Abstract of “Visual Methods for Exploring and Forecasting Time Series” by Philipp Eichmann, Ph.D., Brown University, May 2020.

Extracting and inferring valuable knowledge from raw data requires training in programming and statistics, and more importantly, domain expertise. Yet domain experts, the people who own and are most knowledgeable about their data, often lack the technical skills to accomplish common analysis and prediction tasks themselves. In an effort to democratize data science, researchers have introduced a variety of tools to empower domain experts to explore and mine their data, and to build and interpret predictive models on their own. As a result such users are becoming increasingly sophisticated in their ability to perform data science tasks, in particular on tabular data. However, despite the fact that time series analysis and forecasting are fundamental tasks in data science, the HCI and data visualization communities have paid little attention to the problem of designing interactive tools that lower the barrier for non-expert users to perform time-oriented data science tasks.

In this work we investigate visual techniques aimed at addressing open challenges in building solutions for exploring and forecasting time series. We present solutions to four different but related problems with analyzing time series. First, we propose a sketch-based technique to search large collections of time series for best matches, and present a method to evaluate the subjective accuracy of pattern-matching algorithms against human matches. Second, we explore a method to aid the analysis of correlated events in time series and their associated meta data. Third, we present a system that facilitates black-box analysis of such time series event detection algorithms, such as anomaly detectors, to give domain experts a tool to reason about commonalities and differences among them. Fourth, we develop a system that supports non-expert users in interactively building and evaluating time series forecasting models. Finally, given the importance of fast response times when interacting with both time series as well as general tabular data, we present a benchmark for interactive data exploration, which, contrary to standard synthetic database benchmarks, features workloads that are based on the traces of real interaction logs and metrics that allow adjustment of the speed/accuracy trade-off.

Visual Methods for Exploring and Forecasting Time Series

by

Philipp Eichmann

B. Sc., University of Applied Sciences, Rapperswil, 2012

Sc. M., Brown University, 2015

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island

May 2020

This dissertation by Philipp Eichmann is accepted in its present form by the Department of Computer Science as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Date ______________ ______________________________
Prof. Andries van Dam, Advisor

Recommended to the Graduate Council

Date ______________ ______________________________
Prof. Tim Kraska, Reader
Massachusetts Institute of Technology

Date ______________ ______________________________
Prof. Stan Zdonik, Reader
Brown University

Approved by the Graduate Council

Date ______________ ______________________________
Andrew G. Campbell
Dean of the Graduate School

To my parents Felix and Sofia.

Vita

Philipp Eichmann was born and raised in Bischofszell, Switzerland. After completing a commercial apprenticeship, he studied at the International School of Tourism Management in Zurich, and later obtained a B.Sc. in Computer Sciences at the University of Applied Sciences, Rapperswil. Before moving to the U.S., Philipp worked as a software engineer for various companies in Switzerland. In 2015 he earned his Sc.M. in Computer Science at Brown University and continued his graduate studies as a Ph.D. student under the supervision of Professor Andries van Dam and his co-advisor Professor Tim Kraska. Philipp spent two Summers interning with Microsoft Research in Redmond, WA working on visual methods for time series and graph exploration. He is the recipient of the Hasler Foundation scholarship, the Andries van Dam Graduate Fellowship, as well as the Best Demo Award at ACM SIGMOD 2019.

Acknowledgements

First and foremost, I would like to thank my advisor Andy van Dam for his direction, support, and the trust he has given me throughout my time as a graduate student. Andy has been a huge inspiration and role model for me in so many ways. His encouragement to push my boundaries has made pursuing a Ph.D. with him as an advisor the most rewarding decision of my life. I will be forever grateful he offered me this opportunity five years ago. Likewise, I would like to extend my gratitude to my co-advisor Tim Kraska. Tim has been an amazing mentor and researcher who has reliably had the right plan for everything. I am very honored to have had the chance to work with him so closely. I would also like to thank Stan Zdonik, Jeff Huang, and Eli Upfal who have been extremely supportive of my work throughout my Ph.D. Furthermore, I want to thank my colleague and friend Emanuel Zgraggen. If it weren't for him, I wouldn't be writing this thesis today. Emanuel first introduced me to Andy when I started at Brown University and has been the best "academic big brother" one could ask for. I also want to thank my research director and office mate Bob Zeleznik for all of his advice, friendship, and for the long discussions about politics and stock markets. Throughout my Ph.D. I have been very fortunate to be able to work with a host of amazing collaborators: Alex Galakatos, Andrew Crotty, Bongshin Lee, Carsten Binnig, Chris White, Darren Edge, Dominik Moritz, Eric Metcalf, Franco Solleza, Giuseppe Santucci, Hyun Chang Song, Jean-Daniel Fekete, Junjay Tan, Leilani Battle, Marco Angelini, Matthew Brehmer, Nathan Evans, Rosemary Simpson, Nesime Tatbul, Steven Drucker, Tiziana Catarci, Treent Green, and special thanks to Benedetto Buratti and Zeyuan Shang whom I have been working with very closely during my time at Brown.

I would not have made it through this journey without my friends: Adrian, Angela, Ben, Cedric, Chris, Christoph, Claudio, Emanuel, Erdem, Fabio, Jakob, Janina, Martin, Monica, Nevio, Patrick, Raphael, Roger, Sam, Silas, Sina, Tak, Tonya, Yuki, and especially my family Felix, Sofia, Emily, Pascal, Sabrina, and Mailin for their ever-lasting support throughout my entire life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Thesis Statement

Extracting and inferring knowledge from raw data requires training in programming and statistics, and more importantly domain expertise. Although recent efforts to democratize data science have successfully empowered domain experts and citizen analysts to perform common data science tasks for various different data types, these solutions do not provide adequate support for time-oriented data. In this work I aim to fill this gap by inventing visual techniques to enable non-expert users in exploring and forecasting time series data.

1.2 Motivation

Data Science, the ability to take data, to process, visualize, and communicate it in order to extract value from it continues to be in high demand [139]. Data scientists must master a large spectrum of skills. Organizations across all industries and academic fields hire people with technical expertise, who are capable of building complex algorithms to organize and synthesize large amounts of information used to answer questions, inform decisions and drive strategies in their organization. They often possess a strong background in computer science and statistics with focuses on data mining and machine learning, as well as data visualization.

The specialized knowledge required by data scientists has led to problematic bottlenecks in organizations: too few people are capable of translating questions to computational procedures whose results are often crucial in informing and communicating decisions [31]. While this inadvertently leads to problems such as mis-communications between decision-makers and data scientists, long turn-around times and iterations, data science restricted to only a narrow group of people poses an even bigger challenge: the domain expertise of *non-scientists* cannot be fully leveraged.

Researchers have recently acknowledged this problem and have started to address it. Under the umbrella of the "democratization of data science", they have put forward ideas in the domains of data science systems and algorithms, educational initiatives, and social movements to give a much broader audience access to data science [30, 76]. Companies have reacted to this trend, too. For instance, Airbnb recently announced that every one of their employees "should be empowered to make data informed decisions". To accomplish this goal they made large investments predominantly in data science education [52]. Similarly, many colleges and universities now offer basic and advanced degrees in data science.

Complementary to education in data science, researchers in computer science have invested heavily in building tools to broaden the access to data analysis, data mining and machine learning. For example, for years the data visualization community has developed research prototypes tailored to the needs of non-scientists to solve complex data analysis problems (e.g., exploration of tabular data [153, 33, 41], event sequence mining [151], time series pattern mining [91], exploration of graph/network data [42], etc.). Similarly, the database community has built analytical engines capable of processing huge amounts of data at interactive speeds to power such applications [35, 127, 5, 86]. And finally, the machine learning and HCI community have developed algorithms capable of solving complex tasks with minimal human intervention, and tools enabling more people to build ML solutions [108, 124]. However, despite the fact that time series analysis and forecasting are fundamental tasks in data science, the HCI and data visualization community has only paid little attention on how to design interactive tools that lower the barrier for non-expert users to perform time-oriented data science tasks. In this work I investigate visual techniques aimed to address open challenges in building solutions for exploring and forecasting time series.

Analyzing massive amounts of data is challenging, if not impossible for users who do not possess the required technical skills. In Chapter 2 I propose by a sketch-based technique to query and sort large collections of time series, and propose a method to evaluate the subjective accuracy of time series pattern-matching algorithms.

An ubiquitous task in time series analysis is outlier detection. Humans are remarkably good at visually detecting outliers when datasets are small but have difficulties doing so with large datasets due to the loss of fidelity when scaling. A solution to this problem is to assist humans with algorithmic anomaly detectors that are able to automatically spot irregularities. Yet, as the machine learning community is making advances in anomaly detection by optimizing traditional methods and introducing new neural net-based ones, characterizing their capabilities and reason about their strengths and weaknesses is difficult without adequate visual support. In Chapter 3 I present a system that facilitates black-box analysis of time series anomaly detectors through an intuitive visual user interface that can be used to compare the results of multiple detectors and perform what-if analysis on arbitrary datasets.

While line graphs have great expressive power, they also come with certain limitations. A big shortcoming is that they are only suitable for low dimensional time series due to overplotting. Thus, visually comparing a large number of series quickly becomes intractable. A solution is to model time series as sequences of discrete events, which can be linked to an arbitrary number of related entities. The result of this transformation is a heterogeneous network, which despite its utility for exploration is notoriously hard to navigate when screen real estate is scarce. In Chapter 4 I explore a method to aid the analysis of multivariate heterogeneous networks on mobile phones. I present a mobile application, called Orchard, which combines ideas from both faceted search and interactive network exploration in a visual query language to allow users to collect facets of interest during exploratory navigation.

In order to explore time-oriented data, in many cases users must specify time as a quantitative data dimension for queries. In Chapter 5 I present a formal model that can be used to represent complex time specifications. Our model is the first step in an effort to enhance temporal user interfaces that enables discrete time specifications through a visual query interface.

Time series forecasting is a common data science procedure used for an abundance of tasks. However, developing time series forecasting models is technically challenging and requires domain expertise. In Chapter 6, I present a system that supports an end-to-end forecasting workflow for non-experts users. It exposes a set of tools that allow users to isolate and explore time series in regular tabular datasets, and enable them to model solutions for different forecasting problems.

Finally, it has been shown that interactive latency and therefore minimal database response times when exploring data is crucial for user performance. To that end, in Chapter 7 I present a benchmark for interactive data exploration, which, contrary to existing database benchmarks, features workloads that resemble the traces of real interaction logs and metrics that allow adjustment of the speed/accuracy trade-off.

1.3 Contributions and Summary

In this section I briefly summarize six out of the seven chapters that contain primary research contributions of which I have been the lead researcher and first author. With the exception of Chapter 6, all chapters are peer-reviewed and published conference papers containing only minor modifications for this thesis. References to these publications can be found below.

Chapter 2: Evaluating Subjective Accuracy in Time Series Pattern-Matching Using Human-Annotated Rankings

Finding patterns is a common task in time series analysis which has gained a lot of attention across many fields. A multitude of similarity measures have been introduced to perform pattern searches. The accuracy of such measures is often evaluated objectively using a one nearest neighbor classification (1NN) on labeled time series or through clustering. Prior work often disregards the subjective similarity of time series which can be pivotal in systems where a user specified pattern is used as input and a similarity-based ranking is expected as output (query-by-example). In this chapter, I describe how a human-annotated ranking based on real-world queries and datasets can be created using simple crowdsourcing tasks and use this ranking as ground-truth to evaluate the perceived accuracy of existing time series similarity measures. Furthermore, I show how different sampling strategies and time series representations of pen-drawn queries effect the precision of these similarity measures and provide a publicly available dataset which can be used to optimize existing and future similarity search algorithms.

Philipp Eichmann, Emanuel Zgraggen
ACM IUI '15: Proceedings of the 20th International Conference on Intelligent User Interfaces

Chapter 3: Metro-Viz: Black-Box Analysis of Time Series Anomaly Detectors

Millions of time-based data streams (a.k.a., time series) are being recorded every day in a wide-range of industrial and scientific domains, from healthcare and finance to autonomous driving. Detecting anomalous behavior in such streams has become a common analysis task for which data scientists employ complex machine learning models. Analyzing the behavior and performance of these models is a challenge on its own. While traditional accuracy metrics (e.g., precision/recall) are often used in practice to measure and compare the performance of different anomaly detectors, such statistics alone are insufficient to characterize and compare the algorithms in a systematic, human-interpretable way. In this chapter, I present Metro-Viz, a visual analysis tool to help data scientists and domain experts reason about commonalities and differences among anomaly detectors, and to identify their strengths and weaknesses.

Philipp Eichmann, Franco Solleza, Junjay Tan, Nesime Tatbul, Stan Zdonik
ACM CHI '19: Conference on Human Factors in Computing Systems

Chapter 4: Orchard: Exploring Multivariate Heterogeneous Networks on Mobile Phones

People are becoming increasingly sophisticated in their ability to navigate information spaces using search, hyperlinks, and visualization. But, mobile phones preclude the use of multiple coordinated views that have proven effective in the desktop environment (e.g., for business intelligence or visual analytics). In this work, I propose to model information as multivariate heterogeneous networks to enable greater analytic expression for a range of sensemaking tasks while suggesting a new, list-based paradigm for gestural navigation of structured information spaces on mobile phones. I also present a mobile application, called Orchard, which combines ideas from both faceted search and interactive network exploration in a visual query language to allow users to collect facets of interest during exploratory navigation. Our study showed that users could collect and combine these facets with Orchard, specifying network queries and projections that would only have been possible previously using complex data tools or custom data science.

Philipp Eichmann, Darren Edge, Nathan Evans, Bonghsin Lee, Matthew Brehmer, Chris White
EuroVis '20: Conference on Human Factors in Computing Systems

Chapter 5: Discrete Time Specifications In Temporal Queries

Analysis, exploration, and visualization of time-oriented data are ubiquitous tasks in various application domains, all of which involve the execution of temporal queries. Prior research in interactively specifying the time component for such queries has been focused on defining temporal relationships in data, i.e., querying event sequences through ordinal patterns. However, there has been much less emphasis on how to specify time as a quantitative data dimension in temporal queries. Motivated by the advent of the *Internet of Things* (IoT), I present a formal model that can be used to represent complex time specifications. Our model is the first step in an effort to enhance temporal user interfaces that enables discrete time specifications through a visual query interface.

Philipp Eichmann, Andrew Crotty, Alex Galakatos, Emanuel Zgraggen
ACM CHI '19: Conference on Human Factors in Computing Systems

Chapter 6: Time Series Forecasting for Non-Scientists

Time series forecasting is a common data science procedure used for an abundance of tasks such as inventory and staff planning, prediction of financial assets, projecting electricity consumption, as well as anomaly detection. However, developing time series forecasting models is technically challenging and requires domain expertise. In this chapter, we present a system that facilitates an end-to-end, mix-initiative forecasting workflow for non-experts users. It exposes a set of tools that allow users to identify and explore time series in regular tabular datasets, and enables them to quickly specify and find solutions for common forecasting problems. Users can iteratively add and adjust parameters such as exogenous regressors, upper and lower bounds, as well important events that may impact the forecast. Furthermore, users can explore the effect of custom interventions, and inspect the impact of different variables on any predicted data point.

Chapter 7: IDEBench: A Benchmark for Interactive Data Exploration

In recent years, many query processing techniques have been developed to better support interactive data exploration (IDE) of large structured datasets. To evaluate and compare database engines in terms of how well they support such workloads, experimenters have mostly used self-designed evaluation procedures rather than established benchmarks. In this chapter I argue that this is due to the fact that the workloads and metrics of popular analytical benchmarks such as TPC-H or TPC-DS were designed for traditional performance reporting scenarios, and do not capture distinctive IDE characteristics. Guided by the findings of several user studies I present a new benchmark called IDEBench, designed to evaluate database engines based on common IDE workflows and metrics that matter to the end-user. I demonstrate the applicability of IDEBench through a number of experiments with five different database engines, and present and discuss our findings.

Philipp Eichmann, Emanuel Zgraggen, Carsten Binnig, Tim Kraska
ACM SIGMOD '20: International Conference on Management of Data

Chapter 2

Evaluating Subjective Accuracy in Time Series Pattern-Matching Using Human-Annotated Rankings

2.1 Abstract

Finding patterns is a common task in time series analysis which has gained a lot of attention across many fields. A multitude of similarity measures have been introduced to perform pattern searches. The accuracy of such measures is often evaluated objectively using a one nearest neighbor classification (1NN) on labeled time series or through clustering. Prior work often disregards the subjective similarity of time series which can be pivotal in systems where a user specified pattern is used as input and a similarity-based ranking is expected as output (query-by-example). In this chapter, we describe how a human-annotated ranking based on real-world queries and datasets can be created using simple crowdsourcing tasks and use this ranking as ground-truth to evaluate the perceived accuracy of existing time series similarity measures. Furthermore, we show how different sampling strategies and time series representations of pen-drawn queries effect the precision of these similarity measures and provide a publicly available dataset which can be used to optimize existing and future similarity search algorithms.

