parameters* because performing meaningful and educated parameter changes would have required participants already familiar with sequential pattern mining.

Table 7.2: The four configurations of actions available for the participants. *Actions are expressed as part of our framework presented in subsection 4.1.1.*

	Batch 1 – Without *modifying data*	Batch 2 – With *modifying data*
Steering group	A2, A3	A1, A2, A3
Non-steering group	*(only view exploration)*	A1

1.3 Collected data

During their use of PPMT, we measured the time participants took to answer each questions, the correctness of their answers with regards to predefined expected values, and their use of the available interactions. We also collected their rating of the 16 (23 for the *steering* group) affirmations, and their remarks during the final interview.

2 Results

2.1 Time to answer questions

The time taken by participants to answer each question is presented in figure 7.1. By looking at the median time we can see that members of the *steering* group were faster for every question, except for 2.2. In order to verify our intuition that steering the algorithm leads to faster answers, we computed Welch's t-test for each question, which showed statistical significance for question 1.2 ($p = 0.036$) and 1.3 ($p = 0.03$).

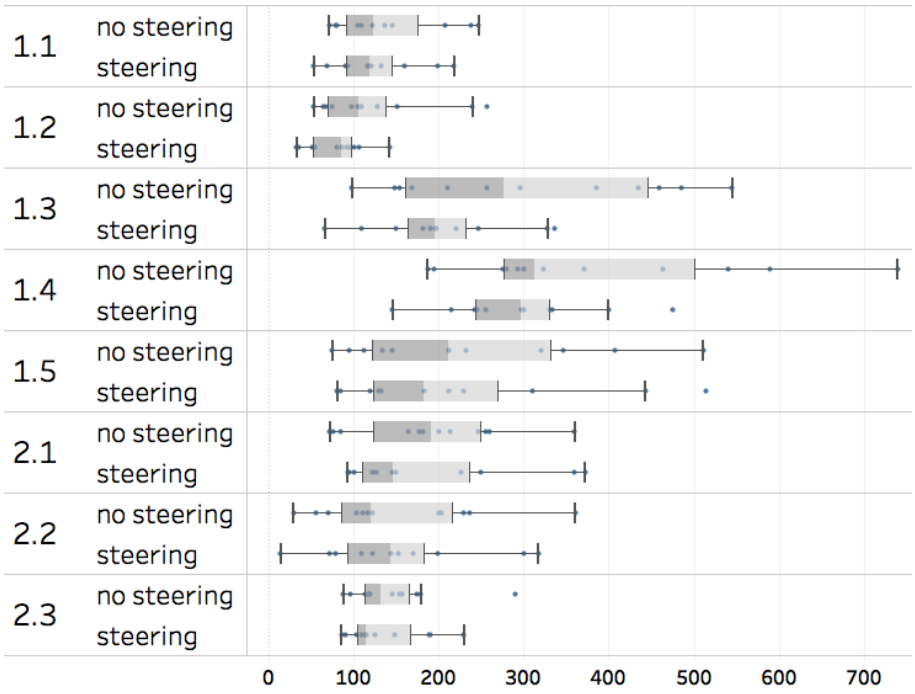


Figure 7.1: Time (in seconds) needed to answer questions.

2.2 Correctness of answers

Data on the answers' correctness is shown in table 7.3. It shows that questions 1.1, 1.2, 1.4 and 2.1 are mostly answered correctly, however looking at the actual answer reveals that the majority of the wrong ones come from misreading the question. The main example is in question 1.4, with a confusion between the "$>$" and "$\geq$" comparison operators. One member of the *non-steering* group is responsible for 3 out of the 5 absences of answer for this group. Even though participants from the *steering* group were slightly more correct, Welch's t-test did not show any significant relation between the number of wrong answers and the participant's ability to *steer* the algorithm.

Table 7.3: Correctness of answers for each question across all participants. *The data indicates the percentage of wrong - right - no answers. Detailed numbers of answers are given between parentheses.*

	Non-steering group (12 members)	Steering group (11 members)
Q1.1	25% - 75% - 0% (3 - 9 - 0)	0% - 100% - 0% (0 - 11 - 0)
Q1.2	25% - 75% - 0% (3 - 9 - 0)	18% - 82% - 0% (2 - 9 - 0)
Q1.3	67% - 25% - 8% (8 - 3 - 1)	64% - 36% - 0% (7 - 4 - 0)
Q1.4	42% - 50% - 8% (5 - 6 - 1)	18% - 73% - 9% (2 - 8 - 1)
Q1.5	50% - 42% - 8% (6 - 5 - 1)	55% - 45% - 0% (6 - 5 - 0)
Q2.1	25% - 67% - 8% (3 - 8 - 1)	18% - 82% - 0% (2 - 9 - 0)
Q2.2	75% - 17% - 8% (9 - 2 - 1)	36% - 64% - 0% (4 - 7 - 0)
Q2.3	50% - 50% - 0% (6 - 6 - 0)	45% - 55% - 0% (5 - 6 - 0)
TOTAL	45% - 50% - 5% (43 - 48 - 5)	32% - 67% - 1% (28 - 59 - 1)

2.3 Interaction with the algorithm

Table 7.4 presents the number of actions performed by the participants (*modifying data* and *steer algorithm*). As expected, questions 1.1, 1.2 and 2.1 led to *steering on results* while questions 1.3, 1.4 and 2.2 led to *steering on data*. All members of the *steering* group performed

a *steering* of the algorithm at least once, and most of them did it once per question. There are however two notable exceptions, the first being two participants that did two *steering on time* at question 1.3. They started by quickly targeting an approximation of the requested time period, which they used to perform a first steering. While it was running, they took their time to carefully target the requested time period. As soon as they where satisfied with their selection, they started a second steering towards it. The second exception is with three members of the *steering* group that respectively performed 3, 3 and 4 times the same *steering on results* for question 2.1, before they realized the pattern they were expecting was already discovered but had been hidden by their current filters.

Regarding *data modification*, 5 participants from the *non-steering* group and 7 from the *steering* group performed this action once. It happened either at question 2.1 or 2.3, except for one user in each group that did it in both questions.

Table 7.4: Number of times each action has been performed across all participants for each question. *Greyed-out cells indicate that the action was not available to the participants.*

Group	Action	1.1	1.2	1.3	1.4	1.5	2.1	2.2	2.3	TOTAL
Non-steering	*Modifying data*						2	0	4	6
Steering	*Modifying data*						2	0	6	8
	Steering	6	7	7	6	8	18	8	6	66

2.4 Affirmation ratings

Results from the participants' ratings of the affirmations are presented in figure 7.2. About the global feedback, participants were unanimous about the ability of PPMT to allow data and pattern exploration. Most also estimated that progressively extracting the patterns was useful, and 21 stated that they were able to know what the algorithm was doing at any moment. Regarding the feedback on PPMT's ability to help estimate the remaining analysis time, the participants are divided. *Data modification*, on the other hand, is identified as a useful feature, both helping save time and familiarize with the data. Finally, participants from the *steering* group unanimously reported that *steering the algorithm* helps save time and made them feel in control of the algorithm, with seven of them considering it mandatory for an efficient analysis.

3 Discussion

3.1 Impact of interactions on answer time

One of our main hypotheses in this experiment was that being able to steer the algorithm (either on results or on data) would help the *steering* group answer questions faster. This

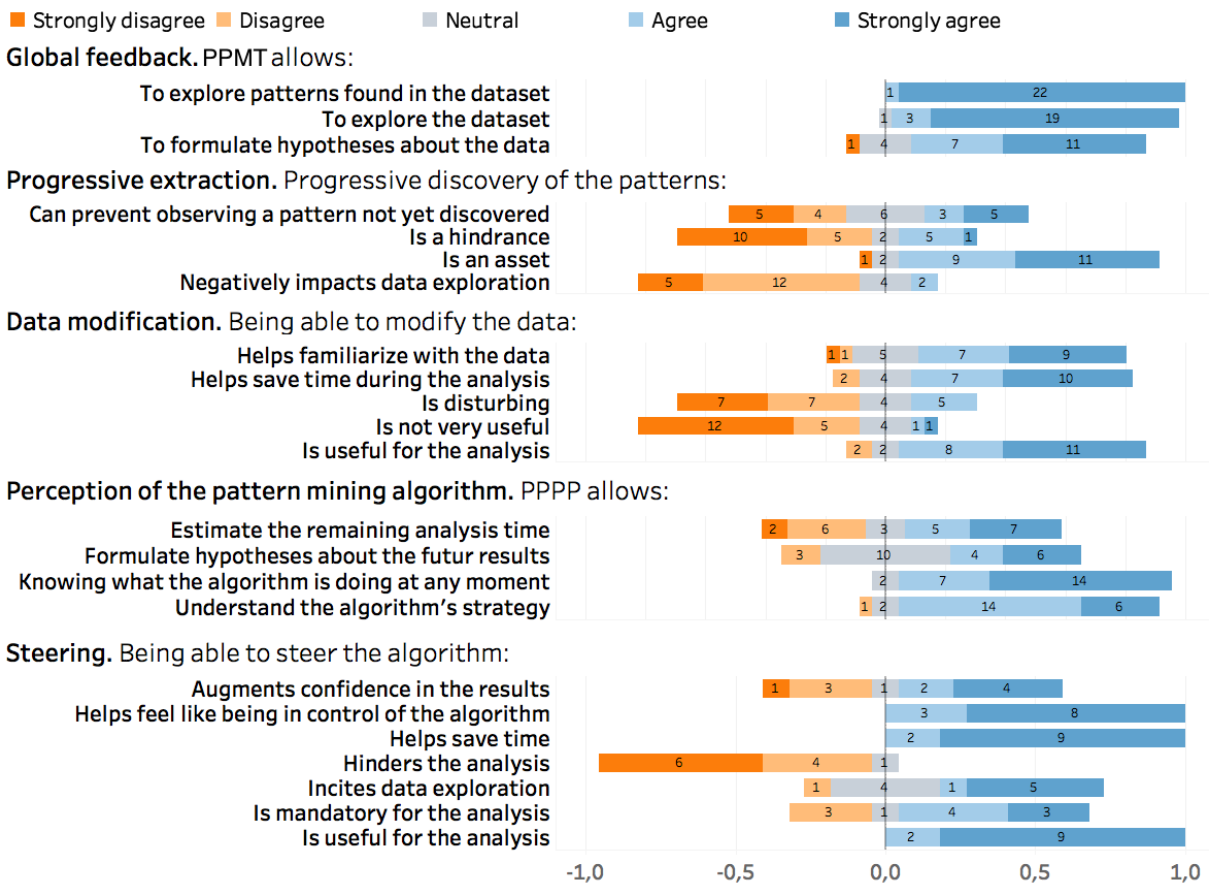


Figure 7.2: Qualitative results.

is most visible for question 1.3, where members of the *steering* group were able to quickly get the information they were looking for while the other group had to wait for the end of the computation to be sure to provide the right answer. In comparison, question 2.3 was designed to be answered by *modifying the data*, which involves no steering and can be done by both groups, and the answer times are not significantly different.

When considering the median answer time, seven questions out of eight confirm our hypothesis, but the fact that only two of these are statistically significant prevents us from drawing a clear conclusion on this matter. However, we believe that this might be a consequence of the way we designed the experiment, by deliberately choosing to target patterns that would allow both groups to correctly answer any question in a reasonable time (approximately five minutes at most). Targeting longer patterns would have added seconds to the *steering* group answer times, while simultaneously adding minutes to the members of the *non-steering* group.

3.2 Impact of interactions on correctness

As explained in the previous paragraph, we designed the questions so that both groups could be able to provide the correct answer in reasonable time. The correctness of answers is thus similar for both groups, with the notable exception of question 2.2. When looking at the data, we found out that the high number of wrong answers in the *non-steering* group was mostly caused by 6 participants that gave up on waiting for the results and answered with estimates without waiting for the end of the computation. While this would require further investigation, it might indicate that even if being able to steer the algorithm does not bring better answers, it can help avoid wrong ones in certain scenarios.

3.3 Use of available actions by the participants

When considering the feedback about the steering of the algorithm, we remarked that most members of the *steering* group used it efficiently, as they did obtain the answer they were looking for with only 1 steering per question. They consistently felt that being able to steer the algorithm was a valuable feature, with one of them saying that "*without [the ability to steer], intermediate results would be way less useful*". Another comment about the interestingness of steering was that "*for more open questions, having the ability to steer the algorithm would have encouraged [him] to explore data, because [he] knows that [he] can investigate potential insights*".

Feedback on the ability to *modify the data* is more ambiguous. When considering the answers to question 2.1, members of the *non-steering* group used it to create a single event from a subpart of the target pattern, expecting that it would extract it faster than with the original data. On the contrary, members of the *steering* group that used this action for this question were mostly curious about it, not expecting faster results than what they could achieve by steering the algorithm. This disparity was not encountered in question 2.3, for which members of both groups that did *modify the data* identified the action as the “right way to do it”. When asking participants that did not use this feature about their decision, some declared that they were not expecting any clear benefit while others said that they feared that the time needed to apply the data modification would be longer than waiting for the algorithm. This particular remark could justify the need for predictive indicators as discussed in section 4.3. Finally, some of the participants felt that it was not intuitive to modify the data, fearing that it would waste time or lead them to “not having the right results”.

4 Conclusion

The experiment we described in this chapter allowed us to investigate the impact that interactions between a human and a progressive algorithm have on the analysis process. Even

though our protocol shares some similarities with the one used by Badam et al. (2017), their interest was in comparing a progressive workflow with a more classical one (which they call "instantaneous"), focusing on the system's user interface components. In our case, we focus entirely on the progressive workflow, being interested in the underlying algorithm, its perception by the analyst and the impact of the various interactions with the analysis process we implemented. To our knowledge, this had not yet been done within the Progressive Visual Analytics paradigm.

Our results seem to confirm our hypothesis that being able to *steer the algorithm* makes the analysis process quicker, with no repercussion on the correctness of answers. On the other hand, more investigation is needed to conclude on the impact of being able to *modify the data*, since user feedback was really diverse on this matter, which may indicate that this specific interaction is less intuitive for analysts.

Maybe due to its exploratory nature, this experiment gave us a wide range of feedback but few statistically significant conclusions. This calls for further studies of this kind that we think should benefit from the following aspects. First, they should be tailored to a specific subset of the possible interactions, in order to understand how analysts perceive them and what their respective impact on a progressive analysis process is. Secondly, they should involve analysts already familiar with the data being explored (or at least with an interest in this particular dataset), especially in the case of exploratory tasks. This would ease the process of insight generation, switching the focus of the participants to "how can I interact with the algorithm to confirm or deny my hypotheses?". Thirdly, we believe that subsequent studies should be performed using different types of algorithms than sequential pattern mining, in order to compare the findings from different types of data analysis techniques.

CHAPTER 8

GENERAL CONCLUSION

Progressive Visual Analytics is still a young research field that can only be brought to its full potential if interaction with algorithms is deemed as important as progressive algorithms and incremental visualization research. From our review of the literature, we identified the following four challenges, either directly tied to Progressive Visual Analytics or specific to pattern mining in a progressive context.

Challenge PVA1

Clarify what "interaction" means in the context of Progressive Visual Analytics, as well as investigate the role of the algorithm in the process.

Challenge PVA2

Investigate the consequences of interactions between the human and the algorithm on the analysis process.

Challenge PPM1

Investigate progressive pattern mining as a tool for data exploration and not only for pattern exploration.

Challenge PPM2

Investigate the ways to make a pattern mining algorithm progressive, the relevant interactions when using such algorithm and how they should be implemented.

Through the course of this manuscript, we have presented six contributions to these challenges, of which we consider four to be major. In this last chapter, we recall these contributions and discuss some of their interesting aspects as well as their limitations, before providing some directions for future works.

1 Contributions

1.1 Major contributions

C1 – A study on interaction within a progressive process (Challenges PVA1, PVA2)

We proposed a framework that describes the different actions an analyst can perform when interacting with a progressive algorithm (C1.1). By considering a generic description of a data analysis system, we proposed a model of how these actions can impact said system (C1.2). We also conducted an evaluation of the impact that "progressiveness" can have on an algorithm's performances (C1.3).

C2 – A clarification of the process of Progressive Visual Analytics, with regards to the role of the algorithm in the process. (Challenge PVA1)

While our framework is in itself valuable to anyone interested in interactions in Progressive Visual Analytics, it provided us with a new way of looking at existing progressive systems. Doing so, we were able to propose a clarification of the meaning of the term *steering*, widely used in the Progressive Visual Analytics literature (C2.1). From this, we were able to propose an updated definition of Progressive Visual Analytics (C2.2). Combining these contributions allowed us to extend the model of the Visual Analytics workflow from (Sacha et al., 2014) into a model of the Progressive Visual Analytics process (C2.3). Our model adds progressive features — Namely the different actions we identified — to more classical Visual Analytics elements. It also highlights the importance of the algorithm during the analysis, by making it a first class component of the process.

C3 – PPMT, a progressive pattern mining system. (Challenges PPM1, PPM2)

The practical contribution of this thesis is PPMT, a Progressive Pattern Mining system (C3.1). Besides providing a usable tool to explore activity data using pattern mining techniques, developing PPMT allowed us to put our theoretical contributions on interacting with a progressive algorithm to the test. In particular, we applied our guidelines for the design of Progressive Pattern Mining algorithms (C6), implemented our interaction framework (C1.1), and supported some of the analysis tasks we identified (C5). An addition we reviewed PPMT's compliance with existing recommendations for the design of Progressive Visual Analytics systems (C3.2).

C4 – A user experiment focused on the impact of the interactions between the analyst and the algorithm on a progressive analysis process. (Challenge PVA2)

We used PPMT to conduct a user experiment in which we focused on the impact of interactions between the analyst and the algorithm on a progressive analysis process. During

this exploratory experiment participants had to use PPMT to answer several questions. We gathered a variety of information: the correctness of answers, the time needed to perform the tasks at hand, the number of times each interaction was performed, as well as feedback from the participants. While mostly not statistically significant, the results tend to suggest that not all interactions are equal from the point of view of the participants. While being able to steer the algorithm was fairly intuitive and easily understood, being able to modify the data sometimes felt unintuitive, with participants not always seeing how it could benefit the analysis process.

1.2 Secondary contributions

C5 – A data and task model for an analyst exploring temporal data with patterns, specialized from the general model of Andrienko and Andrienko (2006). (Challenge PPM2)

When considering Progressive Pattern Mining algorithm development, we wanted to provide an exhaustive list of the tasks that can be performed when analyzing temporal data using patterns. To do so, we used Andrienko and Andrienko (2006)'s task model as a basis, and specialized it for the use of pattern mining techniques.

C6 – A set of five guidelines for the design of a progressive algorithm for pattern mining in sequences. (Challenge PPM1)

Based on our task model and our experience in working with patterns, we proposed five guidelines for the design of a progressive algorithm for pattern mining in sequences, which we used when designing PPMT. It is important to note that these guidelines are to be used alongside existing recommendations for the design of Progressive Visual Analytics systems.

2 Perspectives and future work

On October 2018, Progressive Visual Analytics was the focus of the Dagstuhl Seminar 18411, entitled "Progressive Data Analysis and Visualization". Its focus was on bringing together researchers from the database, visualization and machine learning communities, in order to discuss the challenges associated with the notion of progressiveness, that each of these communities had started to address in their own terms.

Among the several directions identified in the seminar report (Fekete et al., 2019), perhaps the most high-level one is the proposal of the notion of *Progressive Data Science* (Turkay et al., 2018) as an overarching paradigm aiming at bringing the notion of progressiveness to the entire Knowledge Discovery (KDD) process. Since our work is focused on the later stages of the process, it would be interesting to explore the different ways in which it can (and can not) interact with existing approaches towards bringing progressiveness to the earlier stages, such as data selection, data cleaning and data transformation.

Going back to Progressive Visual Analytics, a common interest between our work and the seminar's is the human analyst that takes part in the process. Due to our focus on interaction with the algorithm, we proposed a categorization of the tasks that can be performed to do so, while the seminar's discussions were aiming at characterizing the various types of Progressive Visual Analytics users, identifying their roles, the tasks they want to perform, as well as their focus and potential biases (Micallef et al. (2019), see table 8.1). It would be interesting to consider our set of actions performed to interact with a progressive algorithm in light of this characterization, in order to see how the two intersect. One could for example wonder if the various elements of the characterization (roles, tasks and focus) tend to involve some actions rather than others, or if any action can be relevant for any situation. Another interesting question would be to explore which biases our actions tend to alleviate or reinforce.

Table 8.1: Characterization of Progressive Visual Analytics users. *From Micallef et al. (2019).*

Roles	R1: Observer	R2: Searcher	R3: Explorer
Tasks	T1: Ascertain, T2: Reason, T3: Understand	T4: Analyze, T5: Refine, T6: Compare	T7: Overview, T8: Further, T9: What-If
Focus	F1: Data Space, F2: Algorithmic Space		
Biases	B1: Confirmation, B2: Illusion, B3: Unnecessary waiting, B4: Misjudging uncertainties		

When considering the topic of interaction between a human and a progressive algorithm, we identify three main perspectives for future works. The first perspective is *assistance to the analyst*. As previously said, interacting with an ongoing algorithmic computation is far from a trivial process, and doing so raises some important questions. When dealing with these interactions and their consequences any assistance from the system can be a valuable help, as it allows analysts to focus on their main task, data exploration. We believe the indicators for guiding an analyst we proposed in section 4.3 are relevant to this matter. However, they are only a first step towards tackling this question, and more work is necessary to address it completely. Should one decide to explore this question, it might be interesting to prioritize the actions that were deemed less intuitive in our user experiment, namely the various forms of *modifying the data*.

The second perspective is *applying our framework to other algorithms* to further explore interaction in Progressive Visual Analytics. We designed our framework to be independent from any algorithmic technique, focusing on a few categories of general actions rather than providing a more detailed but specific categorization. Even though we reviewed existing work through the lens of our framework, we only implemented its actions for pattern mining techniques. We

believe that applying our framework to other kinds of algorithms will be useful, in order to provide a better understanding of what "interacting with an algorithm" means in a general sense.