2.2 Introduction

Data in the form of time series can be found in any domain. Stock prices, medical data, trajectories of objects, crime statistics, "Quantified Self" data, temperatures or sales figures can all be represented as data streams over time and even higher dimensional data are straightforward to visualize using two-dimensional line graphs. Such visualizations enable humans to interpret and compare time series data. However, visually identifying patterns within large time series or across multiple time series is a tedious and error prone task. Since traditional query languages (e.g. SQL) are often not well suited for this type of temporal pattern search, researchers have published an abundance of papers introducing pattern-matching algorithms.

Modern hardware, which supports pen and/or touch input allows users to swiftly sketch query patterns for time series data. Consider a medical researcher who wants to find patients with a specific disease who have a decreasing platelet count or a stock-trader who wants to find stocks following a certain trend. While such queries are cumbersome to specify through standard UI controls, drawing an upward or downward line on a tablet through a pen-stroke or a touch-gesture is simple and intuitive. These hand-drawn patterns, however, are not exact and therefore, the retrieved results should be displayed as similarity-based rankings, where the first item is the time series most similar to the input sketch.

Figure 2.1 shows a screenshot of TimeSketch, a pen and touch-based application that allows users to query time series through such hand-drawn patterns and displays results as similarity-based rankings. We implemented this prototype to test and experiment with different time series pattern-matching algorithms and sampling strategies. Through an informal user evaluation of TimeSketch, we realized that algorithms oftentimes produce perceivably inaccurate results, as depicted in the screenshot.

The accuracy of many existing pattern-matching algorithms have been evaluated through nearest neighbor classification (1NN) with much less attention given to how humans gauge the results of such similarity searches. This does not come as a surprise; label prediction with no further distinction within a label group can be sufficient to compare the overall performance of different techniques. However, in cases where the expected output of a search is a similarity-based ranking, a more detailed comparison is required. These rankings also need to align with human intuition and a user's perceived similarity. "Good" matches, as defined by a distance metric of an algorithm, might not coincide with the subjective similarity. This is especially important for the top k matches, whereas dissecting time series which are dissimilar to the query pattern is more difficult and arguably less important to a user. Due to the lack of datasets that contain hand-drawn patterns and human-annotated similarity rankings based on those patterns, it is currently hard to understand the type of patterns users draw

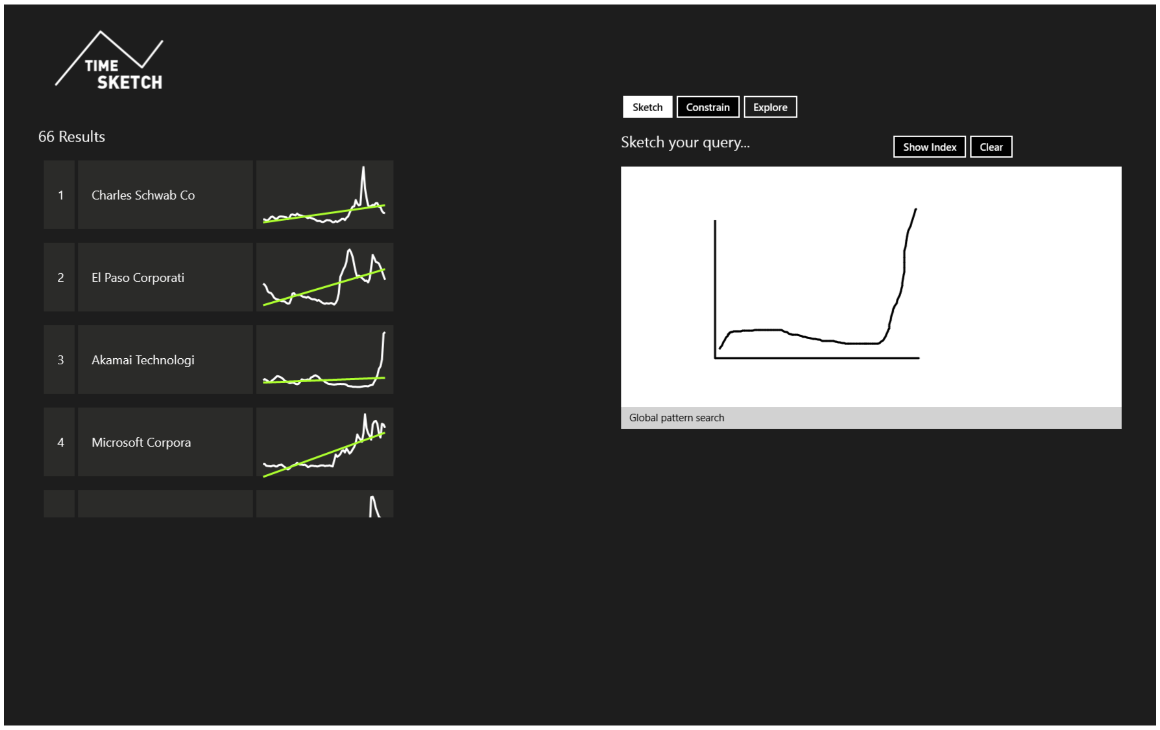


Figure 2.1: Screenshot of TimeSketch, a pen and touch-based application which allows users to query time series through pen or touch-drawn query patterns. Different combinations of sampling strategies and similarity measures (uniform linear downsampling to 80 points and SpADe similarity measure, in this case) sometimes return a perceivably incorrect ranking of the search results. Result number 3, for instance, should be clearly ranked first.

as well as to measure the subjective accuracy of existing algorithms for such use-cases.

In this chapter, we present a methodology to create human-annotated rankings of time series based on the perceived distance to a hand-drawn reference pattern. We then use these rankings to evaluate the accuracy of existing time series similarity measures, across different sampling strategies and with varying time series representations. Furthermore, we provide some analysis of our collected hand-drawn patterns and our evaluation that provide insights for possible future research efforts.

2.3 Preliminaries and Related Work

In this section, we give an overview of research efforts in the areas of query-by-example for time series data, time series representations, and similarity measures. In addition, we introduce definitions that we will refer to throughout this chapter. We refer the reader to [38] and [107] for a more comprehensive summary of time series representations and similarity measures.

2.3.1 Definitions and Notations

In the upcoming sections will use similar definitions as in [111] and [38]:

Time Series: A time series T is an ordered list $T = t_1, t_2, ..., t_n$ where each t_i is a data point in a $d-$dimensional space.

Subsequence: A subsequence $T_{i,k}$ of time series T where i is the starting index in T and $k = |T_{i,k}|$ is the length of the subsequence in number of data points.

Query Pattern: A Query Pattern Q is a subsequence used as input for similarity search whose data points correspond to either a subsequence of an existing time series or to one defined by other means (e.g. mouse/pen/touch-sketch).

2.3.2 Time Series Representations

An unmodified time series is called a *raw representation* [38] and its length depends on both, the time-span of the recorded data and the sampling rate. Typically, before a similarity search is executed, the two time series are converted to a different representation. This has the advantage of reducing (or sometimes also increasing) the number of data-points i.e. the dimensionality for reasons of space and search efficiency, noise reduction or simply scaling both sequences to an equal length. The GEMINI framework [50], for instance, proposes to find candidate matches by downsampling the time series to a dimension that works well with index structures. Once all candidate matches are found, their full-resolution time series is read from disk and compared with the query sequence using a distance metric such as the Euclidean Distance. This greatly speeds up the pattern-matching process as opposed to having to sequentially scan all time series.

Time series representations can be categorized as follows: data-adaptive, non-adaptive, model-based and data-dictated [112]. An overview and description of many representations can be found in [38, 96]. In previous publications, time series representations have been solely compared to each other in terms of indexing performance using a metric called *Tightness of lower the bound (TLB)*. This is the ratio of the estimated distance between two sequences under that representation, over the true distance between the same two sequences [72]. Interestingly, it has been found that there is very little difference between different representations in terms of indexing speed [106, 38]. Moreover, it was claimed that a representation should not be chosen based on its approximation fidelity, but rather on other features such as the visual appearance. In

, we exploit this fact to choose a suitable segmentation technique in order to project a query pattern defined by a pen stroke to a time series using different representations. Specifically, we use variants of Perceptually Important Points (PIP) [53] and Piecewise Linear Segmentation (PLS) [73] among two other approaches which we describe in the Method2.4 section.

2.3.3 Similarity Measures

Similarity measures are used to compute a distance between two time series. [38] distinguishes between *lock-step*, *elastic*, *threshold-based* and *pattern-based* measures.

The simplest similarity measures are lock-step measures, which compute the distance between two time series in a one-to-one fashion, where each data point in one time series is only compared to its direct counterpart in the other. Typically, the *Euclidean Distance* is used because it penalizes larger distances more, but other Lp-norms such as the Manhattan Distance fall into this category. One of the biggest shortcomings of lock-step measures is that they are sensitive to local time shifting [28], such that even two time series that are perceived to be similar could be separated by a large distance. This makes the Euclidean Distance unsuitable for our application, as hand-drawn queries should be invariant to local time shifting and scaling. Nonetheless, we use this measure in our experiments in order to draw a comparison to other methods.

Elastic measures support shifts and scaling in time-dimension. Most notably Dynamic Time Warping (DTW), which was recently claimed to be the best measure in terms of speed and accuracy [111]. It tries to find the optimal alignment between two time series by allowing for local warping within a certain window. DTW uses the Euclidean Distance as distance measure between two points. Thus, the Euclidean Distance is essentially a special case of DTW, where the warping window size is equal to 1. Other edit distance based measures include Longest Common SubSequence (LCSS) [10, 142], Edit Sequence on Real Sequence (EDR) [141], Swale [100], Edit Distance with Real Penalty (ERP) [27]. Elastic measures are especially important for human-drawn queries as the query patterns are never exact. Due to its popularity and recent research efforts, we use DTW in our experiments.

SpADe (Spatial Assembly Distance) is a pattern-based measure that promises to handle time/amplitude-shifting and scaling and noise [28]. It works by computing features over short subsequences of two time series, called *local patterns*, finding similar patterns called *local pattern matches* in both time series and eventually calculating the shortest distance between all found matches from the start of the time series to the end. As we consider time and amplitude invariance important characteristics of similarity measures for human-drawn queries, we also employ SpADe in our experiments.

2.3.4 Pattern-Matching Invariance

Some similarity measures reduce variance in pattern-matching by making the measure flexible to time-shifting/scaling and amplitude-shifting/scaling, which is important in applications where the query pattern is defined by hand. Based on [13], we give a brief overview on how these and other types of invariance can be implemented.

Even though many subsequences in a dataset may look similar to a query pattern, they may be defined at different amplitude ranges or they are located at different offsets. *Amplitude invariance* can be achieved by normalizing the time series which are compared to each other. It is known that a *z-score normalization* follows the original shape of the datapoints more closely than a *min/max normalization* [96].
Elastic similarity measures handle *local scaling* in time-dimension. To fix *global scaling*, on the other hand, we have to either test all possible window sizes or define a time range for a query pattern to eliminate the problem entirely. To tackle *phase invariance*, we have to test all possible alignments in time. Moreover, [13] suggests that *occlusion invariance*, the case where we want to ignore one or multiple subsequences in a pattern, can be achieve by simply ignoring the distance on those time intervals.

2.3.5 Defining Query Patterns

The idea of specifying time series queries in a query-by-example fashion has been around for over a decade. Typically, the goal of a similarity search is to perform a k-NN-search i.e. to return the k nearest neighbors given an input [144]. Similar to our prototype TimeSketch, QuerySketch lets users define arbitrary queries by sketching onto an empty graph, where the time scale can be adjusted by using a zoom function [145]. The Euclidean Distance between the query pattern and the time series in the dataset is used as similarity measure and each result is labeled with the distance to the query.

TimeSearcher uses the concept of timeboxes, rectangular regions that can be drawn over a time series, and uses the containing subsequences as query patterns [63]. Since timeboxes define value-ranges rather than fix-valued data points, the similarity of the specified query and the time series in a dataset is determined by an orthogonal range tree algorithm [37]. Counters are used as degree of similarity by measuring how many data points lie in the specified amplitude and time range. Even though the bulk of this chapter focuses on query patterns rather then range-queries, TimeSearcher provides inspiration for further research (see Discussion and Future Work).

[58] presents TimeSearcher: Shape Search Edition (SSE), an extension of TimeSearcher where query patterns can be defined by shapes such as spikes, sinks, rises, drops, lines, plateaus, valleys and gaps. The authors describe attributes of these shapes by which they can be identified, compared and ranked. The similarity measures we used in our evaluation are purely mathematical. However, as we note in the Discussion and Future Work section, a combination of high and low level similarity

measures is a promising research direction.

2.3.6 Methods to evaluate time series similarity

Keogh et al. propose two ways to measure the true similarity of two time series [72], subjective evaluation and objective evaluation. The latter can be carried out by using a 1NN classification to predict a label for a time series from its nearest neighbor in a training dataset, where the classifier's distance measure is the similarity measure in question. A major drawback of this approach when evaluating similarity is that both, the training and test data must be labeled. Even though there are publicly available labeled datasets, new and unlabeled datasets cannot be tested. Furthermore, this method only measures how often the prediction of a label failed or succeeded but fails to provide a distance of how well it actually matched a query pattern, which is inevitable to create rankings. Also, it is often unclear who created the labels and based on which decisions.

Subjective evaluation relies on human judgment. A trivial way to evaluate the perceived accuracy of time series is to simply visualize matching results of different similarity measures and let people assess the quality of the algorithm. Another commonly used approach is to create a dendrogram of multiple time series using the single linkage method and let humans determine, if similar time series are reasonably linked. The authors of [74] and [73] note that the "correct" distance measure depends on user preference and the data domain. They describe an approach to adjust the similarity metric by using feedback from users. More specifically, they propose the time series to be represented as piecewise linear segments (PLS) and a weight vector to describe the relative importance of each segment. The weights are dynamically updated based on the user's relevance feedback, which allows for a custom weighting scheme for each user and domain.

2.4 Method

While existing time series datasets, such as [75], are suitable to benchmark classification and clustering accuracy, there are currently no datasets that provide a distance-based ranking or other means to fine-tune a similarity search algorithm which returns an ordered list of results, based on a given input sequence. The subjective accuracy of a similarity search algorithms depends on the data-domain and user preference [74, 73]. Therefore, when creating a system similar to TimeSketch, it is important to evaluate the perceived accuracy on a case-by-case basis. In this section, we show how human-annotated rankings can be created for any type of time series data based on [64] and using simple crowdsourcing tasks. Using the help of the crowd is especially advantagous in the absence of expert annotators or when annontating large amounts of time series data becomes unfeasible.

In Computer Vision, for instance, the human-perceived accuracy of image classification systems can be determined by testing the classifier against a human-annotated set of images (ground-truth). Borrowing this idea, we describe how a ground-truth for selected subsequences in a given dataset can be created using a crowdsourcing platform and use this ground-truth to compute a similarity-based ranking over all subsequences.

2.4.1 Creating Human-Annotated Rankings

In order to create a human-annotated ranking for a specific query, we first generate all possible subsequences in a dataset that match the length of a query pattern. We then create a set of all unordered pairs of subsequences and use a pairwise comparison between each of the subsequences to build a global ranking, according to their similarity to the query pattern.

More formally, for a given query pattern q and a dataset d, we generate a set S_q comprising all subsequences in d of the same length as q, where S_q is the set of all subsequences that will be part of the similarity ranking. Let U_{Sq} be the set of all unordered pairs in S_q. For each pair of subsequences $\langle a, b\rangle$ in U_{Sq}, we create a task on a crowdsourcing platform, where the workers' task is to decide, which of the two subsequences resembles the query pattern more (binary classification). We let the same task be answered by n distinct workers, in our case $n = 9$. The degree of agreement among all workers i.e. the probability of one subsequence being more similar to the query pattern than the other, can then be used to compute a global score of similarity. To obtain such a score for any subsequence $a \in S_q$, we can sum up the probabilities of a being more similar to q then all other subsequences in S_q.

$$\mathit{score(a)} = \sum_{b \in S_q} p(\langle a, b\rangle) \tag{2.1}$$

Where p in Equation 2.1 is a function that returns the probability of a being more similar to q than to the other subsequence b of a pair, according to the number of received votes in the binary classification task. To create the ranking, S_q can be sorted by the score of its elements.

2.4.2 Comparing Rankings

Computing a ranking from the distances provided by a similarity measure is as simple as ordering all subsequences based on their distance in ascending order. Once all rankings for all similarity measures and query pattern representations have been established for a particular query, they can be compared to the human-annotated ranking to see how well both rankings align. Given a human-annotated ranking R_H and any similarity-measure based ranking R_S, we compute the Spearman's

ID	Dataset	Length
1	Daily reported crimes in Chicago	4,748
2	Microsoft Stock Values	5,036
3	NBA Player Scores	930
4	City Temperatures	2,920

Table 2.1: List of all datasets used in survey

rank correlation coefficient

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2-1)} \tag{2.2}$$

where d_i is the difference of the ith ranking in R_S and R_H respectively and n is the total number of ranks. A Spearman coefficient of 1 or -1 indicates that the two ranking correlation perfectly (positively or negatively) which means the produced ranking is identical to the human created one. We use this correlation coefficient as a measure of how well two rankings align because the distance between two ranks are squared. Thus small differences in ranks (i.e., transpositions) are less penalized than larger ones.

2.5 Evaluation and Analysis

Using the method described in the previous section, we created 8 different human-annotated rankings through Amazon Mechanical Turk, using 8 pen-drawn query patterns over 4 different datasets and compared the accuracy of 3 different similarity measures in conjunction with 4 distinct sampling strategies.

2.5.1 Datasets

To avoid bias in queries towards a certain domain, we created 4 different datasets with varying lengths and context, as shown in Table 2.1. Dataset 1 contains daily reported crime incidents in the city of Chicago from 2001 to 2013. In the second dataset, we listed the Microsoft stock closing values of all trading days over a period of 10 years (1994-2013). Dataset 3 contains the number of points former NBA player Michael Jordan scored per game for all season games he had played. Finally, dataset 4 comprises maximum daily temperatures of 8 different cities in the United States.

2.5.2 Collecting Query Patterns

Instead of merely using a subsequence of a time series as query pattern, we asked 6 users to draw queries using a pen/touch-based application, which records all strokes and pen-related meta-data. Before recording a query pattern, we instructed the participants on how to use the tool and which

Dataset	Question
1	Crime rate which fluctuates periodically for a couple of months
1	A period of a year, where the crime rate increased steadily
2	Find the 2008 stock market crash
2	A situation where the stock value dropped and remained low for a while
3	Find a player who started off really well at the beginning of his career, but had decreasing scores over his entire career
3	Find a period where a player consistently scored more than 20 points per game
4	Find a place in the United States where it is warmer in Winter than in Summer
4	Find places with hot months in 2013

Table 2.2: Example questions users tried to answer with a pen/touch-based interface.

dataset their queries should target. For instance, a user was requested to draw an arbitrary number of queries for a data-set comprising the maximum daily temperature of 8 different cities in the United States. The user was then asked to define a pattern length in terms of number of data points (e.g. 30 days) and optionally instructed the application to draw guide lines, which some users found helpful to define their queries more precisely. Also, the participants were able to choose whether or not they wish to ignore the given minimum and maximum amplitude in their query and to change the size of the drawing area.

We recorded a total number of 112 pen-drawn queries (shown in Figure 2.8 at the end of

) over the 4 different datasets. The participants of our survey were all graduates students with no prior experience in time series analysis. For each pattern, we recorded what question the user wants to answer with her/his query, some of which can be found in Table 2.2.

2.5.3 Creating a Human-Annotated Ranking

After collecting sketches, we proceeded to creating human-annotated rankings based on our method outlined earlier in the paper. Since the number of required pairwise comparisons becomes very large for a large number of subsequences, we followed the following procedure: from the set of all recorded query patterns, we hand-picked eight (two for each dataset) to assure coverage of the most distinct types of user-drawn queries. Using a window of the size of the pattern length, we then generated all possible subsequences with step size 1. Next, we assigned each retrieved subsequence to one of 25 clusters using k-means clustering and randomly picked 4 from each cluster. This left us with 100 subsequences to be compared with the query pattern. Since this would still require us to do 44,550 pairwise comparisons ($9 \cdot \binom{100}{2}$), we used a two-task crowdsourcing strategy. In a first task, we displayed the query pattern and a plot of one of the 100 subsequences. The workers' task was to

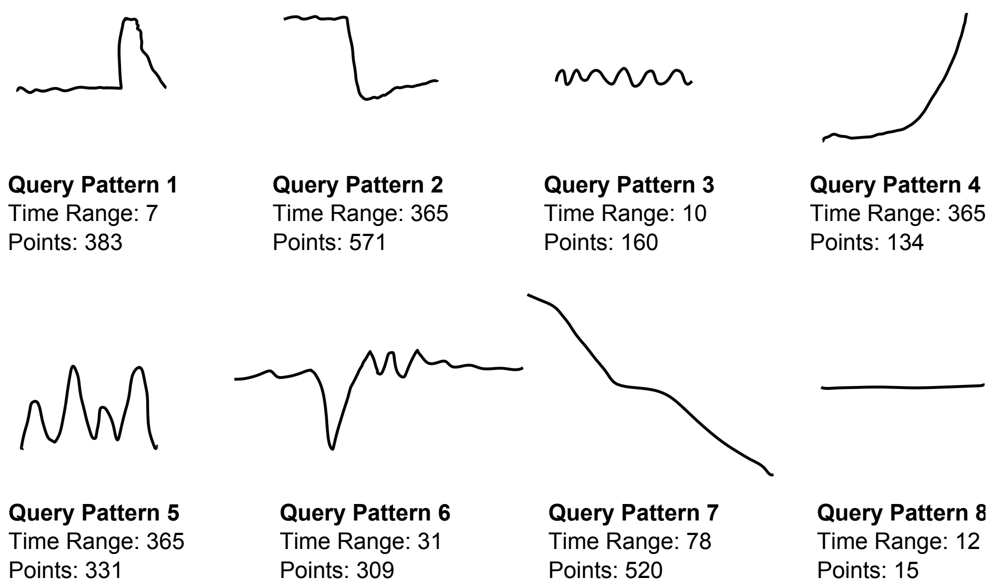


Figure 2.2: Eight hand-picked query patterns used to create human-annotated rankings.

judge the similarity of two graphs by selecting a value from 1 to 10. We let each comparison between the query pattern and every subsequence be answered 5 times and then selected the 15 subsequences that resemble the query pattern most, according to the workers' median answer. Using these steps, we were able to reduce the number of required pairwise comparisons for each query pattern to 105. In a second crowdsourcing task, we employed our method of creating a human-annotated ranking. We created a set of all unordered pairs of the 15 subsequences and built a binary classification task where the query pattern is displayed along with the two subsequences of a pair. The workers' task was then to select the one subsequence of a pair which resembled the query pattern more. We let each task be answered 9 times and then computed the score of each subsequence using Equation 2.1.

2.5.4 Evaluating Similarity Search Accuracy

In order to create a similarity measure based ranking for the query pattern, we first need to transform the sketch into an evenly spaced time series. This is a delicate task, as the strokes provided by pen/touch hardware consist of a varying number of non-uniformly distributed 2D-points, whereas the time series in our datasets are evenly spaced. To conduct this transformation, we first order all points by their time stamp. We then remove all points that are not monotonically spaced in time. Next, we sample the points to match the length of the query pattern with one of the following algorithms:

Uniform Linear: A uniform linear sampling stores the values of all points in an array and

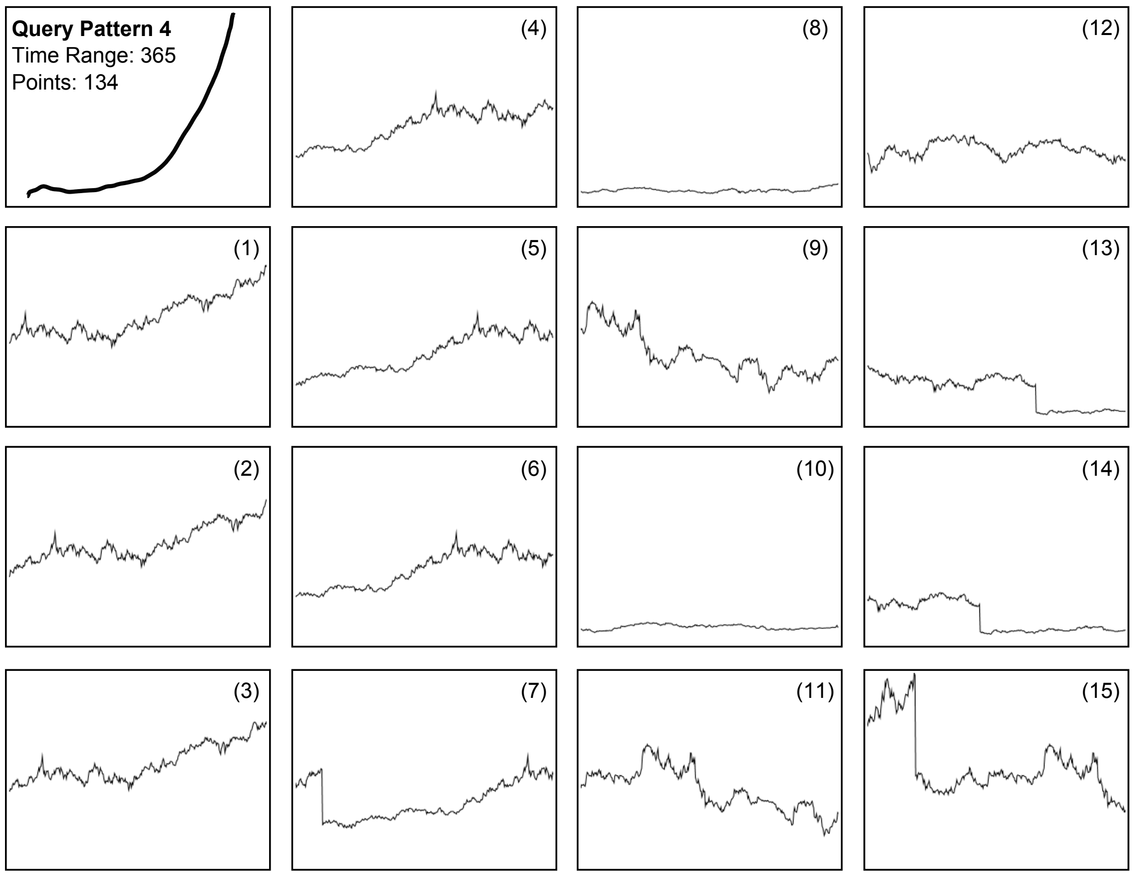


Figure 2.3: Example of a human-annotated ranking created through crowd-sourcing tasks.

performs up/down-sampling using a linear smoothing kernel. It is naive in the sense that it distorts the shape of the originally drawn pattern by directly converting an unevenly spaced time series to an evenly spaced one.

Geometric Linear: A geometric linear sampling maps each point of the resulting time series to a location between two 2D-points of the input stroke. The value at that location is computed using linear interpolation.

PLS: PLS sampling performs a piecewise linear segmentation on the input stroke as described in [73]. The value is then retrieved in the same manner as in the Geometric Linear approach.

PIP: PIP sampling applies the PIP-algorithm on the input stroke as described in [53]. As in PLS, the value is then retrieved in the same manner as in the Geometric Linear approach.

Once the query pattern is converted to a time series, we can z-normalize both the pattern and subsequences and use a similarity measure to compute the distance between them. We employ three different algorithms in our experiments: Euclidean Distance, DTW and SpADe. To avoid

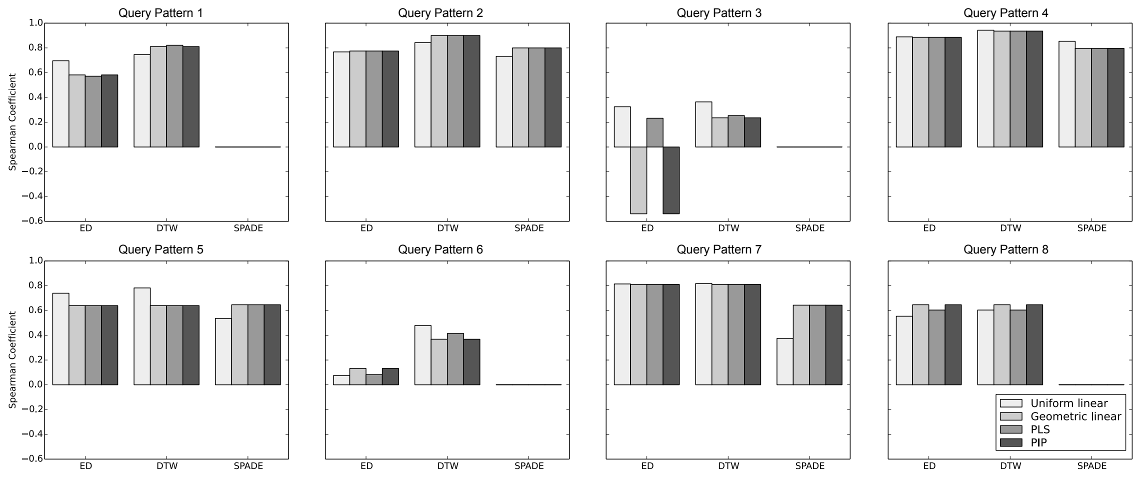


Figure 2.4: Spearman correlation coefficients of different algorith ms and sampling methods when compared to human-annotated rankings (1 indicating perfect positive correlation, −1 perfect negative correlation).

implementation bias, we use the original implementation from [111, 28] respectively. The results of our comparisons can be seen in Figure 2.4. DTW produces rankings that are closest to the human-annotated rankings. For short query patterns (1, 3, 6 in Figure 2.2), SpADe was unable to output meaningful distances. We verified this fact with the author, who confirmed that SpADe does not work well for very short time series. For larger time series it yields competitive results (query patterns 2, 4 and 5). The Euclidean Distance expectedly performed worse than DTW or equally well in some cases. Interestingly, it performed very badly in combination with Geometric Linear and PIP sampling in some cases (query pattern 3 and 6). We found that, especially for short time range queries, Geometric Linear and PIP sampling significantly distort the original sketch, which can lead to a comparitively large Euclidean Distance. DTW, however, manages to remedy a poorly sampled sketch to a certain extend. The naive uniform linear sampling did surprisingly well. One plausible reason for its good performance is that even though there are changes in velocity across the drawing area, as can be seen in Figure 2.5, the smoothing kernel makes up for the non-uniformly distributed points.

Overall, different sampling strategies do not seem to impact the accuracy of algorithms drastically. In fact, Geometric Linear, PLS and PIP actually often yield roughly the same results due to their identical interpolation method and only differ in the way the points are chosen during down-sampling.

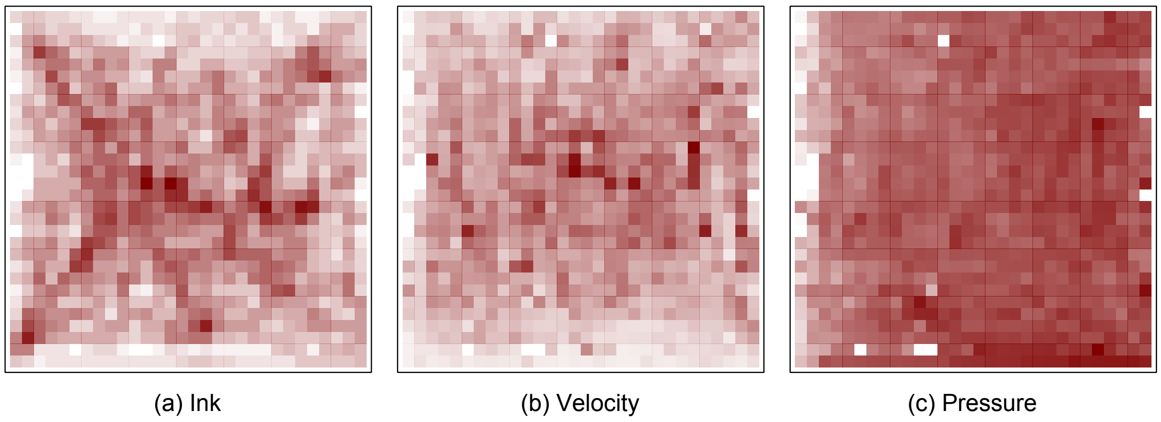


Figure 2.5: Heat-maps visualizing different properties across all query patterns over the normalized sketching area. Dark-red indicates a higher value.