The third perspective is the *creation of natively progressive algorithms*. As we were exploring the design of our progressive pattern mining algorithm, we modified an existing non-progressive algorithm to augment it with progressive features. This allowed us to propose a set of guidelines for the design of such algorithms, but we believe it is now necessary to conceive algorithms with progressiveness in mind from the ground up, possibly by using existing tools such as ProgressiVis (Fekete, 2015). Doing so will allow researchers to optimize for the specific use case of Progressive Visual Analytics, and might lead to additional guidelines for the design of such algorithms.

Additionally, studying the progressiveness of an algorithm's change of parameters seems an interesting aspect to study. Existing implementations of progressive systems, as well as our own, provide ways to interact in a progressive way with the underlying data or with the algorithm's strategy, but changing the parameter values always leads to restarting the algorithm. We believe that such interaction can be achieved in a progressive way, but that the parameters' key role in an algorithm's inner workings makes this a research topic of its own, that might be more easily explored when building a progressive algorithm from the ground up rather than when adapting an already existing non-progressive one.

REFERENCES

Agrawal, R., & Srikant, R., (1994), Fast Algorithms for Mining Association Rules in Large Databases. *in Proceedings of the 20th international conference on very large data bases* (pp. 487–499), VLDB '94, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., retrieved from http://dl.acm.org/citation.cfm?id=645920.672836

Agrawal, R., & Srikant, R., (1995), Mining sequential patterns. *in Data engineering, 1995. proceedings of the eleventh international conference on* (pp. 3–14), IEEE.

Aigner, W., Miksch, S., Schumann, H., & Tominski, C., (2011), *Visualization of time-oriented data*, Springer Science & Business Media.

Albers, S., (2003), Online algorithms: A survey, *Mathematical Programming*, *971*, 3–26, doi:10.1007/s10107-003-0436-0

Amar, R., & Stasko, J., (2004), Best paper: A knowledge task-based framework for design and evaluation of information visualizations. *in Ieee symposium on information visualization* (pp. 143–150), doi:10.1109/INFVIS.2004.10

Andrienko, N., & Andrienko, G., (2006), *Exploratory analysis of spatial and temporal data: A systematic approach*, Springer Science & Business Media.

Angelini, M., & Santucci, G., (2013), Modeling incremental visualizations. *in Proc. of the eurovis workshop on visual analytics (eurova'13)* (pp. 13–17).

Angelini, M., & Santucci, G., (2017), The dark side of progressive visual analytics. *in Workshop on visual analytics, information visualization and scientific visualization (wvis) in the 30th conference on graphics, patterns and images (sibgrapi'17)*.

Ayres, J., Flannick, J., Gehrke, J., & Yiu, T., (2002), Sequential pattern mining using a bitmap representation. *in Proceedings of the eighth acm sigkdd international conference on knowledge discovery and data mining* (pp. 429–435), KDD '02, doi:10.1145/775047.775109

Badam, S. K., Elmqvist, N., & Fekete, J.-D., (2017), Steering the Craft: UI Elements and Visualizations for Supporting Progressive Visual Analytics, *Computer Graphics Forum*, Eurographics Conference on Visualization (EuroVis 2017), *363*, 491–502, doi:10.1111/cgf.13205

Bertini, E., & Lalanne, D., (2010), Investigating and reflecting on the integration of automatic data analysis and visualization in knowledge discovery, *ACM SIGKDD Explorations Newsletter*, *112*, 9–18.

Borodin, A., & El-Yaniv, R., (2005), *Online computation and competitive analysis*, cambridge university press.

Brandes, U., & Pich, C., (2007), Eigensolver methods for progressive multidimensional scaling of large data, *in* M. Kaufmann & D. Wagner (Eds.), *Graph drawing* (pp. 42–53), Berlin, Heidelberg: Springer Berlin Heidelberg.

Card, S. K., Mackinlay, J. D., & Shneiderman, B., (1999), Using vision to think. *in Readings in information visualization* (pp. 579–581), Morgan Kaufmann Publishers Inc.

Casas-Garriga, G., (2003), Discovering Unbounded Episodes in Sequential Data. (pp. 83–94), doi:10.1007/978-3-540-39804-2_10

Chang, R., Ziemkiewicz, C., Green, T. M., & Ribarsky, W., (2009), Defining insight for visual analytics, *IEEE Computer Graphics and Applications*, *292*, 14–17, doi:10.1109/MCG.2009.22

Chittaro, L., Combi, C., & Trapasso, G., (2003), Data mining on temporal data: A visual approach and its clinical application to hemodialysis, *Journal of Visual Languages & Computing*, *146*, 591–620, Visual Data Mining, doi:https://doi.org/10.1016/j.jvlc.2003.06.003

Cook, K. A., & Thomas, J. J., (2005), *Illuminating the path: The research and development agenda for visual analytics*, Pacific Northwest National Laboratory (PNNL), Richland, WA (US).

Dean, T. L., & Boddy, M. S., (1988), An analysis of time-dependent planning. *in Aaai* (Vol. 88, pp. 49–54).

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P., (1996), From Data Mining to Knowledge Discovery in Databases, *AI Magazine*, *173*, 37–37, doi:10.1609/AIMAG.V17I3.1230

Fekete, J.-D., (2015), ProgressiVis: a Toolkit for Steerable Progressive Analytics and Visualization. *in 1st Workshop on Data Systems for Interactive Analysis* (p. 5), Chicago, United States, retrieved from https://hal.inria.fr/hal-01202901

Fekete, J.-D., Fisher, D., Nandi, A., & Sedlmair, M., (2019), Progressive Data Analysis and Visualization (Dagstuhl Seminar 18411), *Dagstuhl Reports*, *810*, 1–40, doi:10.4230/DAGREP.8.10.1

Fekete, J.-D., & Primet, R., (2016), Progressive Analytics: A Computation Paradigm for Exploratory Data Analysis, retrieved from https://arxiv.org/abs/1607.05162%20https://arxiv.org/pdf/1607.05162.pdf

Fisher, D., Popov, I., Drucker, S., & schraefel m.c., m., (2012), Trust me, i'm partially right: Incremental visualization lets analysts explore large datasets faster. *in Proceedings of the sigchi conference on human factors in computing systems* (pp. 1673–1682), CHI '12, doi:10.1145/2207676.2208294

Fournier-Viger, P., Chun, J., Lin, W., Kiran, R. U., Koh, Y. S., & Thomas, R., (2017), *A Survey of Sequential Pattern Mining* (tech. rep. No. 1), retrieved from http://www.philippe-

fournier-viger.com/survey%7B%5C_%7Dsequential%7B%5C_%7Dpattern%7B%5C_%7Dmining.pdf

Fournier-Viger, P., Lin, J. C.-W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., & Lam, H. T., (2016), The spmf open-source data mining library version 2, *in* B. Berendt, B. Bringmann, É. Fromont, G. Garriga, P. Miettinen, N. Tatti, & V. Tresp (Eds.), *Machine learning and knowledge discovery in databases* (pp. 36–40), Cham: Springer.

Gabadinho, A., Ritschard, G., Mueller, N. S., & Studer, M., (2011), Analyzing and visualizing state sequences in r with traminer, *Journal of Statistical Software*, *404*, 1–37.

Gan, M., & Dai, H., (2010), A study on the accuracy of frequency measures and its impact on knowledge discovery in single sequences. *in Data mining workshops (icdmw), 2010 ieee international conference on* (pp. 859–866), doi:10.1109/ICDMW.2010.83

Garofalakis, M. N., Rastogi, R., & Shim, K., (1999), Spirit: Sequential pattern mining with regular expression constraints. *in Vldb* (Vol. 99, pp. 7–10).

Glueck, M., Khan, A., & Wigdor, D. J., (2014), Dive in!: Enabling progressive loading for real-time navigation of data visualizations. *in Chi*.

Green, T. M. [T. M.], Ribarsky, W., & Fisher, B., (2008), Visual analytics for complex concepts using a human cognition model. *in 2008 ieee symposium on visual analytics science and technology* (pp. 91–98), doi:10.1109/VAST.2008.4677361

Green, T. M. [Tera Marie], Ribarsky, W., & Fisher, B., (2009), Building and applying a human cognition model for visual analytics, *Information Visualization*, *81*, 1–13, doi:10.1057/ivs.2008.28, eprint: https://doi.org/10.1057/ivs.2008.28

Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., & Hsu, M.-C., (2000), Freespan: Frequent pattern-projected sequential pattern mining. *in Proceedings of the sixth acm sigkdd international conference on knowledge discovery and data mining* (pp. 355–359), KDD '00, doi:10.1145/347090.347167

Han, J., Pei, J., & Yin, Y., (2000), Mining frequent patterns without candidate generation. (pp. 1–12), doi:10.1145/342009.335372

Hetzler, E. G. [E. G.], Crow, V. L., Payne, D. A., & Turner, A. E., (2005), Turning the bucket of text into a pipe. *in Ieee symposium on information visualization, 2005. infovis 2005.* (pp. 89–94), doi:10.1109/INFVIS.2005.1532133

Hetzler, E. G. [Elizabeth G.], & Turner, A., (2004), Analysis experiences using information visualization, *IEEE Computer Graphics and Applications*, *245*, 22–26, doi:10.1109/MCG.2004.22

Huang, K.-Y., & Chang, C.-H., (2008), Efficient mining of frequent episodes from complex sequences, *Information Systems*, *331*, 96–114, doi:10.1016/J.IS.2007.07.003

Huang, K.-Y., Chang, C.-H., & Lin, K.-Z., (2004), Prowl: An efficient frequent continuity mining algorithm on event sequences. *in Data warehousing and knowledge discovery* (pp. 351–360), Springer.

Iwanuma, K., Ishihara, R., Yo Takano, & Nabeshima, H., (2005), Extracting Frequent Subsequences from a Single Long Data Sequence: A Novel Anti-Monotonic Measure and a Simple On-Line Algorithm. *in Fifth ieee international conference on data mining (icdm'05)* (pp. 186–193), doi:10.1109/ICDM.2005.60

Joshi, M. V., Karypis, G., & Kumar, V., (1999), A universal formulation of sequential patterns.

Keim, D. A., Mansmann, F., Schneidewind, J., Thomas, J., & Ziegler, H., (2008), *Visual Analytics: Scope and Challenges*, doi:10.1007/978-3-540-71080-6_6

Keim, D. A., Mansmann, F., Schneidewind, J., & Ziegler, H., (2006), Challenges in visual data analysis. *in Proceedings of the international conference on information visualisation* (pp. 9–14), doi:10.1109/IV.2006.31

Keim, D. A., Mansmann, F., & Thomas, J., (2010), Visual analytics: How much visualization and how much analytics?, *SIGKDD Explor. Newsl. 112*, 5–8, doi:10.1145/1809400.1809403

Keim, D. A., Panse, C., Sips, M., & North, S. C., (2004), Visual data mining in large geospatial point sets, *IEEE Computer Graphics and Applications*, *245*, 36–44, doi:10.1109/MCG.2004.41

Keim, D., Andrienko, G., Fekete, J.-D., Görg, C., Kohlhammer, J., & Melançon, G., (2008), Visual Analytics: Definition, Process, and Challenges, *in* A. Kerren, J. T. Stasko, J.-D. Fekete, & C. North (Eds.), *Information Visualization* (4950, pp. 154–175), Lecture Notes in Computer Science, DOI: 10.1007/978-3-540-70956-5_7, Springer Berlin Heidelberg, retrieved November 19, 2015, from http://link.springer.com/chapter/10.1007/978-3-540-70956-5_7

Keim, D., Kohlhammer, J., Ellis, G., & Mansmann, F., (2010), Mastering the information age: Solving problems with visual analytics.

Kim, H., Choo, J., Lee, C., Lee, H., Reddy, C., & Park, H., (2017), Pive: Per-iteration visualization environment for real-time interactions with dimension reduction and clustering, retrieved from https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14381

Laxman, S., Sastry, P. S., & Unnikrishnan, K. P., (2007), A fast algorithm for finding frequent episodes in event streams. *in Proceedings of the 13th acm sigkdd international conference on knowledge discovery and data mining - kdd '07* (p. 410), doi:10.1145/1281192.1281238

Lee, A., Girgensohn, A., & Zhang, J., (2004), Browsers to support awareness and social interaction, *IEEE Computer Graphics and Applications*, *245*, 66–75, doi:10.1109/MCG.2004.24

Lin, J., Keogh, E., Lonardi, S., Lankford, J. P., & Nystrom, D. M., (2004), VizTree: a Tool for Visually Mining and Monitoring Massive Time Series Databases. *in Proceedings of the*

thirtieth international conference on very large data bases - volume 30 (p. 1380), Morgan Kaufmann Publishers, retrieved from https://dl.acm.org/citation.cfm?id=1316811

Liu, Z., & Heer, J., (2014), The effects of interactive latency on exploratory visual analysis, *20*, 2122–2131.

Luo, C., & Chung, S., (2004), A scalable algorithm for mining maximal frequent sequences using sampling. *in Tools with artificial intelligence, 2004. ictai 2004. 16th ieee international conference on* (pp. 156–165), doi:10.1109/ICTAI.2004.16

MacEachren, A., (1995), *How maps work: Representation, visualization, and design*, New York: Guilford Press.

Mackinlay, J. D., Robertson, G. G., & Card, S. K., (1991), The perspective wall: Detail and context smoothly integrated. *in Proceedings of the sigchi conference on human factors in computing systems* (pp. 173–176), ACM.

Mannila, H., & Toivonen, H., (1996), Discovering generalized episodes using minimal occurrences. *in Kdd* (Vol. 96, pp. 146–151).

Mannila, H., Toivonen, H., & Inkeri Verkamo, A., (1997), Discovery of frequent episodes in event sequences, *Data Mining and Knowledge Discovery*, *13*, 259–289, doi:10.1023/A:1009748302351

Mannila, H., Toivonen, H., & Verkamo, A. I., (1995), Discovering frequent episodes in sequences. *in Proceedings the first conference on knowledge discovery and data mining* (pp. 210–215).

Masseglia, F., Cathala, F., & Poncelet, P., (1998), The psp approach for mining sequential patterns, *in* J. Żytkow & M. Quafafou (Eds.), *Principles of data mining and knowledge discovery* (Vol. 1510, pp. 176–184), Lecture Notes in Computer Science, doi:10.1007/BFb0094818

Méger, N., & Rigotti, C., (2004), Constraint-Based Mining of Episode Rules and Optimal Window Sizes. (pp. 313–324), doi:10.1007/978-3-540-30116-5_30

Micallef, L., Schulz, H.-J., Angelini, M., Aupetit, M., Chang, R., Kohlhammer, J., ... Santucci, G., (2019), The human user in progressive visual analytics. *in Eurovis (short papers)* (pp. 19–23).

Mooney, C. H., & Roddick, J. F., (2013), Sequential pattern mining – approaches and algorithms, *ACM Comput. Surv. 452*, 19:1–19:39, doi:10.1145/2431211.2431218

Mühlbacher, T., Piringer, H., Gratzl, S., Sedlmair, M., & Streit, M., (2014), Opening the Black Box: Strategies for Increased User Involvement in Existing Algorithm Implementations, *IEEE Transactions on Visualization and Computer Graphics*, *2012*, 1643–1652, doi:10.1109/TVCG.2014.2346578

Nesbitt, K. V., & Barrass, S., (2004), Finding trading patterns in stock market data, *IEEE Computer Graphics and Applications*, *245*, 45–55, doi:10.1109/MCG.2004.28

Orlando, S., Perego, R., & Silvestri, C., (2004), A new algorithm for gap constrained sequence mining. *in Proceedings of the 2004 acm symposium on applied computing* (pp. 540–547), SAC '04, doi:10.1145/967900.968014

Pei, J., & Han, J., (2002), Constrained frequent pattern mining, *ACM SIGKDD Explorations Newsletter*, *41*, 31, doi:10.1145/568574.568580

Pei, J., Han, J., Mortazavi-asl, B., Pinto, H., Chen, Q., Dayal, U., & Hsu, M.-c., (2001), Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. (pp. 215–224).

Pezzotti, N., Lelieveldt, B. P. F., v. d. Maaten, L., Höllt, T., Eisemann, E., & Vilanova, A., (2017), Approximated and user steerable tsne for progressive visual analytics, *IEEE Transactions on Visualization and Computer Graphics*, *237*, 1739–1752, doi:10.1109/TVCG.2016.2570755

Pirolli, P., & Card, S., (2005), The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. *in Proceedings of international conference on intelligence analysis* (Vol. 5, pp. 2–4), McLean, VA, USA.

Plaisant, C., Mushlin, R., Snyder, A., Li, J., Heller, D., & Shneiderman, B., (1998), Lifelines: Using visualization to enhance navigation and analysis of patient records. *in Proceedings of the amia symposium* (p. 76), American Medical Informatics Association.

Raveneau, V., Blanchard, J., & Prié, Y., (2016), Toward an open-source tool for pattern-based progressive analytics on interaction traces, *IEEE VIS 2016 Workshop on Temporal and Sequential Event Analysis, Baltimore, MD, USA*.

Raveneau, V., Blanchard, J., & Prié, Y., (2018), Progressive sequential pattern mining: Steerable visual exploration of patterns with ppmt. *in Visualization in data science*.

Sacha, D., Stoffel, A., Stoffel, F., Kwon, B. C., Ellis, G., & Keim, D., (2014), Knowledge generation model for visual analytics, *Visualization and Computer Graphics, IEEE Transactions on*, *20*, 1604–1613, doi:10.1109/TVCG.2014.2346481

Sadallah, M., Encelle, B., Maredj, A.-E., & Prié, Y., (2015), Towards reading session-based indicators in educational reading analytics, *in* G. Conole, T. Klobučar, C. Rensing, J. Konert, & E. Lavoué (Eds.), *Design for teaching and learning in a networked world* (pp. 297–310), Cham: Springer International Publishing.

Savary, L., & Zeitouni, K., (2005), K.: Indexed bit map (ibm) for mining frequent sequences. *in In: Proc. 9th european conference on principles and practice of knowledge discovery in databases (pkdd* (pp. 659–666), Springer.

Schmidt, G. S., Chen, S., Bryden, A. N., Livingston, M. A., Rosenblum, L. J., & Osborn, B. R., (2004), Multidimensional visual representations for underwater environmental uncertainty, *IEEE Computer Graphics and Applications*, *245*, 56–65, doi:10.1109/MCG.2004.35

Schulz, H., Angelini, M., Santucci, G., & Schumann, H., (2016), An enhanced visualization process model for incremental visualization, *IEEE Transactions on Visualization and Computer Graphics*, *227*, 1830–1842, doi:10.1109/TVCG.2015.2462356

Seno, M., & Karypis, G., (2002), Slpminer: An algorithm for finding frequent sequential patterns using length-decreasing support constraint. *in Data mining, 2002. icdm 2003. proceedings. 2002 ieee international conference on* (pp. 418–425), doi:10.1109/ICDM.2002.1183937

Servan-Schreiber, S., Riondato, M., & Zgraggen, E., (2018), ProSecCo: Progressive Sequence Mining with Convergence Guarantees. *in 2018 ieee international conference on data mining (icdm)* (pp. 417–426), doi:10.1109/ICDM.2018.00057

Shneiderman, B., (1996), The eyes have it: A task by data type taxonomy for information visualizations. *in Proceedings of the 1996 ieee symposium on visual languages* (pp. 336–), VL '96, Washington, DC, USA: IEEE Computer Society, retrieved from http://dl.acm.org/citation.cfm?id=832277.834354

Srikant, R., & Agrawal, R., (1996), Mining sequential patterns: Generalizations and performance improvements. *in Proceedings of the 5th international conference on extending database technology: Advances in database technology* (pp. 3–17), EDBT '96, London, UK, UK: Springer-Verlag, retrieved from http://dl.acm.org/citation.cfm?id=645337.650382

Stolper, C., Perer, A., & Gotz, D., (2014), Progressive visual analytics: User-driven visual exploration of in-progress analytics, *IEEE Transactions on Visualization and Computer Graphics*, *2012*, 1653–1662, doi:10.1109/TVCG.2014.2346574

Teoh, S. T., Ma, K., Wu, S. F., & Jankun-Kelly, T. J., (2004), Detecting flaws and intruders with visual data analysis, *IEEE Computer Graphics and Applications*, *245*, 27–35, doi:10.1109/MCG.2004.26

Turkay, C., Pezzotti, N., Binnig, C., Strobelt, H., Hammer, B., Keim, D. A., ... Rusu, F., (2018), Progressive Data Science: Potential and Challenges, arXiv: 1812.08032, retrieved from http://arxiv.org/abs/1812.08032

van der Maaten, L., & Hinton, G., (2008), Visualizing Data using t-SNE, *Journal of Machine Learning Research*, *9Nov*, 2579–2605, retrieved from http://www.jmlr.org/papers/v9/vandermaaten08a

van Wijk, J. J., (2005), The value of visualization. *in Vis 05. ieee visualization, 2005.* (pp. 79–86), doi:10.1109/VISUAL.2005.1532781

Wang, T. D., Plaisant, C., Shneiderman, B., Spring, N., Roseman, D., Marchand, G., ... Smith, M., (2009), Temporal summaries: Supporting temporal categorical searching, aggregation and comparison, *Visualization and Computer Graphics, IEEE Transactions on*, *156*, 1049–1056.

Ward, M. O., Grinstein, G., & Keim, D., (2010), *Interactive data visualization: Foundations, techniques, and applications*, CRC Press.

Williams, M., & Munzner, T., (2004), Steerable, progressive multidimensional scaling. *in Proceedings of the ieee symposium on information visualization* (pp. 57–64), INFOVIS '04, doi:10.1109/INFOVIS.2004.60

Wong, P. C., & Thomas, J., (2004), Visual analytics, *IEEE Computer Graphics and Applications*, *245*, 20–21, doi:10.1109/MCG.2004.39

Yang, Z., & Kitsuregawa, M., (2005), Lapin-spam: An improved algorithm for mining sequential pattern. *in Data engineering workshops, 2005. 21st international conference on* (pp. 1222–1222), doi:10.1109/ICDE.2005.235

Yi, J. S., ah Kang, Y., Stasko, J. T., & Jacko, J. A., (2007), Toward a deeper understanding of the role of interaction in information visualization, *Visualization and Computer Graphics, IEEE Transactions on*, *136*, 1224–1231.