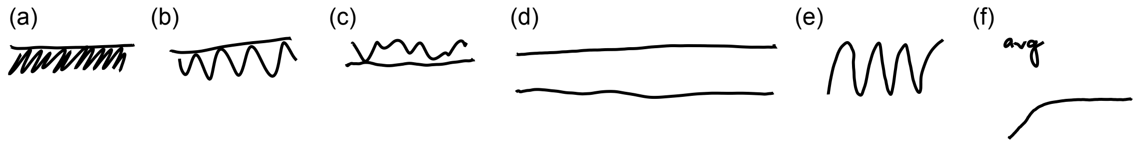


Figure 2.6: More involved multi-stroke query patterns that are not supported by tested time series pattern-matching algorithms.

2.5.5 Query Patterns

In order to get a better understanding of the types of questions users try to answer with the patterns they draw, we analyzed various properties of the sketches. Figure 2.5 shows some results of that analysis. (a) depicts the average ink distribution of all queries over a normalized sketching area (note that users were able to adjust the size of the sketching area). Users tend to focus on patterns that either target some extrema (corners) or align their patterns within the center region. We also recorded pen-speed as well as pen-pressure for each pattern. On average, velocity is highest within the center of the sketching area (b). We explain this through the ease-in/ease-out style that users tend to draw their patterns with. (c) suggests that there is no clear trend for pen-pressure as it appears to be uniform across the drawing area.

While most of our 112 pen-drawn query patterns were simple single stroke patterns that were intended to find subsequences which followed a specific trend, some of our users wanted to search for more complex relations in the data. Questions like “Find a period where a player consistently scored more than 20 points per game” were usually drawn with multi-stroke sketches by our users. Figure 2.6 shows some queries with these characteristics. The sketches in (a), (b) and (c) are instances where users wanted to search for inequalities (e.g., “find everything above / below X”), while (d)

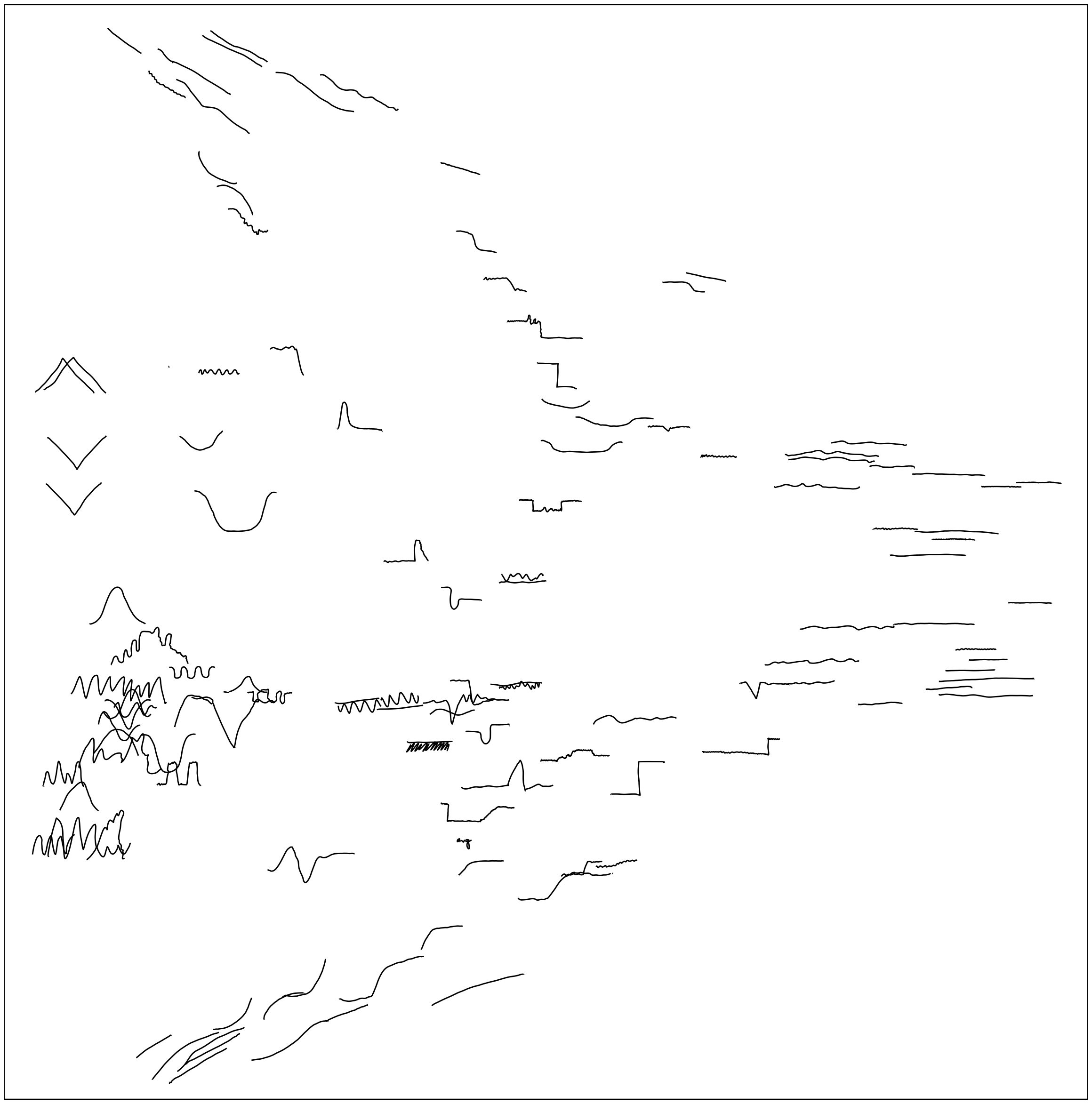


Figure 2.7: Automatic layout of all query sketches generated by applying PCA to the feature histograms of individual sketches.

depicts a range query (e.g., “find everything within X and Y”). Query (e) is an example where a user wanted to find time series which shows periodic fluctuations over a specific time period (“find all inconsistent basketball players”) . However, our user did not intend to find time series that were close in shape to her sketch. (f) is a case, where a user annotated the sketch to provide some more information to the system. It is important to note that current state of the art time series pattern-matching algorithms, such as the ones we tested, do not support such queries directly. In a future effort, we aim to collect more hand-drawn sketches and try to extract a common sketching language that could be used to express other relations such as inequalities, ranges, fuzziness or fluctuations.

2.6 Discussion and Future Work

To get an intuition of the quality of the produced human-annotated rankings we inspected them visually. Although we agreed on most of the decisions made by the crowd, there were cases where we felt that a pair of subsequences should have been swapped. It is important to note that apart from a qualification test, the workers’ answers were not validated or filtered using sentinel tasks or any other noise reduction technique. Also, with only 9 votes per pairwise comparison, chances are relatively high that some subsequences receive the same score. Increasing the number of required votes could help reducing ambiguities in generated rankings. Alternatively, multiple 9-round tasks could be averaged.

The analysis of the sampling methods used in our evaluation suggests that there are more suitable sampling methods than the ones used in our experiments. Sampling between two points artificially distorts the resulting time series. This is especially the case for short-length query pattern with high complexity. An adaptive sampling method, where the perceivably important points are aligned with the resulting time series could potentially lead to better results.

To further understand the different types of queries that users draw, we wanted to see if there are certain categories of patterns that occur across users and across datasets. Just by visually inspecting Figure 2.8, we can identify that for example straight-line or upward/downward trend patterns are drawn frequently. We obtain automatic clusters of such sketch categories by applying a similar approach as discussed in [48]. Each sketch is rendered as an image which we then represent by using a normalized feature histogram that is computed through a bag-of-words model over densely sampled local features. Figure 2.7 shows an automatic layout generated by applying dimensionality reduction (PCA) over these feature histograms. This initial experiment hints at the possibility of creating an accurate classifier to assign category labels to input sketches. Such labels could in turn be used to detect different pattern types, such as inequalities, range queries, etc. or to optimize algorithms through fast-rejection techniques.

We are planning to conduct a more rigorous study on what type of time series queries users are interested in. The results of the study could be used to define a visual vocabulary i.e. sketching-language which supports common types of queries. Because our approach of evaluating the subjective accuracy in time series pattern-matching can be extended to an entire conceptual query space, algorithms supporting these kind of queries could then be evaluated using the method proposed in this chapter. Other possible areas for future research include finding better sampling/alignment methods for hand-drawn queries (possibly using weighted strokes) and combining high and low-level similarity measures to improve the perceived accuracy of pattern-searches. For instance, DTW could be used to retrieve the top n matches while shape attributes described in [58] could be used to generate a ranking over the n matches.

Our comparison of algorithms and sampling strategies through our 8 collected human-annotated rankings offers anecdotal evidence that human-rankings can differ drastically from rankings generated by algorithms. For a better understanding of the different approaches, rigorous statistical analysis is needed. Such an analysis should cover a wide range of time series from different domains and queries drawn by experts in these domains. Our method to create human-annotated rankings can be used to create datasets for different domains that would, over time, allow for such an extensive analysis.

2.7 Conclusion

In this chapter, we showed why currently available datasets are not suitable for testing the subjective accuracy of similarity measures and demonstrated how human-annotated rankings can be created using crowdsourcing tasks. We published a freely available dataset comprising 8 different human-annotated rankings including the query pattern and 15 subsequences per query. This dataset can be used to optimize new and existing algorithms and serves as starting point for further research in this area. Furthermore, we collected 112 different query patterns from different users, analyzed their meaning and characteristics and proposed future ideas for a visual vocabulary for time series queries.

All queries and human-annotated rankings are available on
`http://cs.brown.edu/~peichmann/timeseries/`

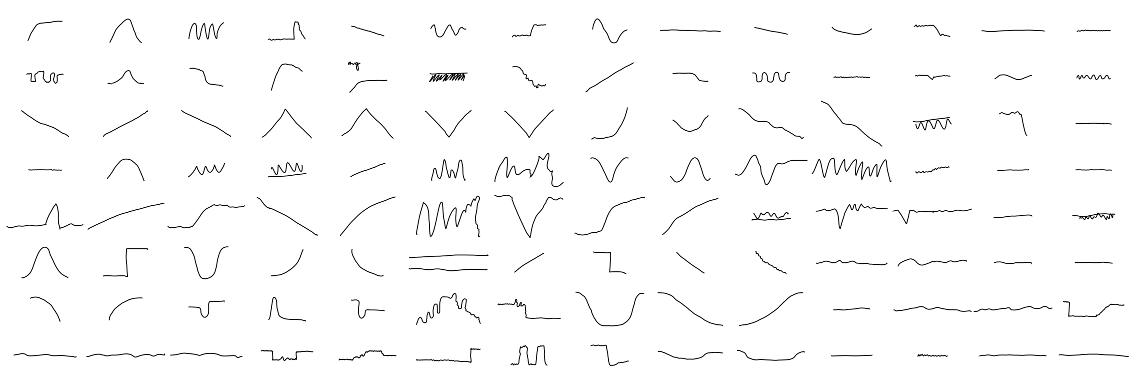


Figure 2.8: 112 Pen-drawn query patterns collected from 6 different users.

2.8 Acknowledgments

The authors wish to thank Bob Zeleznik, Joseph La Viola and Andries van Dam for their comments and suggestions, Yueguo Chen and Eamon Keogh for the source code as well all our participants and Mechanical Turk workers for their time and their invaluable feedback.

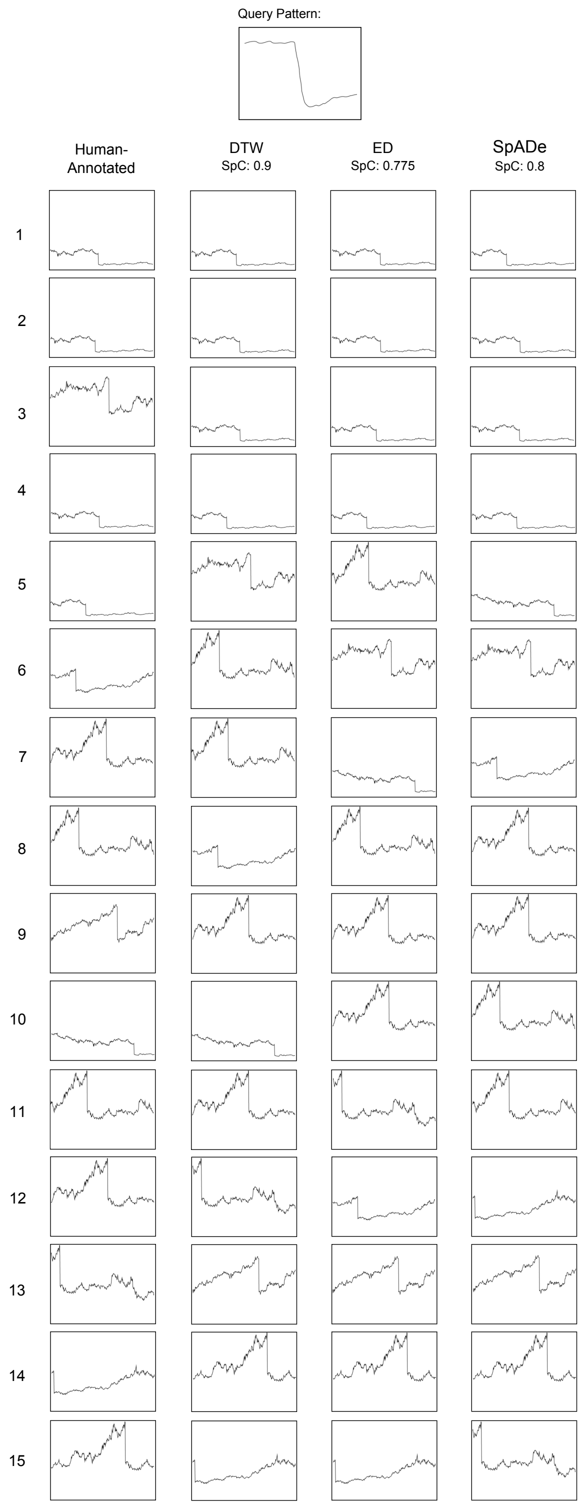


Figure 2.9: A human-annotated ranking for query pattern 2 and three different rankings produced by DTW, ED, and SpAde similarity measure using PIP sampling. The algorithms produce reasonable results according to the Spearman coefficient (SpC). Note that even the crowd sometimes fails to agree upon a most similiar shape (see rank 3 in the human-annotated ranking).

Chapter 3

Metro-Viz: Black-Box Analysis of Time Series Anomaly Detectors

3.1 Abstract

Millions of time-based data streams (a.k.a., time series) are being recorded every day in a wide-range of industrial and scientific domains, from healthcare and finance to autonomous driving. Detecting anomalous behavior in such streams has become a common analysis task for which data scientists employ complex machine learning models. Analyzing the behavior and performance of these models is a challenge on its own. While traditional accuracy metrics (e.g., precision/recall) are often used in practice to measure and compare the performance of different anomaly detectors, such statistics alone are insufficient to characterize and compare the algorithms in a systematic, human-interpretable way. In this extended abstract, we present Metro-Viz, a visual analysis tool to help data scientists and domain experts reason about commonalities and differences among anomaly detectors, and to identify their strengths and weaknesses.

3.2 Motivation

When tasked to perform time series anomaly detection (i.e., finding unusual data points or patterns) on a new data source [23], data scientists see themselves confronted with questions like: which anomaly detection systems work best for this specific task, where do these algorithms fall short, and why. Similar questions arise in virtually any classification domain. In machine learning, the performance of classifiers is typically evaluated using summary statistics such as precision, recall,

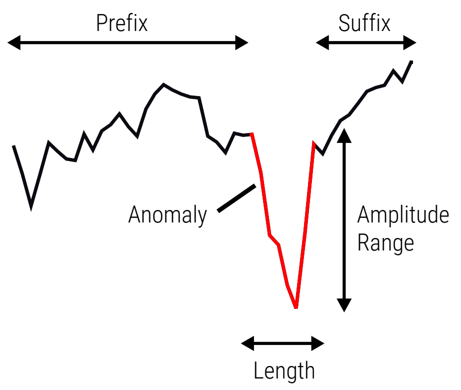


Figure 3.1: Example of implicit time series anomaly properties.

or F-score. In a recent paper, we introduced a scoring model that is based on these statistics, specifically tailored to time series classification systems [132]. Yet, while these metrics are useful for benchmarking and to get a sense of overall model performance, more involved questions that would lead to actionable insights for model improvements, such as the ones above, cannot be answered by any of these summary statistics directly [40].

To address this issue for tabular and image data, researchers have built tools like ModelTracker [9], Squares[113], and Google's What-If Tool [55], which allow users to inspect and understand correct and false predictions of a single machine learning model. However, these tools lack support for time series classification tasks. In this extended abstract, we introduce Metro-Viz, a system that aims to narrow this gap by providing a set of tools that are specifically designed for users who want to analyze the results of time series anomaly detectors in a systematic way. Through our work on building practical tools and systems for enabling next-generation time series anomaly detection solutions [131], as well as through closely collaborating with machine learning experts, we identified a set of functional requirements. We summarize these requirements in terms of four core challenges that we aim to address with Metro-Viz:

- *Challenge 1: Ambiguity.* What constitutes an anomaly depends on the application domain and context. For example, whether or not a sudden change in value or an irregular pattern should be considered as an anomaly might depend on factors such as the dataset and the task, the time of occurrence, periodicity, what happened before and after the anomaly occurred, as well as global trends (see Figure 3.1 for an example). Furthermore, anomalous patterns may evolve over time; what may have been anomalous a year ago might not be now. Exposing meta-data of anomalies helps data scientists to characterize how anomaly detectors behave under such conditions.

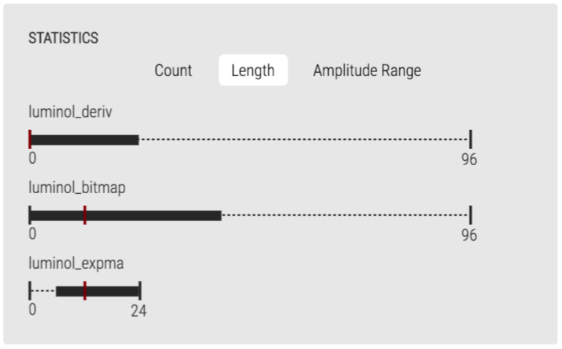


Figure 3.2: Users can inspect statistics, e.g., the length of all anomalies, in a box plot, showing the minimum, maximum, median, and interquartile range (IQR) per detector.

- *Challenge 2: Comparing Multiple Models.* We learned that many anomaly detection system developers struggle to compare multiple different models in a rigorous and systematic way, which goes beyond using simple summary statistics. This is mostly due to the overhead of having to create plots of success and failure cases, creating a mapping between the same anomalies detected by different detectors, or even more pragmatically, "seeing the data". Streamlining this process is crucial for a more efficient analysis.

- *Challenge 3: Time Granularity.* Anomalies might only become visible at certain time granularities or aggregation levels. Depending on how raw data is aggregated, e.g., by computing the average or maximum value for fixed time intervals like days or seconds, anomalous data points might get smoothed-out or accentuated. Understanding the impact of sampling and aggregation on time series anomaly detector performance is important when tweaking a detector for a specific use case.

- *Challenge 4: Missing and Unknown Labels.* Anomalies are, by definition, rare or unique events. This makes it difficult to determine ground-truth for certain problems. Captured training datasets therefore often only exhibit a small and incomplete set of possible anomalous patterns. As a result, classifiers might be misguided during the training process, requiring careful human testing and probing before deployment. Furthermore, it is infeasible to train anomaly detection algorithms for every possible scenario. Thus, functionality to modify input data and test hypothetical scenarios can help users to evaluate a model, even when training/test data is scarce.

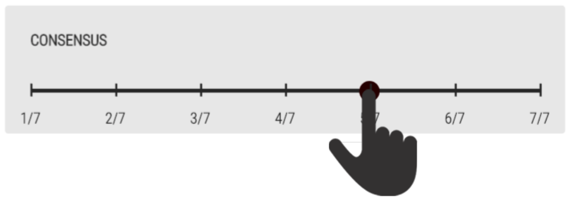
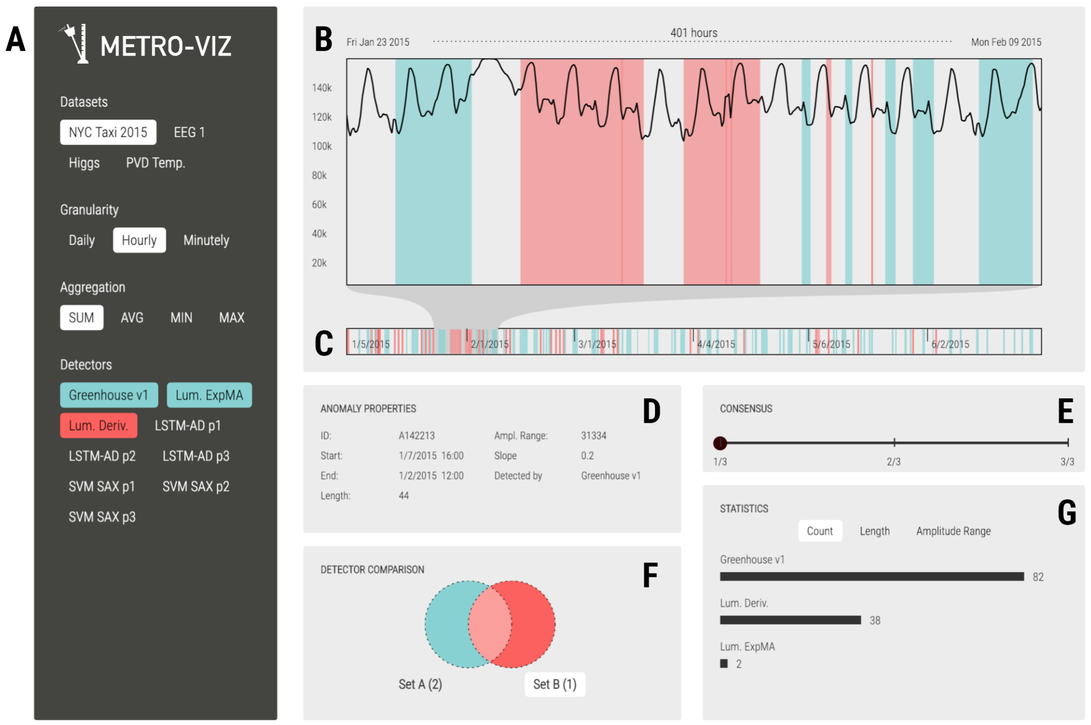


Figure 3.3: The consensus slider can be used to filter anomalies displayed in the time series viewer to ones that are detected by at least a certain number of detectors (5 out of 7 in this example).



ANOMALY PROPERTIES

ID:	A142213	Ampl. Range:	31334
Start:	1/7/2015 16:00	Slope	0.2
End:	1/2/2015 12:00	Detected by	Greenhouse v1
Length:	44		

STATISTICS

Greenhouse v1	82
Lum. Deriv.	38
Lum. ExpMA	2

Figure 3.4: A screenshot of Metro-Viz's user interface: A) Main menu, B) Time series viewer, C) Time axis showing the full time range, D) Anomaly properties of a currently selected anomaly, E) The consensus slider that can be used to limit the shown anomalies to those detected by a minimum number of detectors, F) The set operation widget that can be used to define and query anomaly sets, G) Summary statistics for all anomaly detectors selected in the menu.

3.3 Metro-Viz

Metro-Viz is a web application that runs in any modern web-browser, with an accompanying backend written in Python. It features a set of default anomaly detectors and allows users to plug in custom detectors by implementing an API that Metro-Viz uses to communicate with the detector. This enables Metro-Viz to be used with a diverse set of off-the-shelf or newly developed anomaly detectors, irrespective of the programming language in which they are written. Data can be loaded into Metro-Viz either by specifying a CSV file or by implementing a database adapter.

3.3.1 Features

In the following we describe the core functionality we built into Metro-Viz to address the challenges outlined in the previous section.

Visual Anomaly Analysis

To address Challenge 1, we developed tools to browse through time series data and analyze the detected anomalies. In Metro-Viz, a time series is displayed as a line graph in the time series viewer (Figure 3.4B), when selected from the main menu (Figure 3.4A). The time series viewer shows a window of a customizable number of data points. Users can pan to slide the window across the time axis (Figure 3.4C), or alternatively zoom into a subsequence of interest by using a drag-and-drop gesture. To trigger anomaly detection on the current time series, one or more anomaly detectors can be selected from the menu on the left. Found anomalies are highlighted in the line graph, color-coded by set membership (see Figure 3.4F). Hovering over the line graph displays different anomaly properties in a separate view below (Figure 3.4D). Furthermore, users can identify anomalies that are more likely to be true anomalies (those that are detected by more than just a single detector), by adjusting the *consensus* value through a slider (see Figure 3.3 and Figure 3.4E).

Comparing Multiple Detectors

Metro-Viz provides functionality to answer questions like: which anomalies were detected by both algorithm A and algorithm B, by algorithm A but not by algorithm B, and vice-versa (Challenge 2). Users can assign detectors to either one of two sets, and click on the desired portions in a Venn Diagram (Figure 3.4F and Figure 3.5) to select the anomalies displayed in the time series viewer. By default, two anomalous ranges are defined to be equal if they share at least one data point. The amount of overlap required for the system to consider two subsequences as equal can be configured in the settings.

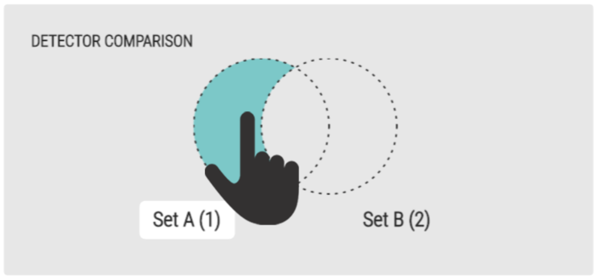


Figure 3.5: Anomaly detectors can be assigned to either set A or set B. This allows users to selectively display anomalies, e.g., only anomalies detected by detectors in set A but not in set B, by clicking on the corresponding portion in the Venn diagram.

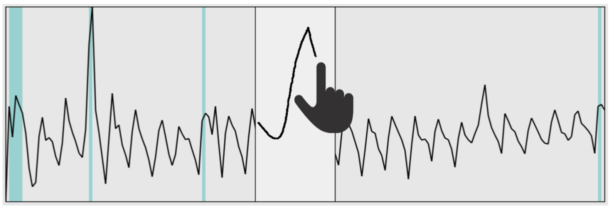


Figure 3.6: Metro-Viz allows users to erase arbitrary portions of a time series and replace them with a sketch in order to probe anomaly detectors with a pattern not present in the data.

Aggregating across Time

To address Challenge 3, time series can be aggregated across different time scales/granularities (Figure 3.4A), e.g., day, hour, minute, using four different aggregation functions (`sum`, `avg`, `min`, `max`). Time series data that is not explicitly time-stamped, but has an implicit order can be aggregated using a set of configurable sampling methods.

Interactive Hypothesis/What-If Testing

We address Challenge 4 by enabling data scientists and domain experts to simulate and understand the behavior of anomaly detectors thorough patterns not present in the data. For example, users can test scenarios like: Would detector A still classify this peak as an anomaly, if its amplitude were lower or if its prefix were different? Metro-Viz provides features to modify a time series displayed in the time series viewer. Data points can be selected through a gestural interface, as proposed by [46]. Once selected, these points can be moved along the vertical axis, changing the amplitude of the selected subsequence. Furthermore, similar to functionality introduced in TimeSketch [47] or Qetch [91], users can select a subsequence in the time series viewer and replace it with a hand-drawn sketch (see Figure 3.6). A modification to the original time series will automatically re-run all selected detectors. Immediate responses from the backend allow the user to engage in a back-and-forth dialog with the system when probing detectors.

3.4 Future Work and Conclusion

Detector-Guided Labeling

With the increasing variety of streaming data, users become data curators in addition to analysts [3]. While Metro-Viz has proven useful within a small group of test subjects to analyze anomaly detection algorithms, our pilot users expressed that in many cases they would have liked to mark anomalies as false positives or false negatives, i.e., adding or refining ground-truth labels while analyzing the results. Based on this feedback, we are currently extending Metro-Viz with functionality to perform what we refer to as *detector-guided labeling*. This means that, instead of having users manually scan a time series to annotate anomalous ranges, Metro-Viz will guide users to anomalous time series ranges discovered by a set of pre-trained detectors, and ask users to classify them as false, true, or partially true by correcting the anomalous range. Given that multiple pre-trained detectors are used, every anomaly can be assigned a probability (confidence) of it being a true anomaly, analogous to the consensus feature outline above. Our hope is that users will experience the task of labeling time series as less daunting and will be more efficient, as anomaly verification can be done iteratively, starting from the ones with high confidence to the ones with low confidence.

Scalable Data Backend

Motivated by the scale of the data that we received from our collaborators in healthcare and computer networking sectors, the database experts on our team are currently architecting a data management system that will allow users to scalably perform Metro-Viz style interactive analysis on large datasets.

Conclusion

In this extended abstract, we presented an early design of Metro-Viz, a tool that allows data scientists and domain experts to analyze the behavior and results of time series anomaly detectors. Further information about this work is available on our project website [1].

3.5 Acknowledgments

This research has been funded in part by Intel and by NSF grant IIS-1514491.

[1]http://metronome.cs.brown.edu/

Chapter 4

Orchard: Exploring Multivariate Heterogeneous Networks on Mobile Phones

People are becoming increasingly sophisticated in their ability to navigate information spaces using search, hyperlinks, and visualization. But, mobile phones preclude the use of multiple coordinated views that have proven effective in the desktop environment (e.g., for business intelligence or visual analytics). In this work, we propose to model information as multivariate heterogeneous networks to enable greater analytic expression for a range of sensemaking tasks while suggesting a new, list-based paradigm with gestural navigation of structured information spaces on mobile phones. We also present a mobile application, called Orchard, which combines ideas from both faceted search and interactive network exploration in a visual query language to allow users to collect facets of interest during exploratory navigation. Our study showed that users could collect and combine these facets with Orchard, specifying network queries and projections that would only have been possible previously using complex data tools or custom data science.

4.1 Introduction

People regularly interact with complex information networks that model the attributes and relationships of real-world entities. Whether through browsing hyperlinked networks or the ranked lists of search results, navigating between related pages, people, products, and media is a universal quality of the online experience. At the same time as the search, list, and link paradigm is expanding to ever more information seeking experiences, the trend towards integrating analytic sensemaking capabilities in consumer experiences such as video games and fantasy sports is driving the adoption of business-like dashboards beyond the business context [95].

The representation and analysis of complex information networks has rich history in the fields of information visualization and visual analytics, with the analysis of social networks (e.g., [17]) and citation networks (e.g., [81, 69, 25, 156]) featuring prominently. Additional use cases include exploratory analysis of data published by governments (e.g., migration flows [137]) or businesses (e.g., company-investor networks [25]), as well as investigative analysis of potential impropriety or illegality (e.g., in Enron email communications [121], Panama Papers documentation [78], or "open secrets" interactions between politicians and lobbyists[2]).

What is less well examined in the literature, however, is how such information networks may be explored on mobile devices. We posit that there are potentially significant latent demands across many everyday activities increasingly performed on mobile phones. These include sports and gaming analysis (including fantasy sports and e-sports), media consumption (e.g., music, books, films and videos, news), and personal research into major life decisions (e.g., jobs, houses, schools, cars, vacations). In this work, we present a new interaction paradigm for exploring such information on mobile phones.

We draw design inspiration from faceted approaches to search, browsing, and analysis in which users select from prefabricated filters. These filters are automatically extracted from multiple orthogonal data dimensions and their application has the effect of progressively reducing the number of matching entities to a meaningful and manageable subset. Notable examples of this approach from the literature include FacetLens [85] and Immense [88], which present multiple coordinated views in a fixed 2D layout, and GraphTrail [42] and PanoramicData [153] that provide freeform 2D canvases on which to compose queries and inspect results. Business Intelligence dashboards in general-purpose tools like Tableau, Qlik, and Power BI may also be constructed in a faceted style (e.g., to allow browsing of news and social media "documents" [45]).

We note that all such faceted interfaces have deficiencies when applied to multivariate heterogeneous networks on mobile phones. First, faceted interfaces do not convey the underlying heterogeneous network structure. They do not present how facets of nodes in a network or nodes themselves relate to one other, making it difficult for users to grasp the concept of pivoting between different node types. Second, faceted interfaces as developed in the literature do not translate well to mobile phones, where space is limited. While it is technically feasible to employ such interfaces within mobile applications, they would likely impair the user experience. For example, navigating between views in 2D layouts would require users to pan and zoom excessively (analogous to when browsing non-mobile-optimized websites).