Yi, J. S., Stasko, Y. A. K. J. T., & Jacko, J. A., (2008), Understanding and characterizing insights: How do people gain insights using information visualization? *in Proceedings of the 2008 conference on beyond time and errors: Novel evaluation methods for information visualization 2008, beliv'08* (p. 1), doi:10.1145/1377966.1377971

Zaki, M., (2001), Spade: An efficient algorithm for mining frequent sequences, *Machine Learning*, *421-2*, 31–60, doi:10.1023/A:1007652502315

Zgraggen, E., Galakatos, A., Crotty, A., Fekete, J. D., & Kraska, T., (2017), How progressive visualizations affect exploratory analysis, *IEEE Transactions on Visualization and Computer Graphics*, *238*, 1977–1987, doi:10.1109/TVCG.2016.2607714

Zhang, M., Kao, B., Yip, C.-L., & Cheung, D., (2001), A gsp-based efficient algorithm for mining frequent sequences. *in Proc. of ic-ai* (pp. 497–503).

Zhang, S., Zhang, C., & Yang, Q., (2003), Data preparation for data mining, *Applied Artificial Intelligence*, *175-6*, 375–381, doi:10.1080/713827180, eprint: https://doi.org/10.1080/713827180

Zilberstein, S., (1996), Using anytime algorithms in intelligent systems, *AI magazine*, *173*, 73.

Appendices

APPENDIX A

EXISTING PATTERN MINING ALGORITHMS

This appendix provides a very brief overview of some existing pattern mining algorithms. For a detailed description, one should refer to the relevant papers.

1 Apriori-like algorithms

Table A.1: Non-exhaustive overview of existing Apriori-like Sequential Pattern mining algorithms. *Strategy can be either Breadth- (BFS) or Depth-first search (DFS). Data format can be either vertical (V) or horizontal (H). Pattern type can be either sequential patterns (Seq. patterns) or Episodes*

Algorithm	Reference	Pattern type	Strategy	Data format
AprioriAll	Agrawal and Srikant (1995)	Seq. patterns	BFS	H
AprioriSome	Agrawal and Srikant (1995)	Seq. patterns	BFS	H
DynamicSome	Agrawal and Srikant (1995)	Seq. patterns	BFS	H
GSP	Srikant and Agrawal (1996)	Seq. patterns	BFS	H
PSP	Masseglia, Cathala, and Poncelet (1998)	Seq. patterns	BFS	H
SPIRIT	Garofalakis, Rastogi, and Shim (1999)	Seq. patterns	BFS	H
MFS	M. Zhang, Kao, Yip, and Cheung (2001)	Seq. patterns	BFS	H
SPADE	Zaki (2001)	Seq. patterns	Both	V
SPAM	Ayres, Flannick, Gehrke, and Yiu (2002)	Seq. patterns	DFS	V
CCSM	Orlando, Perego, and Silvestri (2004)	Seq. patterns	BFS	V
MSPS	Luo and Chung (2004)	Seq. patterns	BFS	H
LAPIN-SPAM	Yang and Kitsuregawa (2005)	Seq. patterns	DFS	V
IBM	Savary and Zeitouni (2005)	Seq. patterns	BFS	V
WinEpi	Mannila, Toivonen, and Verkamo (1995)	Episodes	BFS	H
MinEpi	Mannila and Toivonen (1996)	Episodes	BFS	H

AprioriAll, AprioriSome and **DynamicSome** use a similar strategy to extract sequential patterns form a transaction database. The data is first sorted by customer id and time rather than sequence, a format that is then used to find frequent item types, which are called *litemset*. Each sequence is then transformed by replacing its items with the corresponding litemset. The final step is to count the pattern occurrences, where the three algorithms differ. AprioriAll counts every sequence, unlike AprioriSome and DynamicSome that can skip short sequences included in longer ones since they focus on maximal patterns.

GSP (**G**eneralised **S**equential **P**atterns) extracts frequent patterns by generating candidate patterns whose occurrences are then searched within the data. Candidate of size n are generated by combining the frequent patterns of size $n-1$. The algorithm allows one to specify constraints such as the gap between sequences or consecutive events.

PSP uses a strategy similar to that of GSP, but differs in the data structure used to store the candidates, making use of a prefix-tree rather than a hash table.

The **SPIRIT** (**S**equential **P**attern m**I**ning with **R**egular expressIon cons**T**raints) family of algorithms uses the same strategy as GSP, but uses regular expressions to constrain the set of expected patterns. Should the constraint be anti-monotone, it can be applied during the candidate generation. Otherwise, a relaxed form of the initial constraint is used, which leads to different algorithms depending on the level of relaxation (hence the term of "family of algorithms").

MFS (**M**aximal **F**requent **S**equences) jumpstarts its candidate generation by using GSP on a sample of the data, which provides an estimate of the set of frequent patterns over the complete data. This set estimate can then be used to generate candidate patterns, which removes then need to start from patterns of size 1 since this algorithm is only interested in maximal patterns.

SPADE (**S**equential **PA**ttern **D**iscovery using **E**quivalence classes) transforms the data to increase its performances, enabling the discovery of frequent patterns with at most three passes over the initial database.

SPAM (**S**equential **PA**ttern **M**ining using a bitmap representation) uses a strategy similar to GSP, but uses bitmaps (one for each event type) to speed up the candidate generation process. This benefit is however balanced by the added constraint that the aforementioned bitmaps need to be stored in memory.

CCSM (**C**ache-based **C**onstrained **S**equence **M**iner) uses a strategy similar to GSP for the extraction of frequent patterns of size 1 and 2. It then transforms the data to continue with a strategy similar to SPADE. For the second part, a cache is used to store the location of already-known frequent patterns, which speeds up the search for occurrences of patterns that show a common prefix.

MSPS (**M**aximal **S**equential **P**atterns using **S**ampling) uses a strategy similar to MFS, where they extract maximal patterns on a sample of the data in order to generate candidates. Although

the sampling technique differs, it provides a similar benefit in that all the sub-patterns can be pruned once a longer pattern is found to be frequent.

LAPIN-SPAM (**L**ast **P**osition **IN**duction **S**equential **PA**ttern **M**ining) is an optimization over SPAM due to a more efficient pruning of potential candidate patterns.

IBM (**I**ndexed **B**it **M**ap for mining frequent sequences) first reads the data and stores it in an indexed bit map. Once this transformation is complete, a candidate generation and verification process similar to GSP is conducted using this new data structure.

WinEpi mines both serial and parallel episodes, using windows and state automata to count the occurrences of candidate patterns.

MinEpi is similar to WinEpi, but focuses on minimal occurrences of episodes to compute their support.

2 Pattern growth algorithms

Table A.2: Non-exhaustive overview of existing pattern growth pattern mining algorithms. *Strategy can be either Breadth- (BFS) or Depth-first search (DFS).*

Algorithm	Reference	Pattern type	Strategy
FreeSpan	Han, Pei, Mortazavi-Asl, et al. (2000)	Sequential patterns	DFS
PrefixSpan	Pei et al. (2001)	Sequential patterns	DFS
SLPMiner	Seno and Karypis (2002)	Sequential patterns	DFS
PROWL	Huang, Chang, and Lin (2004)	Episodes	DFS

FreeSpan (**Fre**que**n**t pattern-projected **S**equential **Pa**ttern mining) uses one pass over the data to construct a set of projected databases (one for each frequent event type) that do not include events of having a non-frequent type. The remaining process of candidate generation and verification is then performed using only these projected databases.

PrefixSpan (**Prefix**-projected **S**equential **Pa**ttern mining) is based on FreeSpan. Although the strategy is similar, the projected databases distinguish between the prefix and suffix of frequent patterns to improve the candidate generation.

SLPMiner allows one to specify a minimum support constraint that varies based on the length of the candidate patterns (*i.e.* the minimum support is higher for shorter patterns compared to longer ones). As a consequence of this feature, this algorithm differs from the others in that a pattern can be frequent while one of its sub-patterns is not.

PROWL (**Pro**jected **W**indow **L**ists) extracts episodes not only within a given transaction but also across several ones.

APPENDIX B

FIRST USER EXPERIMENT

Even though evaluating PPMT's interactive interface to perform an interactive progressive analysis is our main goal, conducing such evaluation can only lead to significant results if classical data exploration tasks can be performed efficiently. As such, this experiment was designed to validate PPMT's ability to allow one to perform such classical data analysis tasks, in a progressive context where patterns are provided as intermediate results by the mining algorithm.

Participants

For this experiment, we recruited 8 participants, 6 men and 2 women, between 21 and 25 years old that were all Computer Science Master students from the university of Nantes.

Dataset

We used the *coconotes* dataset, described in subsection 6.1.3. It contains the anonymized activity data of students using the COCoNotes[1] video annotation platform. It spans a four months period from September 2016 to January 2017, totaling 201.000 events, 32 event types and 211 users. Event types range from login and navigation actions to interactions with the video player (play, pause, change volume, move to a specific timestamp...), and annotation management (create, edit, share...) events.

Protocol

The participants first signed a consent form, before being introduced to the dataset and the platform from which its data has been collected. We then presented PPMT's relevant features and user interface. They then were given a list of six tasks and had 40 minutes to answer them using PPMT. After the 40 minutes, the participants had to rate 27 affirmations on a 5-points Likert scale. These were targeting their overall feedback (3 affirmations), progressive pattern extraction (4 affirmations), data modification (5 affirmations) and their perception of the algorithm (4 affirmations). Members of the *steering* group had 7 more affirmations to rate that

[1] http://coconotes.comin-ocw.org/

were about the action of steering the algorithm. For every participant, their session concluded with a discussion where they were asked about their opinion on PPMT's usability and ability to explore activity data.

During this experiment, the progressive algorithm provided intermediate results, but the analyst were not able to perform any interaction with the algorithm.

Exploration tasks

Participants had to perform the following list of tasks:

1. Which events will never be part of a frequent pattern, and why?
2. What strategy does the mining algorithm use to extract frequent patterns?
3. What observations can you make about occurrences of the *created played* pattern, for example with regards to their distribution over time or over the users?
4. Describe the behavior of user *user011* during their first session.
5. What observations can you make about the users that have events of type *Mdp_media_seeked* within their trace?
6. What observations can you make about the patterns *paused played* and *Mdp_media_pause Mdp_media_play* and their occurrences? For example, with regards to their distribution over time or over the users? What conclusions or hypotheses can you draw from these observations?

These tasks have been designed to provide incentives for the participants to leverage the entire set of features of PPMT, except for the ones that involve interaction with the algorithm. Questions 1 and 2 target their understanding of the algorithm's behavior, questions 4 and 5 target the traces' analysis, and questions 3 and 6 target the patterns' analysis.

Results and discussions

Answers to the exploration tasks

While the algorithm's strategy is well understood (question 2: 6 correct answers, 1 bad one and 1 without answer), estimating future results based on intermediate ones and the algorithm's parameters has proven to be difficult (question 1: 1 correct answer correctly explained, 3 correct answers wrongly explained and 4 without answer). While this could indicate that PPMT does not support these tasks well enough, it also seems reasonable to assume that these difficulties

can come from the limited time participants had, in addition to them not being very familiar with the pattern mining process.