To address these issues, we present Orchard, a solution that applies faceted search concepts to the exploratory analysis of multivariate heterogeneous networks on mobile phones. Our design draws on the familiar concept of keyword search and result listing, a paradigm that has already established itself as the dominant form of mobile information seeking. With Orchard, users can explore aggregated node and edge attributes through sortable list-based views that simultaneously function as horizontal bar charts for aggregate values. Orchard combines faceted search on these views with lateral pivoting (following links across one or many hops) that automatically "collects" the ensuing facet trail for future use.

Orchard pivoting extends prior work on interactive network exploration by allowing free navigation between nodes of *any* type, whether or not they are connected directly in the data model. By inferring a full graph query specification from any partial query, demand for additional user input is minimized and the result is an uninterrupted pivoting experience. Orchard also introduces the notion of *reflective pivoting*, i.e., pivoting from/to the same node type. While reflective pivoting is usually considered a *no-op*, Orchard fills this semantic gap by interpreting commands to pivot nodes onto themselves as a command to project the nodes into homogeneous network in which nodes are connected based on shared connections to nodes of another type (as selected by users). This allows users to seamlessly transform and view any subset of a heterogeneous network as a homogeneous network, while collecting more expressive facets in the process (such as pairs of nodes connected by highly-weighted links).

In summary, our contributions are:

- We introduce and motivate the concept of interacting with multivariate heterogeneous networks on mobile phones.

- We present a novel query specification model for multivariate heterogeneous networks called *Pivot Trails* that is particularly suitable to be implemented on small screen devices, requiring minimal user input to formulate network queries.

- We design and implement *Orchard*, a mobile application that uses Pivot Trails to support casual exploratory workflows over multivariate heterogeneous networks using a list-based paradigm with gestural navigation.

- We report a qualitative evaluation of the design of Orchard with 12 participants. Initial results show that users quickly understand Orchard's data and query model, and that they are able to accomplish common exploratory network tasks.

4.2 Background and Related Work

Orchard relates to prior work on supporting information exploration in heterogeneous information networks, and is inspired by work on exploratory search [92, 147]. An information network represents an abstraction of the real world, focusing on the objects (nodes) and their attributes, and interactions (links) between the objects [129]. In multivariate heterogeneous networks, multivariate refers to the fact that each node or link in a network can have multiple attributes, and heterogeneous indicates that there are multiple node or link types in a network.

4.2.1 Core Concepts

There is a large body of prior work on facilitating exploration and analysis of multivariate heterogeneous networks, including the development of taxonomies describing the nature of tasks associated with such networks [83, 6, 110]. We have identified five core concepts (CC1-5) that capture the characteristics of prior network exploration systems.

CC1: Faceted Search. To allow iterative and incremental filtering of large information networks down to a few items of interest, a common solution is to provide keyword search functionality and the means to apply filters based on a facet type, facet value, and facet count. For example, in faceted search of a movies dataset, one facet type might be “Genre”, its values might include “Drama,” “Comedy,” and “Sci-Fi,” and each of these values may be accompanied by the count of movies in the dataset matching that facet value. Early approaches to faceted search, such as Flamenco [149], presented clickable facets that could be progressively applied as filters to “drill down” into a target dataset. Similarly, FaThumb [70] was the first to introduce faceted search on mobile devices. Visualizations have also been introduced to the faceted search experience as an aid navigation. For instance, PaperLens [81] features multiple views to formulate a limited set of publication data-oriented questions, such as the relationships between authors or frequently referenced papers. FacetMap [126] is a domain-agnostic generalization of PaperLens that supports simple searching and browsing tasks using a tree-map that scales with display size. FacetLens [85] extends this line of work further by exposing facet values as visualizations such as bar charts whose elements can be selected directly.

Our approach with Orchard extends the functionality of these systems in several important ways. First, facet values are shown in a list-based view, allowing users to sort either alphabetically by facet value or numerically by facet count and other aggregations. Second, users can select multiple facet values from such lists and choose to match either any or all of these values. Third, by incorporating network projection into the analysis of heterogeneous networks, users can retrieve potentially interesting combinations of facet values more easily and systematically.

CC2: Topological Search. Researchers have devised various visual query languages to express network queries (e.g., [18, 21, 24]). A recent example of such a system is Visage [109], a visual graph querying approach in which users iteratively refine attribute and topological constraints on a 2D layout. The system populates user-defined patterns with examples and allows them to browse other possible matches.

Unlike Visage, Orchard does not require users to define topological constraints by specifying a graph, but instead lets them explore aggregated nodes simply by following links.

CC3: Node and Link Aggregation. A common issue when dealing with node-link representations of multivariate networks is space and legibility – such representations can easily turn into undifferentiated hairballs for large, dense networks [44]. To visualize arbitrarily-sized networks compactly, nodes and edges can be aggregated rather than represented individually. Aggregation is typically applied on the attribute level, e.g., meta-nodes representing all nodes with a given categorical attribute value or numeric attribute range. For example, PivotGraph [146] aggregates and positions nodes based on their attributes in a grid, encoding the relationship between node aggregates by the thickness of visual links. Similarly, OntoVis [120] uses information in the ontology associated with a network to semantically prune a large, heterogeneous network. Juniper [103] lets users explore the relationships of nodes in selected sub-networks using tree visualization, where aggregated nodes can be collapsed and expanded. Finally, both GraphTrails [42] and work by Van den Elzen & Van Wijk [137] use common data visualizations such as bar charts and tag clouds to summarize subsets of nodes.

Orchard's approach to aggregation is similar to GraphTrails' in that users can choose how facet values of a particular node type are grouped and aggregated. However, in addition to showing count of nodes per group, Orchard provides additional metrics, such as the count of nodes of a linked node type, that users can sort by.

CC4: Pivoting to Linked Nodes. Pivoting is complementary to searching and filtering as it expands, rather than reduces, the options for onward navigation. Given a subset of nodes, users can choose to "pivot" to view linked nodes, e.g., those of a specified type. Various styles of pivoting between node types have been proposed in the literature. FacetLens [85] provides a one-to-many pivot mechanism by displaying pivot options for a selected node. In PivotSlice [156], users can configure a 3-by-4 multiple focus view that sub-divides and visualizes network data based on user-defined facets. PivotPaths [39] enables pivoting between partially overlapping sets of resources and facet values, and attempts to make pivots more comprehensible by using animated transition to highlight the overlap in elements between successive views. GraphTrails [42] introduces a many-to-many pivot mechanism that allows users to pivot from a subset of nodes to the set of directly connected nodes, providing a freeform canvas on which users can lay out multiple pivot paths in 2D space.

Orchard adopts a similar kind of many-to-many pivoting as GraphTrails. However, in Orchard users can freely navigate from a set of nodes of a particular type to directly or indirectly linked nodes of *any* other node type in the network, preventing them from having to fully understand the underlying network structure and from getting stuck in leaf nodes.

CC5: Projections of Sub-networks. Given a sub-network, it is often insightful to analyze interactions between the nodes of a particular type as a homogeneous network. A commonly used technique is to “project” a heterogeneous network to a homogeneous one by connecting two nodes of the same type if both are connected to the same node of a different type [77]. Systems such as Orion [60] and Ploceus [90] are built to support users in formulating such transformations as a pre-processing step: users can map tabular data to a network structure, by defining the semantics of links between nodes. Such projections are more readily interpretable as node-link representations than the full heterogeneous network. Typical analyses of homogeneous networks include the extraction of connected components and communities of nodes that are preferentially attached to one another.

Orchard does not make this assumption, but instead promotes on-demand projection of any node subset to homogeneous networks, all as part of an exploratory workflow rather than as a pre-processing step.

4.2.2 Data Exploration on Touch-Enabled Devices

Designing visual interfaces for exploring data on touch-enabled devices is becoming an increasingly important topic within the information visualization community [82, 79, 29, 80]. For example, PanoramicData [153], Vizdom [33], and SketchInsight [84] featured a pen and touch interface for visual data exploration on large interactive displays. Tangraphe [134] proposed a set of single hand, multi-touch gestures for interactive exploration of network visualizations, while Schmidt et al. presented a set of multi-touch interactions for network visualizations, focusing on edge interactions [118]. Prior work has also focused on smaller tablet devices. For example, TouchWave presented a set of multi-touch gestures to interact with a stream graph on tablet devices [15]. Kinetica employed physics-based affordances with multi-touch interaction for multivariate data exploration on a scatterplot [116]. TouchViz evaluated a gestural interface for choosing, filtering, and sorting data in familiar charts on tablet devices [41]. Sadana and Stasko [117] have also explored challenges in creating multiple coordinated views in data visualization systems for tablet computers. Finally, TouchPivot [66] proposed a pen-and-touch user interface that aids visual data exploration on tablet devices, targeting novice users.

Orchard builds on this trend towards touch-enabled data exploration, focusing on single-touch gestures on mobile phones for the simplest and more broadly applicable interaction style for casual everyday activities.

4.3 Design Goals

Our research focuses on the casual exploration of multivariate heterogeneous networks, specifically in the context of mobile phones. We set out by designing and implementing various prototypes to support individual low-level tasks applicable to heterogeneous networks (Table 4.1), and by combining them to support end-to-end exploratory workflows. To derive a set of design goals we presented and tested early designs for solving these tasks on mobile phones with invited test users and colleagues at our research institution. We prepared various test datasets ranging from movie databases to social news feeds, academic publication databases, and publicly available communication networks. The challenges we faced while iterating on the designs with our test users can be broadly categorized into representational and functional challenges. On the one hand we sought to find suitable mobile-friendly representations for techniques that have been proposed in the literature in the context of network analysis for larger displays. On the other hand, we learned that in some cases users were interrupted in their flow due to missing functionality that has not been explored in prior research multivariate heterogeneous networks. Guided by these findings we identified two representational (DG1 and DG2) and two functional design goals (DG3 and DG4) that we address in this paper.

DG1: Introduce mobile-friendly list-based representation. Views and operations to explore networks should be exposed through a design suitable for the targeted form factor. With Orchard we aim to support casual exploration scenarios through a simple user interface design reminiscent of that of other information retrieval applications for mobile phones. Our goal is to introduce a new list-based paradigm with gestural navigation of large multivariate heterogeneous networks.

DG2: Support lightweight gestural navigation. Transitioning between different node types plays an important role in analyzing heterogeneous networks [42]. However, understanding which

ID	Low-Level Task
L1	Determine the attributes and values associated with nodes
L2	Find the nodes with specific attribute values
L3	Find a derived property of a set of nodes with specific attribute values
L4	Find the set of entities adjacent to or accessible from an entity
L5	Find a derived property of the entities adjacent to or accessible from an entity
L6	Find the entity with the maximum/minimum number of adjacent entities
L7	Common connection: Given a set of entities, find a set of entities that are connected to all of them
L8	Follow path
L9	Revisit an entity and infer further knowledge
L10	Characterize sets of nodes as belonging to different groups, based on node attributes

Table 4.1: A set of low-level tasks from Pretorious et al. [110] that are applicable to heterogeneous networks.

nodes can be pivoted to and deciding on which nodes to pivot to places extra cognitive load on users. To promote free-form exploration, we aim to make pivoting a lightweight, single-touch operation that can easily be reverted, allowing users to quickly explore multiple pathways.

DG3: Consolidate drill-down filtering and lateral pivoting. During testing of early prototypes of Orchard we observed that users often select facets of interest of different node types while pivoting through the network, until at some point they want to apply multiple previously selected facets to the current set of nodes. Drill-down filtering in the style of faceted search can be used to quickly narrow down a collection of data items to a smaller subset, whereas lateral pivoting allows users to move between sets of nodes related through multiple hops. Our goal is to consolidate these complementary mechanisms by allowing users to turn a sequence of lateral pivot operations into facets that can be applied as filters, or vice versa.

DG4: Integrate network projections into exploratory analysis. Although heterogeneous and homogeneous networks can hardly be explored in the same view, we postulate that users benefit from on-demand network transformations as part of a regular network exploratory workflow. Being able to project arbitrary sub-networks of interest to homogeneous ones allows users to gain insights about such sub-networks from a different, customized perspective, in ways that can inform decisions on what to explore next in a heterogeneous context.

4.4 Pivot Trails

To retrieve the data necessary to render network nodes in a list-based visualization (DG1) users must be able to specify aggregation queries for subsets of nodes satisfying topological and attribute constraints. In this section, we introduce a query specification model called *pivot trails*, designed to capture common queries for nodes in multivariate heterogeneous networks, and capable of consolidating drill-down filtering and lateral pivoting (DG3).

4.4.1 Data Assumptions

Our pivot trails model relies on three assumptions on the data.

First, every node instance of a node type in a heterogeneous network can have an arbitrary number of attributes/facets.

Second, the network is multi-partite (k-partite), meaning that it can be divided into k independent sets, such that no link connects two nodes of the same type. Consider, for instance, Figure 4.1 showing how different entities in a film dataset and the relationships between them are represented in a k-partite network.

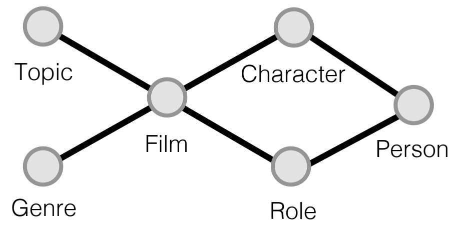
Third, links are homogeneous and univariate, meaning that there exists one link type only, with no associated attributes. Instead, a heterogeneous relationship is modeled by association to new node type. Consider, for instance, the relationship between `Person`, `Role`, and `Film` in Figure 4.1. An alternative schema could model `Role` as an attribute of a link connecting Person and Film. Yet, promoting heterogeneous relationships to their own node type obviates the need for the concept of *links* in a user interface, without the loss of any of the link semantics.

Furthermore, because the network is k-partite any set of nodes containing nodes of two different types can be projected to a homogeneous network. In Figure 4.1, for instance, the sub-network containing topics and films can be projected onto two different homogeneous networks: a `film` $\leftrightarrow$ `film` network, where two films are connected by a weighted link representing the count of common topics, or a `topic` $\leftrightarrow$ `topic` network, where two topics are connected by a weighted link representing the count of films about the two topics.

4.4.2 Query Specification Model

A typical aggregate network query consist of the following components [42]:

- A set of *topological and node attribute constraints* that describe how to arrive at a subset of nodes of interest;



Film	Title, Release Year, Revenue, Runtime, Rating
Topic	Category, Name
Genre	Category, Name
Character	Name
Role	Name
Person	Name, Year of Birth, Year of Death

Figure 4.1: The schema of a multivariate heterogeneous network containing information about films, actors, and so on, as well as the attributes for every node type.

- A *grouping strategy* for the attribute of interest in the retrieved subset of nodes; and
- An *aggregation strategy* that defines which metric should be calculated for each bin.

With *pivot trails* we introduce a convenient way to model network queries as topological and node attribute constraints, applicable to a wide range of network exploration scenarios. Intuitively a pivot trail defines the order in which to traverse, filter, and combine nodes in a heterogeneous network to a set of nodes of interest, i.e., a query result. Pivot trails are directed graphs that can easily be translated to graph query languages such as Cypher, or even SQL.

Formally, assume a k-partite heterogeneous network H where each node is of a node type t_m where $m \in \{0, 1, \ldots, k-1\}$ (Figure 4.2a). A pivot trail is a directed acyclic graph. Each node P_i in a pivot trail (*pivot node*) has a node type $T(P_i)$, a set of node-attribute constraints (filter predicates) $C(P_i)$, and a set operation $O(P_i)$. A pivot trail contains exactly one target node P_n, where $T(P_n)$ is the node type of interest.

A pivot trail defines how nodes in H of a particular type are retrieved. The pivot trail shown in Figure 4.2b, for instance, translates to the following query: retrieve all nodes in H of type $T(P_0)$. For all nodes that satisfy constraints $C(P0)$ traverse their links leading to a set of nodes of type $T(P1)$. For all nodes that satisfy constraints $C(P1)$ traverse their links leading to a set of nodes of type $T(P2)$, and so on, until the target pivot node is reached.

Pivot Trails are designed to express both lateral pivoting (navigating from a subset of nodes of type a to a set of linked nodes of type b, where $a \neq b$) and drill down filter operations (navigating

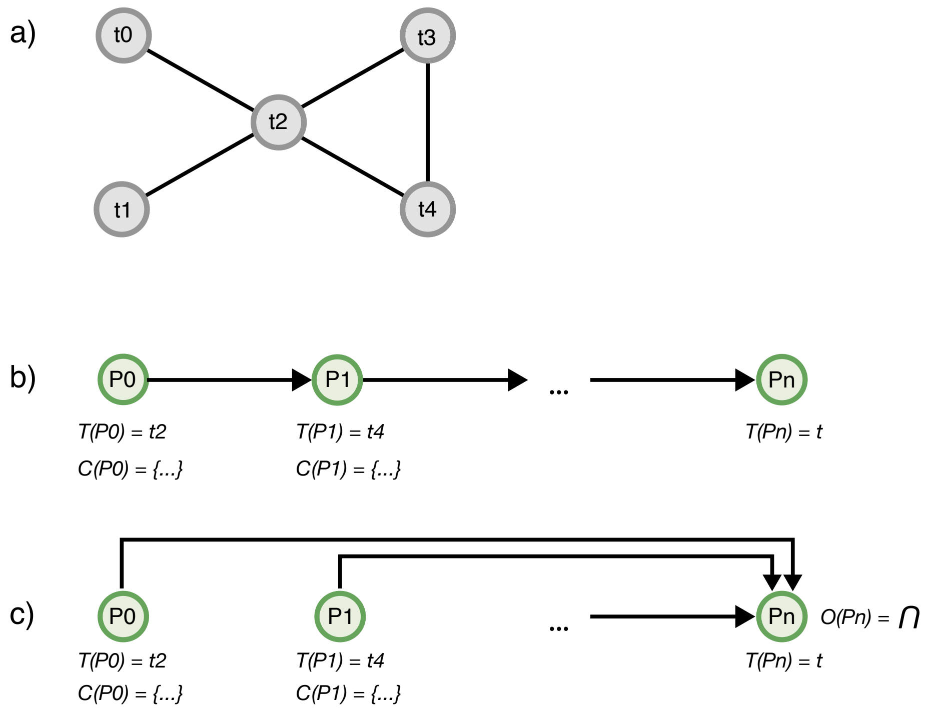


Figure 4.2: a) An example schema of a heterogeneous network with node types t_0 - t_4; b) a pivot trail representing a lateral pivoting operation; and c) a pivot trail where all pivot nodes are directly linked to the target pivot, representing a drill down operation (set operations for pivot nodes with one or no incoming links were omitted)

.

to a set of nodes that are linked to m other sets of nodes, where $m > 1$), or combinations thereof. Figure 4.2b, for instance, represents a sequence of lateral pivot operations. Conversely, this pivot trail can be re-structured to a drill down operation by linking any or all $n - 1$ pivot nodes directly to the final pivot node P_n and setting its set operation $O(P_n)$ to $\cap$, as shown in Figure 4.2c.

A pivot trail graph does not need to be fully specified, but can be inferred from a partial specification (at the minimum a target pivot node must be given). In contrast to traditional query models proposed in the literature, which only consider adjacent nodes as valid pivot paths, this allows users to formulate queries in a more flexible way. For instance, given the schema in Figure 4.1, people can be directly related to genres or topics, without the need to specify intermediate hops. Figure 4.3 shows a number of concrete examples of user-defined pivot trails and how full pivot trails are inferred from partial ones using the schema in Figure 4.1.

4.5 Orchard

Guided by the four design goals (DG1-4) presented in Section 4.3 we built a research prototype, called *Orchard*. To inform the design of a visual exploration user interface that incorporates the five core concepts (CC1-5) of network exploration outlined in Section 4.2.1, we carried out an iterative design process (Section 4.3). In this section, we present the final design of Orchard.

4.5.1 Implementation

Orchard is a mobile application written in TypeScript and HTML/CSS, built to run on all modern mobile phones. It uses a dynamic layout that fits the screen of the phone and is designed to be used in portrait mode. Orchard is backed by a C#-based implementation of the query model we propose in Section 4.4.

4.5.2 Attribute Exploration

The *Attribute Exploration View* (Figure 4.4), Orchard's default view, is to browse aggregates of nodes in a network. Every item in the list represents an aggregate (a category for a categorical attribute or a range for a numerical attribute) defined by the grouping and aggregation strategy (Figure 4.4a,b). List items contain a bar whose length encodes the aggregated value per bin, such as the count of all nodes linked to the nodes in a bin, the average of one of their numerical attributes, etc. Tapping on one of the list items reveals more details about the nodes in that group. By tapping the grouping strategy (Figure 4.4a), users can switch the node type of the current pivot node and

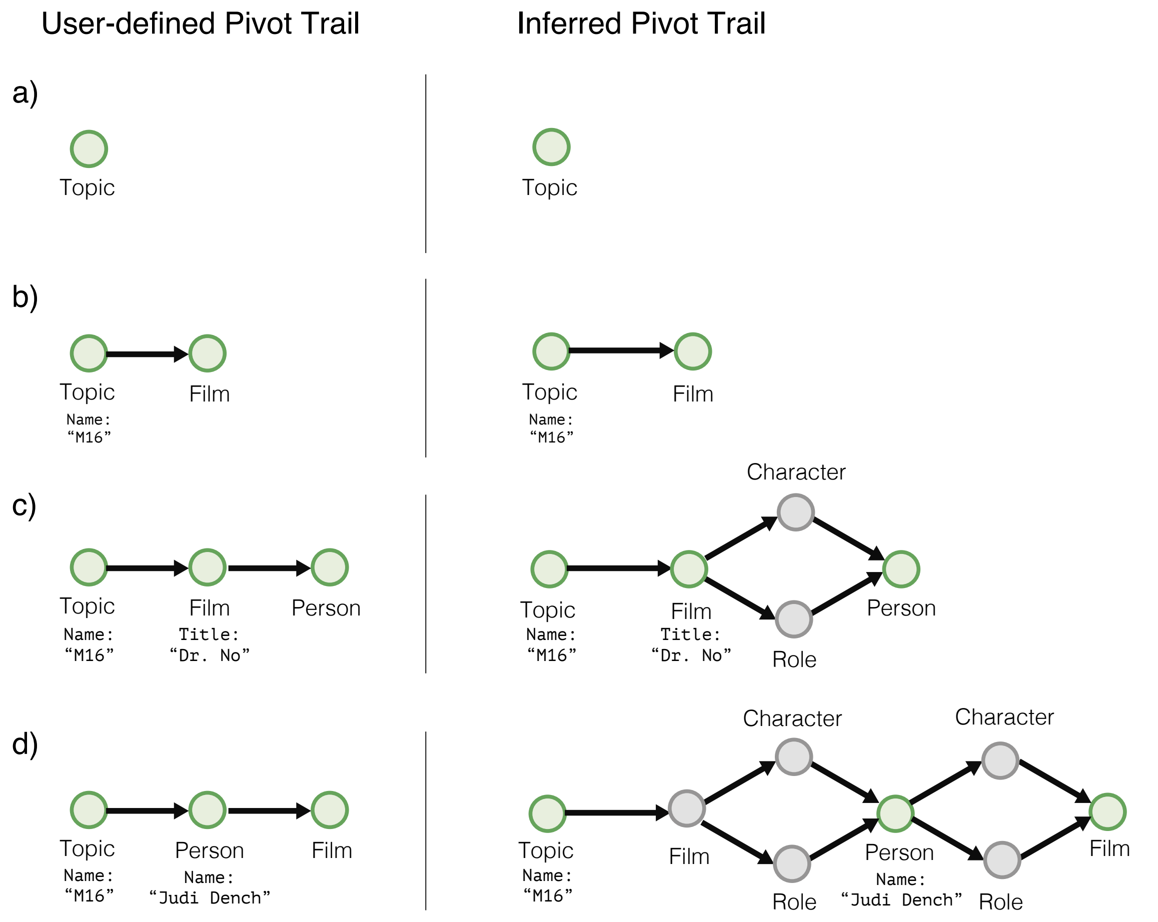


Figure 4.3: Examples of pivot trails. a) find all topics; b) find all films associated with a topic whose name includes 'MI6'; c) of films related to a topic containing 'MI6' in its name, find all people linked to the films containing 'Dr. No' (note that the intermediate nodes required to reach `Person` from `Film` are not explicitly specified, but inferred from the network schema); and d) find all the films that are linked to person nodes containing 'Judi Dench' in their name, which are linked to a topic containing 'MI6' in their name.

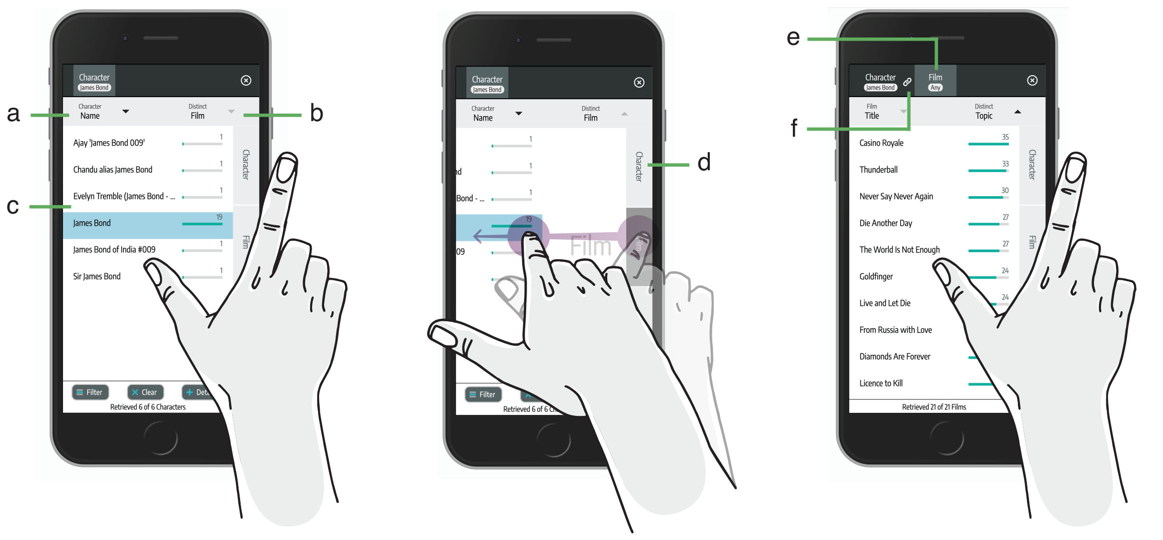


Figure 4.4: Orchard's facet exploration view and swipe-based pivot mechanism. Users can configure how nodes are grouped (a), aggregated (b), and then visualized in the bar chart-like list (c). The pivot bar (d) displays all node types of a dataset as tabs along the right edge of the application. Starting a swipe-left gesture on one of these node types moves the current view to the left and pivots to all nodes in the network that are linked to nodes in the selected group (films linked to characters named "James Bond"). Upon completion of the gesture, a new instance of the swiped node type is added to the pivot trail (e) and the result list is updated. Since the first pivot node is always directly connected to the target pivot node when there are only two nodes in the pivot trail, the link between them is activated (f).

pick a facet of interest. Similarly, the aggregation strategy can be changed by tapping on the second column (Figure 4.4b). There are two types of aggregate values that can be computed:

1. The count of distinct linked nodes of a certain type (e.g., the number of films per actor name, how many films have the same title or a rating between 90 and 100).
2. The min/mean/max of linked numeric node attributes (e.g., the mean revenue of all "Alice in Wonderland" films, the average age of all people per film).

A default grouping and aggregation strategy can be configured per attribute per node type. For instance, users can choose to always display the count of associated films when looking at genre, or the count of associated people when looking at characters. Furthermore, lists can be sorted by the facet value (Figure 4.4a) or by the aggregated value (Figure 4.4b).

4.5.3 Visual Query Interface

To support network exploration we designed a visual query interface that implements the pivot trail model. Orchard exposes a visual representation of a pivot trail as horizontal list at the top of the screen. On start up, a single pivot node of a configurable default node type is shown in the pivot trail, which corresponds to the data visualized in the Attribute Exploration View. The pivot bar on the right of the screen shows all pivot node options, and scrolls vertically if the number of pivot nodes exceeds the height of the screen. Users can add a pivot node to the end of the pivot trail (Figure 4.4d) by performing a "swipe left" gesture on the pivot node. As the finger slides across the display while swiping, the current list moves to the left and visual feedback indicates which type of pivot node users are about to add to the pivot trail (Figure 4.4).

In open-ended browsing sessions, we observed that users often select facets of interest of different node types while pivoting through the network, until at some point they want to apply multiple previously selected facets to the current set of nodes. Orchard supports this browsing behavior (*DG3*), allowing users to choose between lateral pivoting and drill down filtering by changing how nodes in the pivot trail are linked. By default, all nodes in a trail are linked sequentially, analogous to the depiction in Figure 4.2b. To switch from lateral pivoting to drill down filtering, users can link individual pivot nodes to the target pivot node by pressing the link icon to the right of each pivot node. An activated link indicates the corresponding pivot node is linked directly to the target pivot node, a de-activated link indicates that the pivot node is linked to its successor in the trail.

By pivoting from subsets of nodes to new subsets, users keep adding pivot nodes to the pivot trail. They can navigate back and forth in the pivot trail by swiping left or right, analogous to traversing an undo/redo stack. They can also jump to or remove a specific node, or start a new trail from an existing pivot node by using the context menu that appears when tapping a node in the pivot trail. Furthermore, users can constrain the subset of nodes from which to pivot by selecting list items, and select whether *all* or *any* of the selected items should be matched via the context menu. Pivoting from a list of nodes where no selection has been made is treated as if all items were selected.

4.5.4 Keyword Search

A common strategy to explore large graphs is a bottom-up approach, where the analysis begins with a search and more context is added as needed [138, 143]. Orchards provides keyword search functionality where partial matches of node attributes are displayed in a list (Figure 4.5A). Users can navigate to the matched nodes by tapping on one of the results, which initializes the pivot trail with the required pivot node and constraints.

4.5.5 Reflective Pivots

A special type of pivot operation are *reflective pivots*, i.e., pivoting from/to the same node type. As there are no self-referential nodes in *k*-partite networks, this interaction is, in theory, meaningless. With Orchard we address this semantic gap by treating two consecutive pivot nodes with the same type as an instruction to project their matching nodes to a homogeneous network (*DG4*). Such a projection generates a link between any pair of matched nodes, if they are linked to the same node of another node type specified by users. The assigned link weight between such a pair of nodes is defined by the number of distinct nodes to which the pair is connected in their heterogeneous context. Since network projections are computationally expensive, Orchard computes the results in a progressive fashion, giving users the ability to refresh the results on demand. The result of a projection is displayed as a link list comprising pairs of facets values and counts indicating how frequently that combination occurs (the link's weight). The link semantics can be changed by changing the node type used in the aggregation strategy, allowing users to create arbitrary projections of interest. For example, as Figures 4.5D-F illustrate, a repeated swipe on `Character` leads to a projected network shown as an edge list in Figure 4.5E. Changing the aggregate pivot, for instance, to `Topic`, will re-project the current sub-network, creating pairs of `Character` nodes if they are connected to the same topic. The edge list behaves analogously to the node list; items are sorted based on the link weights, and pairs of node attributes can be selected as constraints, i.e., filter predicates.

4.6 Applicability

Orchard's design facilitates domain-agnostic, casual exploratory workflows of multivariate heterogeneous networks. To demonstrate its broad applicability, we tested Orchard on a number different datasets, ranging from politics, social media, to sports and cooking. In this section, we briefly highlight three exploration examples and detail the steps of a fictional usage scenario using a movie dataset (also depicted in the supplementary video).

4.6.1 Usage Scenarios

Given a dataset containing political lobbying information of the city of Chicago [1], our users found answers to questions such as: who are prolific or unsuccessful congress members? Who do they work with? What are common topics in the bills they endorse? Who endorses similar bills? Is there bias in different committees?