With regards to tasks that target the traces' analysis, answers to the question about a local analysis are mostly correct (question 4: 3 correct descriptions accompanied by intuitions based on observations, 3 correct descriptions and 2 irrelevant answers). However, the question targeting a specific event type presents more ambiguous answers (question 5: 4 correct observations about the temporal distribution and 4 without answer).

Finally, tasks targeting the patterns' analysis present positive results for the analysis of a single pattern (question 3), both using the time dimension (7 correct analysis of temporal distribution, although not very developed, and 1 without answer) and using the user dimension (3 correct analysis of user distribution, 3 correct descriptions and 1 without answer). Comparing several patterns (question 6) also shows positive results when it involves their temporal distributions (6 correct answers, 1 wrong answer, 1 without answer), but is less successful when it involves their distribution over the users (4 barely relevant answers and 4 without answer).

Post-manipulation evaluation

After having performed the aforementioned tasks, the overall feeling of the participants about PPMT's ability to allow one to explore a dataset and its underlying patterns while formulating hypotheses about these data is positive (over 3 questions: 15 *completely agree*, 7 *partially agree* and 2 *neutral* answers). The same observation can be made about PPMT's ability to support data exploration tasks (over 5 questions: 26 *completely agree*, 9 *partially agree*, 2 *neutral* answers and 3 *partially disagree*).

The majority of participants' feedback indicate that PPMT enables interacting with the data over any dimension (*event type*, *time* or *user*), although two participants indicate that they do not really agree with the fact that PPMT allows one to approach the data from the *user* dimension. A similar distinction between approaching the data using the *user* dimension and versus using the *time* dimension is also present when exploring the patterns. One can also note that PPMT's representation of pattern occurrences are received positively, as opposed to its ability to support pattern comparison tasks, for which most of the feedback is neutral, with a similar number of positive and negative perceptions.

With regards to the progressive pattern mining algorithm, the overall feedback is neutral, or evenly spread between positive and negative reception when considering the perception of its current state. However, the fact that patterns are provided progressively has mostly been perceived as not detrimental to the analysis, even though its usefulness in this particular version of PPMT is not highlighted (3 *partially agree*, 1 *neutral* answers, 3 *partially disagree* and 1 *disagree*), due to the absence of interactions with the algorithm.

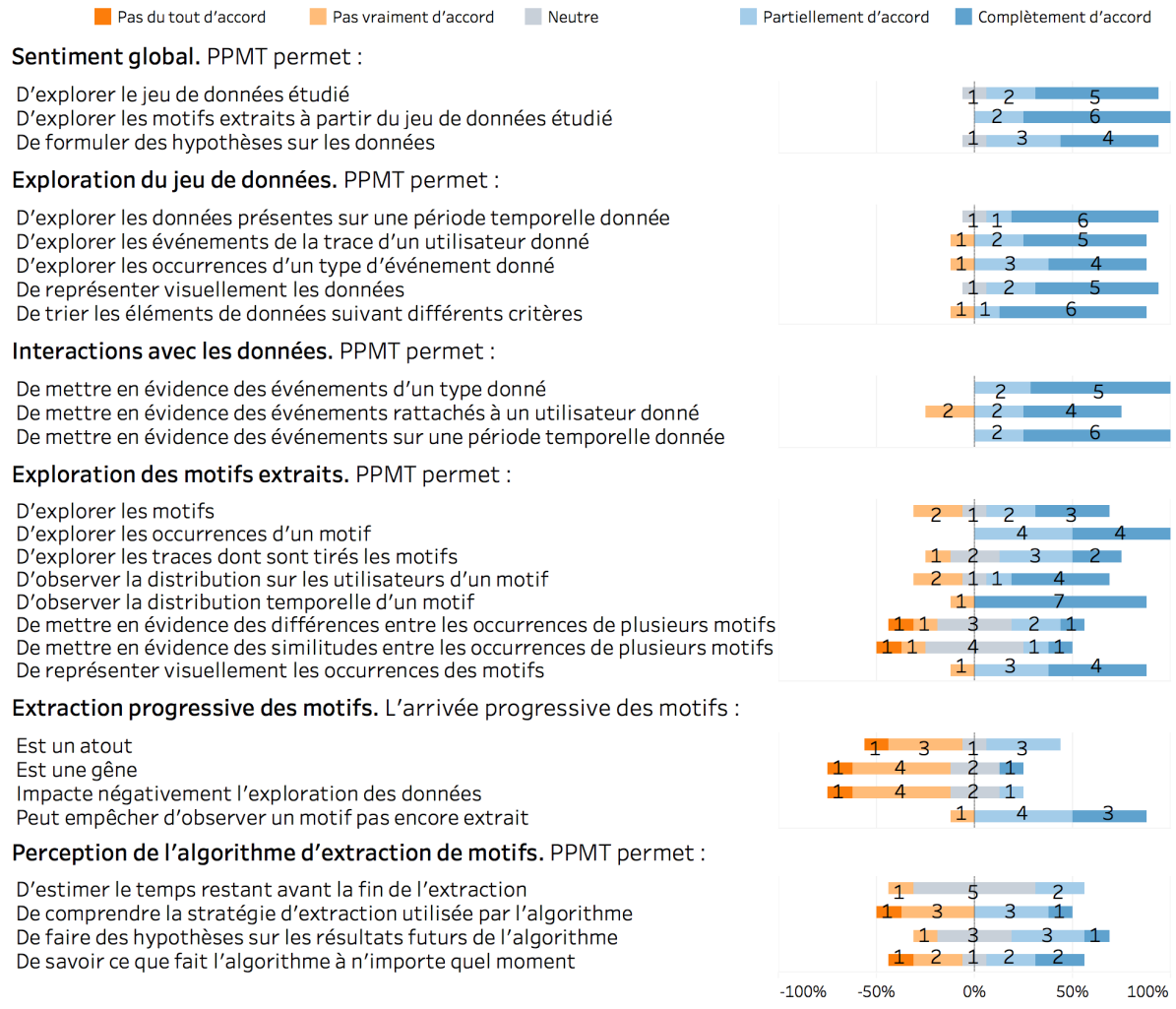


Figure B.1: *Participants' ratings of PPMT's ability to allow one to analyze a dataset in a progressive pattern mining context.*

APPENDIX C

EVOLUTION OF PPMT'S USER INTERFACE

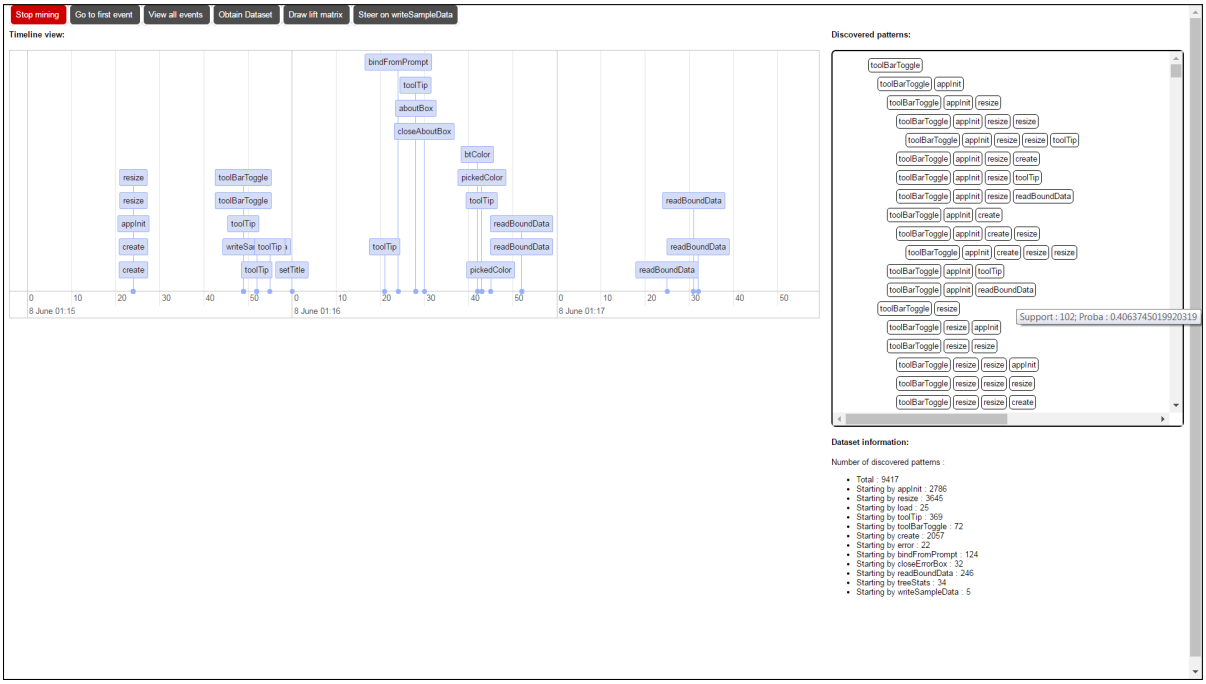


Figure C.1: PPMT in October 2016.

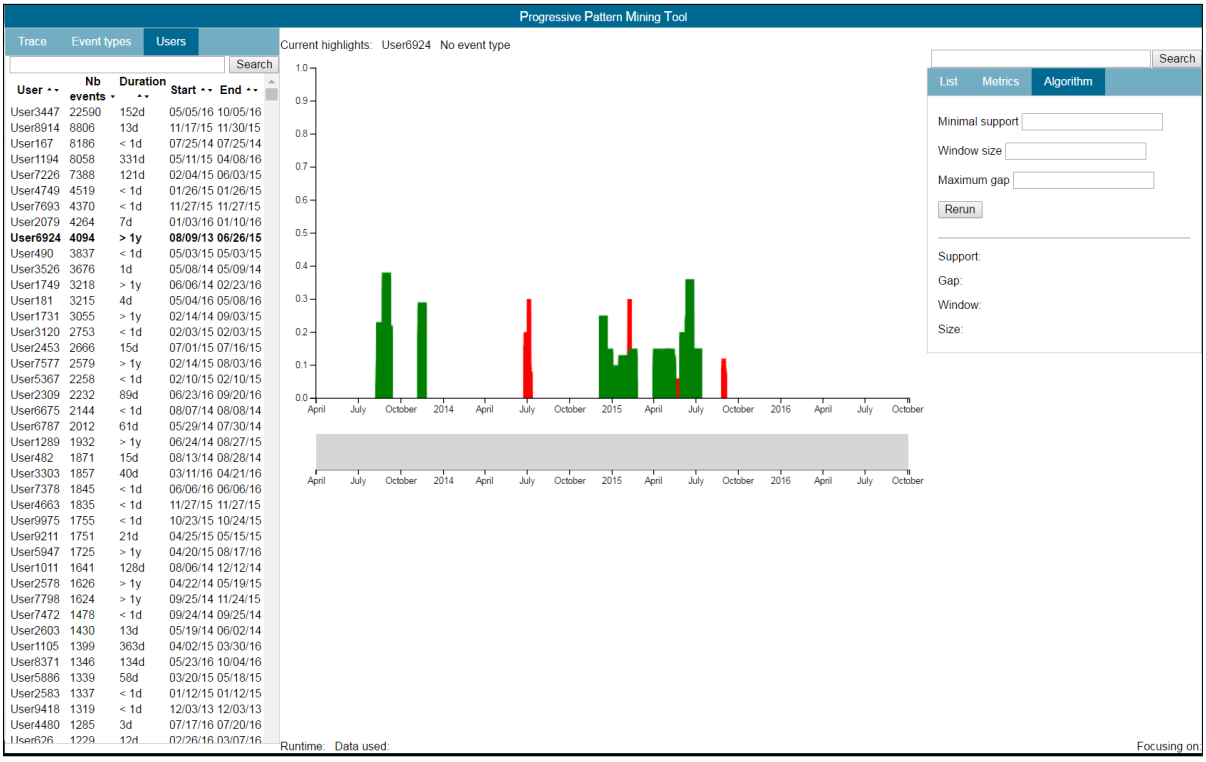


Figure C.2: PPMT in January 2017.

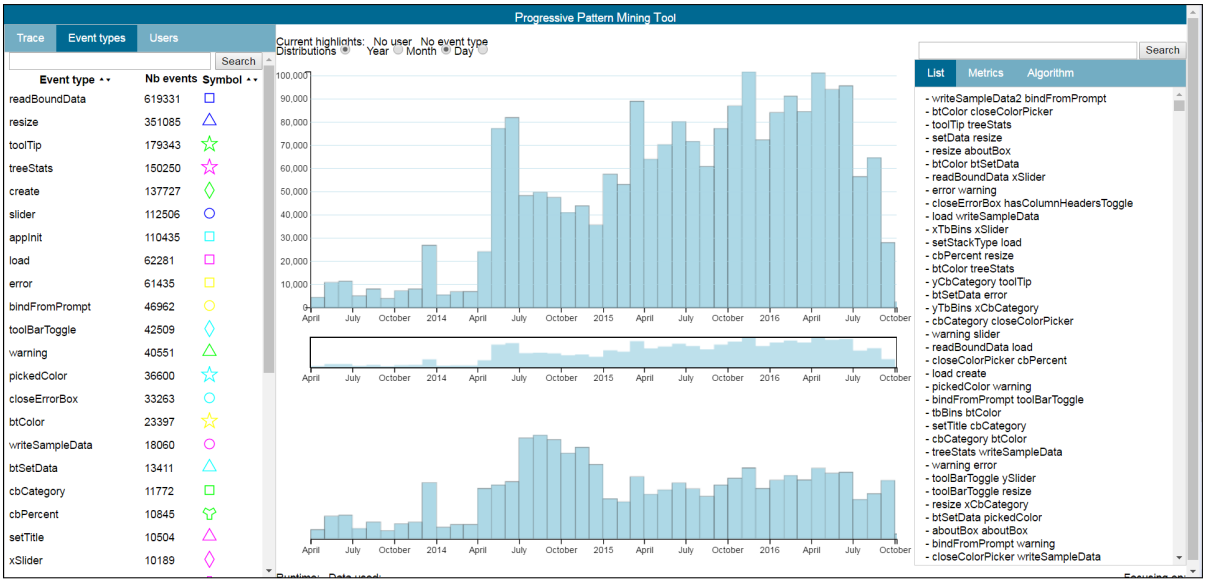


Figure C.3: PPMT in February 2017.

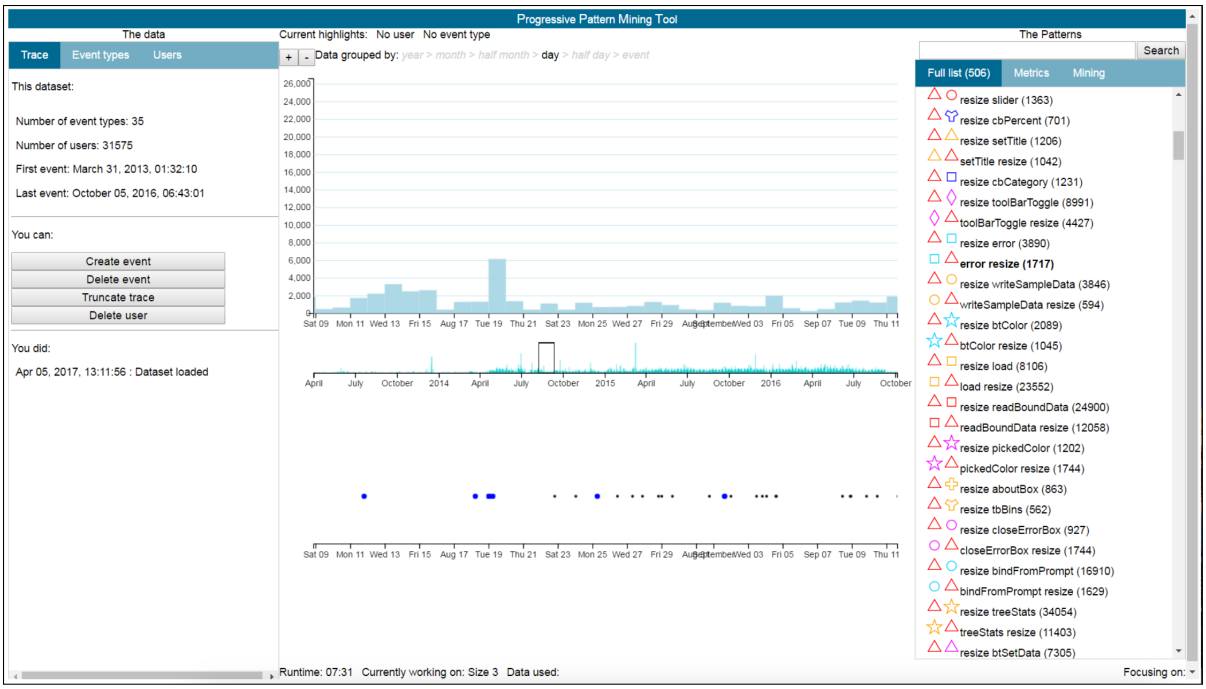


Figure C.4: PPMT in April 2017.

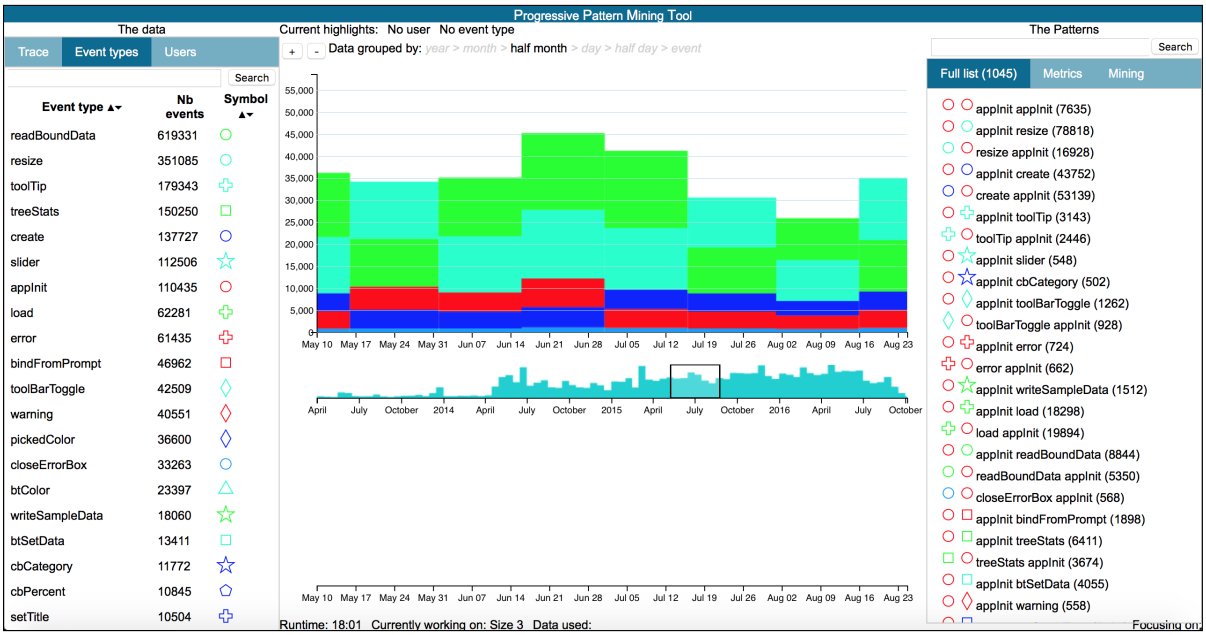


Figure C.5: PPMT in June 2017.

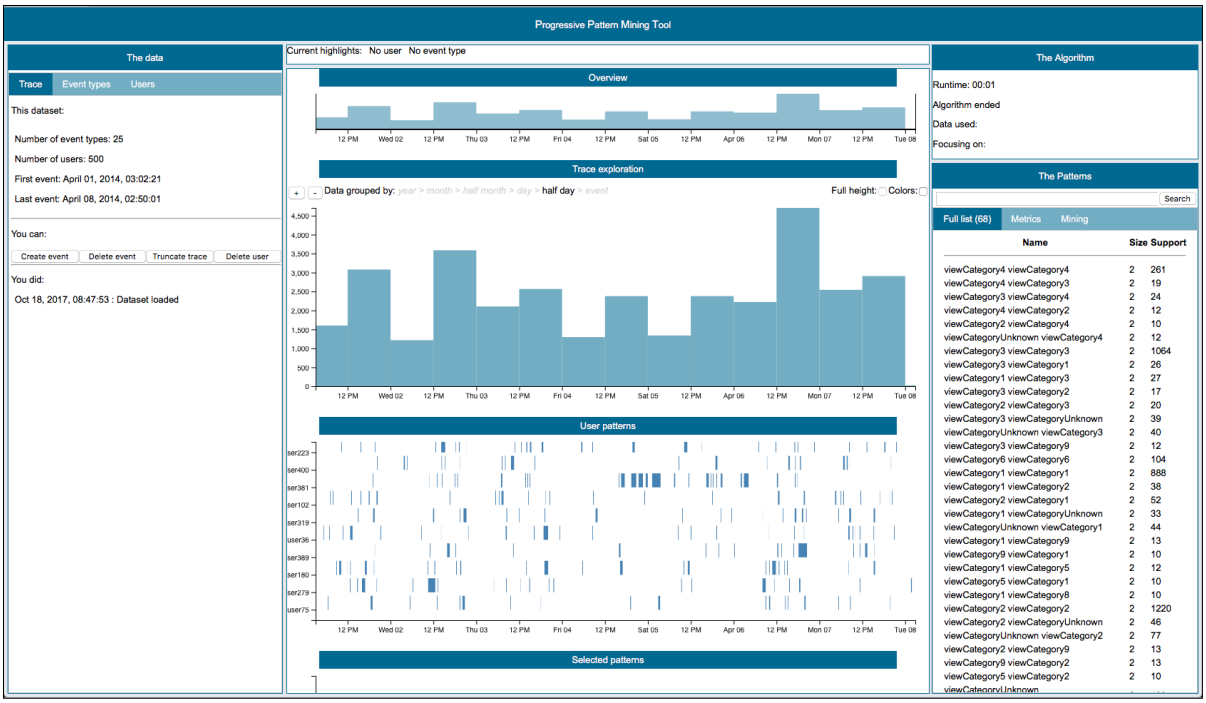


Figure C.6: PPMT in October 2017.

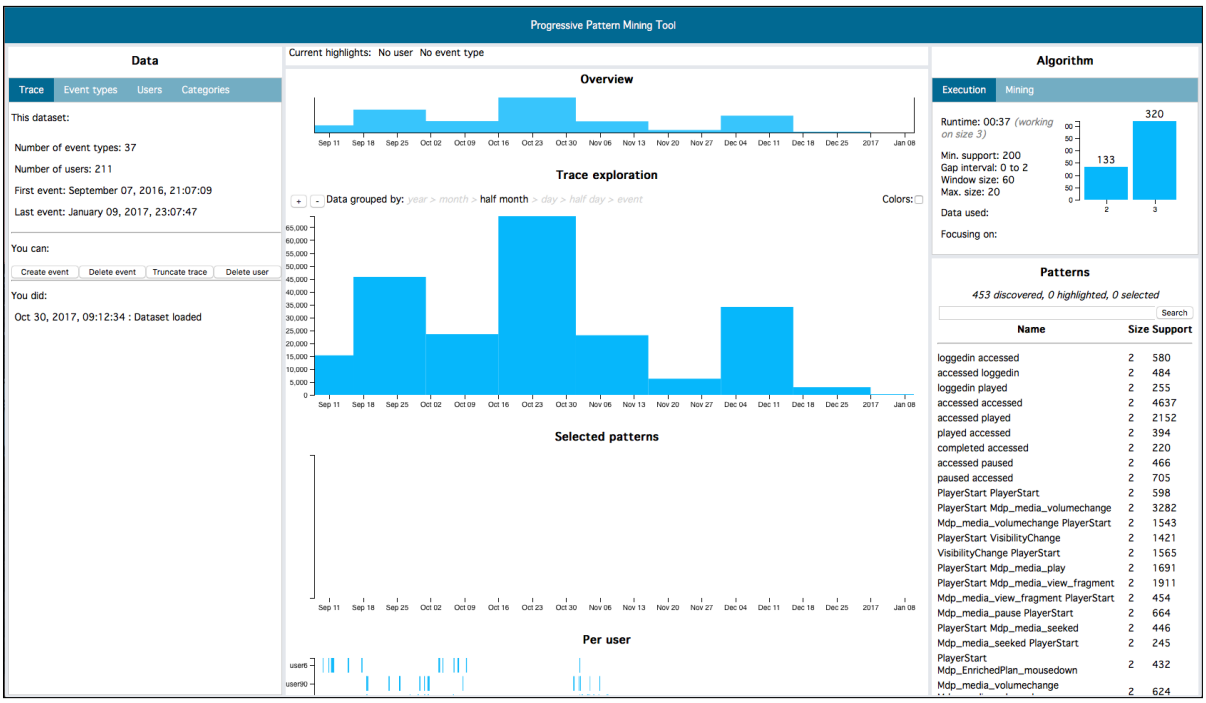


Figure C.7: PPMT in October 2017.

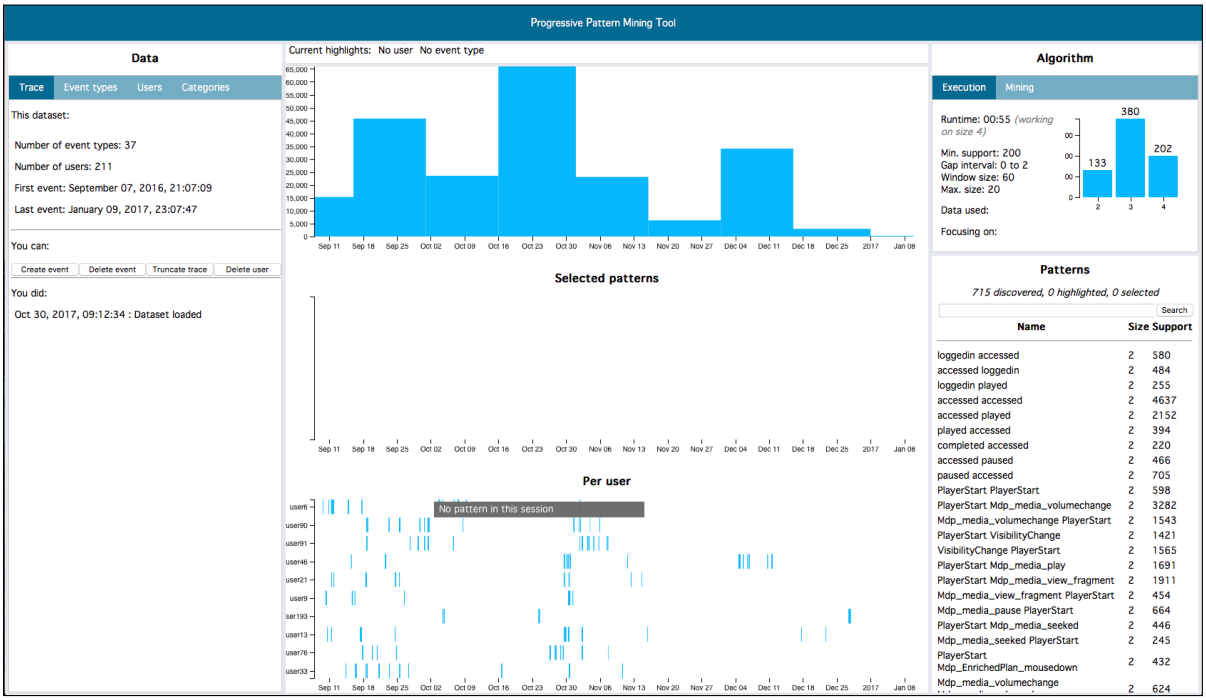


Figure C.8: PPMT in October 2017.

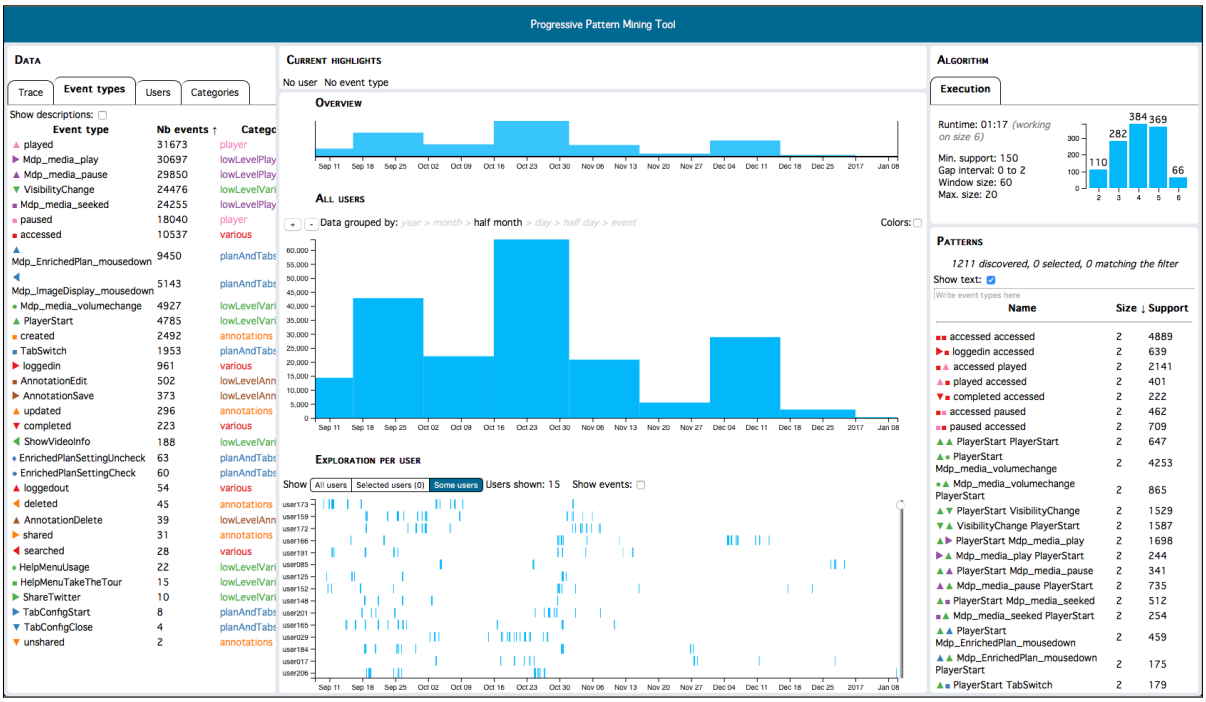


Figure C.9: PPMT in January 2018.

DOCTORAT BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE NANTES

Titre: Interaction en Analyse Visuelle Progressive. Une application à la fouille progressive de motifs séquentiels.

Mot clés : Analyse Visuelle Progressive, Fouille Progressive de Motifs, Fouille de Motifs Séquentiels, Interaction, Données Séquentielles

Resumé : Le paradigme de Progressive Visual Analytics (PVA) a été proposé en réponse aux difficultés rencontrées par les Visual Analytics lors du traitement de données massives ou de l'utilisation d'algorithmes longs, par l'usage de résultats intermédiaires et par l'interaction entre humain et algorithmes en cours d'exécution. Nous nous intéressons d'abord à la notion d'"interaction", mal définie en PVA, dans le but d'établir une vision structurée de ce qu'est l'interaction avec un algorithme en PVA. Nous nous intéressons ensuite à la conception et à l'implémentation d'un système et d'un algorithme progressif de fouille de motifs séquentiels, qui permettent d'explorer à la fois les motifs et les données sous-jacentes, en nous concentrant sur les interactions entre analyste et algorithme. Nos travaux ouvrent des perspectives concernant 1/ l'assistance de l'analyste dans ses interactions avec un algorithm dans un contexte de PVA; 2/ une exploration poussée des interactions en PVA; 3/ la création d'algorithmes nativement progressifs, ayant la progressivité et les interactions au cœur de leur conception.

Title: Interaction in Progressive Visual Analytics. An application to progressive sequential pattern mining

Keywords : Progressive Visual Analytics, Progressive Pattern Mining, Sequential Pattern Mining, Interaction, Sequential Data

Abstract : The Progressive Visual Analytics (PVA) paradigm has been proposed to alleviate difficulties of Visual Analytics when dealing with large datasets or time-consuming algorithms, by using intermediate results and interactions between the human and the running algorithm. Our work is twofold. First, by considering that the notion of "interaction" was not well defined for PVA, we focused on providing a structured vision of what interacting with an algorithm in PVA means. Second, we focused on the design and implementation of a progressive sequential pattern mining algorithm and system, allowing to explore both the patterns and the underlying data, with a focus on the analyst/algorithm interactions. The perspectives opened by our work deal with 1/ assisting analysts in their interactions with algorithm in PVA settings; 2/ further exploring interaction in PVA ; 3/ creating natively progressive algorithms, for which progressiveness and interaction are at the core of the design.