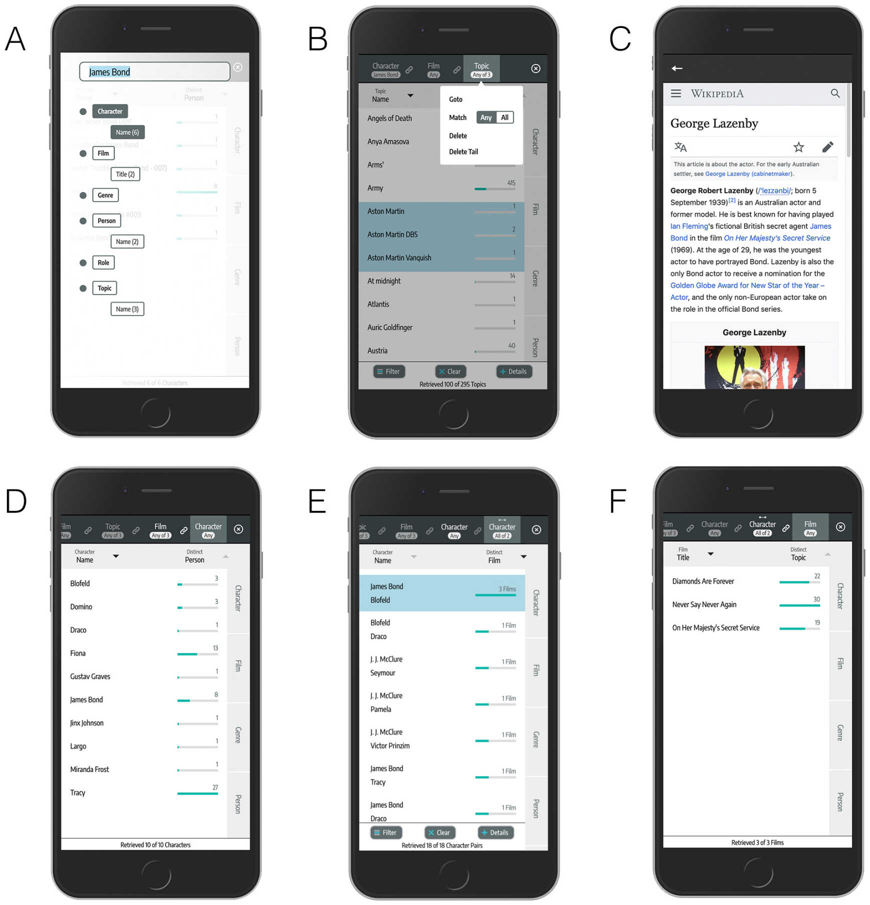
Using "likes"-data from Facebook [115], we set out to enable users to explore how different artists and public figures relate to companies, food, sport and TV shows. We annotated the data with additional information through public knowledge graph APIs. For example, we added descriptions such as *singer/songwriter* for Taylor Swift or *Fast-Food company* for McDonald's. Browsing this dataset, users were able to quickly identify unexpected and entertaining commonalities between celebrities.

Finally, from a publicly available soccer dataset [93] we extracted information about entities such as players, matches, leagues, and teams. Our users were interested in answering questions such as: which players have played against each other most frequently? How many teams have they played for? Have these players ever played on the same team? Which leagues transfer most players among each other?

4.6.2 Film Dataset Use Case

Emily - a film enthusiast - heard rumors about a new James Bond film being released soon. Knowing that Daniel Craig will be starring in the new Bond film, she is curious to learn more about other Bond films, associated actors and actresses, their roles, and so on. On her commute back from work Emily launches Orchard on her mobile phone and selects a dataset containing information about films and TV shows, such as title, release year, ratings, associated people, characters, and topics (see Figure 4.3 for more details).

What film characters are named "James Bond"? Emily first uses keyword search for "James Bond" and sees multiple potential facet types in which to search for a match (Figure 4.5A). She



Topic Name

Angels of Death	
Anya Amasova	
Arms'	
Army	415
Aston Martin	1
Aston Martin DB5	2
Aston Martin Vanquish	1
At midnight	14
Atlantis	1
Auric Goldfinger	1
Austria	40

Retrieved 100 of 295 Topics

George Lazenby

This article is about the actor. For the early Australian settler, see George Lazenby (cabinetmaker).

George Robert Lazenby (/ˈleɪzənbi/; born 5 September 1939)[2] is an Australian actor and former model. He is best known for having played Ian Fleming's fictional British secret agent James Bond in the film *On Her Majesty's Secret Service* (1969). At the age of 29, he was the youngest actor to have portrayed Bond. Lazenby is also the only Bond actor to receive a nomination for the Golden Globe Award for New Star of the Year – Actor, and the only non-European actor take on the role in the official Bond series.

George Lazenby

Character Name	Distinct Person
Blofeld	3
Domino	3
Draco	1
Fiona	13
Gustav Graves	1
James Bond	8
Jinx Johnson	1
Largo	1
Miranda Frost	1
Tracy	27

Retrieved 10 of 10 Characters

Character Name	Distinct Film
James Bond / Blofeld	3 Films
Blofeld / Draco	1 Film
J. J. McClure / Seymour	1 Film
J. J. McClure / Pamela	1 Film
J. J. McClure / Victor Prinzim	1 Film
James Bond / Tracy	1 Film
James Bond / Draco	1 Film

Retrieved 18 of 18 Character Pairs

Film Title	Distinct Topic
Diamonds Are Forever	22
Never Say Never Again	30
On Her Majesty's Secret Service	19

Retrieved 3 of 3 Films

Figure 4.5: Different views in Orchard. (A) Keyword search with result listing; (B) The context menu of a pivot node in the pivot trail; (C) An example of customizable detail view for a specific node, in this case the Wikipedia page of an actor; (D,E) Shows an example of a reflective pivot: swiping from `Character` to `Character` projects a subset of network nodes shown in (D) to a homogeneous `Character` $\leftrightarrow$ `Character` network (E). The result of this operation is a list showing a selected attribute (character name) of pairs of nodes (E). Each pair shares one or more linked nodes of a selected type (`Film` in this case). Selecting a pair of nodes in this view is equivalent to selecting both nodes individually such that they can be used as facets for subsequent views (F).

decides to match against characters named “James Bond,” and then find all associated films. To do so, she selects the `Character` node type from the search results (Figure 4.5A) and picks the exact match, which also has the greatest number of associated `Film` node types (Figure 4.4). To pivot to all films associated with the James Bond character, she touches the `Film` tab on the right and swipes left to reveal the results.

Which James Bond film has the highest rating? Emily changes the aggregation strategy, setting it to `Film.Rating`. She then sorts the results in descending order by tapping the sort icon next to the aggregate arrow next to the aggregation strategy.

What topics are associated with James Bond films? Curious about the topics associated with any of these films, she swipes left from the `Topic` tab. While scrolling through the list of James Bond film topics, she finds that there are three different kinds of “Aston Martin” topics, with 1, 2, and 1 linked films, respectively (Figure 4.5B). But how many of them are James Bond films? She swipes left from the `Film` tab to see all films associated with these topics. She then taps the link icon next to the prior “James Bond” `Character` selection to add this facet to the current filter. As a result, all non-Bond films are removed from the results list.

Which actors played James Bond in these films? Emily swipes left again from `Person` to get a list of actors. She knows Pierce Brosnan, Roger Moore, and Sean Connery, but who is George Lazenby? To find out she taps on the “Details” button at the bottom, which takes her to a detail view, in this case George Lazenby’s Wikipedia page (Figure 4.5C).

Which Bond characters frequently appear together? Emily goes back to the previous view by swiping right, and then swipes left from `Character` to see all characters associated with Bond films (Figure 4.5D). To learn which of these characters frequently co-occur in the same film, she swipes left from `Character` again to project the current heterogeneous sub-network (`Character` $\leftrightarrow$ `Film`) to a homogeneous network (`Character` $\leftrightarrow$ `Character`). She finds that James Bond and Blofeld are the most frequently occurring pair, co-occurring in three different Bond films (Figure 4.5F).

4.7 Evaluation

To assess the utility of Orchard, to learn how quickly users are able to grasp how to formulate common network queries, and to uncover potential usability issues in Orchard, we conducted a lab study with 12 participants (4 females). All participants were graduate students and all except one reported to be at least “slightly interested” in films and TV shows.

ID	User Study Task	Low-Level Tasks
T1	In which year was the film "The Shawshank Redemption" released"?	L1, L2
T2	Which people were involved with this film?	L3, L7, L8
T3	Of the people involved with this film, who contributed most to other films?	L4, L5, L6
T4	Of the people involved with this film, who contributed to the most distinct genres?	L4, L5, L6
T5	How many writers contributed to the film?	L6, L8
T6	Who were these writers?	L3, L4, L5, L8
T7	Which other films have these writers worked on together?	L3, L4, L5
T8	Which two people associated with this film are the strongest collaborators?	L9, L10

Table 4.2: Eight tasks we asked used in our user study, and their corresponding low-level tasks (shown in Table 4.1).

4.7.1 Dataset and Tasks

We designed our study based on similar experiments carried out in prior work [156]. As most people have at least some familiarity with films or TV shows, we decided to create a custom movie dataset containing data from IMDB (www.imdb.com/interfaces) joined with with topics extracted using our own algorithms from film plots in the CMU Movie Summary Corpus [12]. We then developed a series of tasks for evaluating Orchard. The tasks are inspired by Pretorious et al.'s task taxonomy for multivariate network analysis [110] which is based on work by Lee et al. [83] and Amar et al. [6]. We considered only those tasks that are equally applicable to homogeneous and heterogeneous networks alike (Table 4.2).

4.7.2 Procedure

We began with a five-minute demonstration of the features of Orchard by walking the participants through a scenario similar to the introductory use case described in Section 4.6.2. The demonstration included knowledge required to accomplish the eight tasks we created for this evaluation. Following the introduction we handed the mobile phone over to the participants and asked them to repeat the same or a similar sequence of steps. The participants were encouraged to ask questions during this initial phase. In the second part of the study we instructed participants to complete eight tasks we prepared (Table 4.2). They were given two minutes to solve each task without any help. If they were not able to find an answer in two minutes, we showed them how to get to the solution. In the third part of the study we asked participants to freely explore the dataset. During this process, participants were encouraged to think-aloud, i.e., to narrate their actions, intentions, and reactions. Finally, we gave participants a questionnaire and conducted semi-structured interviews to collect their feedback. Each session lasted approximately 50 minutes.

4.7.3 Results

In the second part of the study the majority of our participants were able to complete all eight tasks quickly (Figure 4.6), given that they had only gone through approximately ten minutes of introduction and training that covered a total of 14 operations to prepare them for the tasks. More than a half (56.3%) of all tasks were completed in less than 30 seconds, and 21.8% in less than 10 seconds. In only 14.6% of all cases, participants exceeded the two-minute limit and were offered assistance. In T1, for instance, some participants struggled to find the keyword search or the button to get to the details of the selected film(s). In T3, some participants forgot how to change the aggregation strategy to configure what is being shown in the bar chart. However, once they found out they were able to complete a similar task much faster (T4). A task of similar complexity was T5, which required participants to use and configure three pivot nodes, as well as the aggregation strategy. Out of all twelve participants, only one was unable to correctly apply facets such that the count of writers was restricted in the manner directed. Similarly, only one participant needed assistance in T6 because he did not remember he could swipe left on person to see the people's names associated with a selected role. In T7, some participants forgot how to change the boolean operator from matching any selections, to match all selections. Finally, most participants did not immediately recall how to create pair-wise combinations, but were able to figure it out in less than a minute.

In the third part of the study we asked participants to freely explore the dataset. We made the following key observations:

- *Intuitive swiping:* Participants were immediately comfortable navigating back/forth by swiping left or right. Swiping on a related entity to pivot was reported to feel natural and intuitive.

- *Long pivot trails*: The number of nodes in a pivot trail during exploration can grow rapidly. Various participants pointed out that it would be helpful to be able to store intermediate results by collapsing a (sub)-trail into a named pivot node.

- *Difficulty in distinguishing between lateral pivoting and drill-down filtering:* Although most participants were able to understand the semantics of Orchard quickly through trial-and-error,

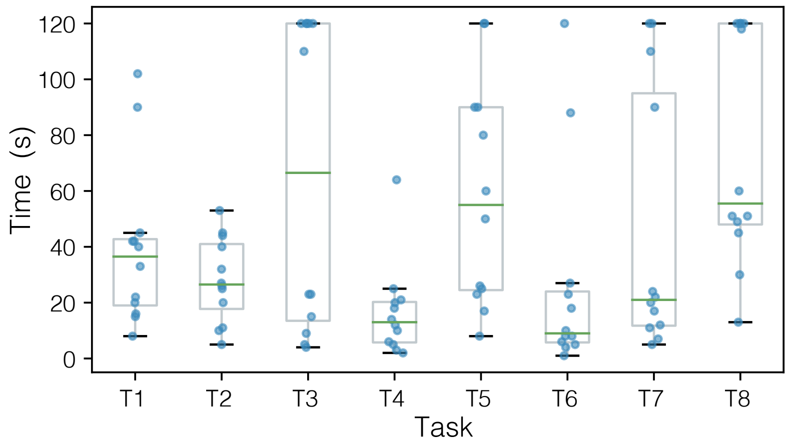


Figure 4.6: Task completion time for each task.

we observed that it was not immediately obvious to participants which facets were applied to the current view, i.e., how activating links in the trail affect the result in the current view.

- *Opportunity in exposing the computational path on demand*: While our query model hides the complexity of queries that involve multiple hops, some participants mentioned that exposing the computational path on demand would help users gain a better understanding of the network's topology.

- *Configurable defaults*: In some cases users would have preferred different defaults (e.g., the default aggregation strategy per node type, the order of node types in the swiper).

When completing a questionnaire of five questions using a 7-point Likert scale (1: "Strongly disagree" to 7: "Strongly agree"; Figure 4.7), despite the minimal training, most participants indicated that Orchard was easy to learn and use (Q1 and Q2). Participants also indicated that Orchard helped them find new and re-discover previously known insights (Q3 and Q4), and that swiping aided orientation while navigating through the network (Q5).

In addition, we overall received encouraging feedback: *"Once I got used to the defaults it started to become very intuitive."*; *"I liked the trails because it was very visual, you can think about everything as a line, everything is like a chain."*; *"Swiping to navigate felt very intuitive."*; *"Exploration was easy, you get where you want to."*; and *"I see this broadly applied to other datasets."*

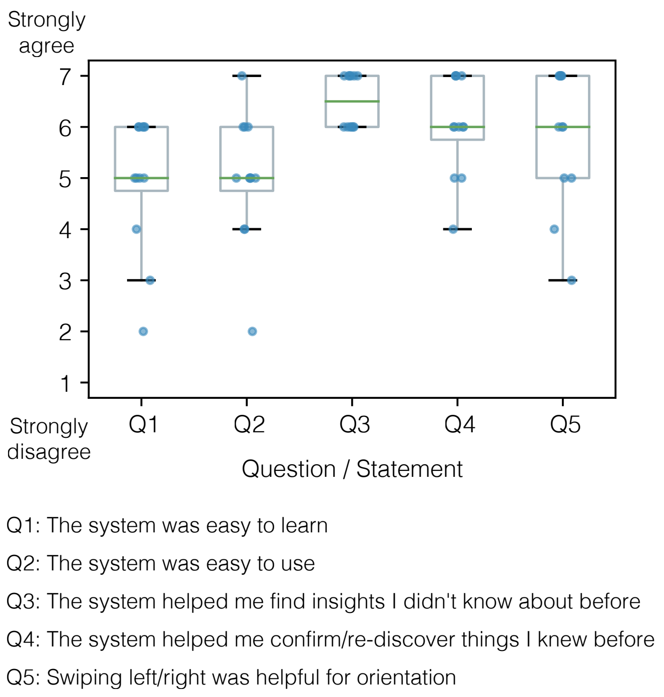


Figure 4.7: Results of our post-study questionnaire.

4.8 Future Work

While we have received encouraging feedback, there are a number of intriguing opportunities for extensions and improvements. Orchard currently supports on-demand network projections for any pair of node types, for any sub-network. Although our prototype shows weighted links for any combination of two nodes aggregates, we believe that users would benefit from a tighter integration with homogeneous projections. More specifically, Orchard could extract entire groups of nodes from projected (sub-)networks, i.e., by computing clusters/communities, connected components, etc. Homogeneous projections also open up new possibilities to add meta-data to nodes that could be used for actions like sorting. For instance, users could sort a list of nodes by common metrics used in network analysis, such as degree- or betweenness centrality, rather than just an aggregated facet value or facet count. However, blending-in concepts from homogeneous networks in workflows for exploring heterogeneous networks also poses challenges: how can we better communicate what projected networks represent, and how can we convey what complex networks metrics indicate in a way that non-expert users can understand? These questions would be useful to pursue in future design iterations.

4.9 Conclusion

We have presented Orchard, a mobile application that uses a novel query model to facilitate a simple user interface design to explore multivariate heterogeneous networks. Using our approach, minimal user input is required to formulate common network queries. With Orchard, users can fluidly navigate through multivariate heterogeneous networks, using gesture-based mechanisms to create facet trails, to pivot to entities of interest, and to collect and apply facets as part of an exploratory workflow. Given the ubiquity of information networks and the encouraging feedback we received from experts and lay users, we see great potential in providing a mobile solution for casual browsing of richly associated datasets.

Chapter 5

Discrete Time Specifications In Temporal Queries

5.1 Abstract

Analysis, exploration, and visualization of time-oriented data are ubiquitous tasks in various application domains, all of which involve the execution of temporal queries. Prior research in interactively specifying the time component for such queries has been focused on defining temporal relationships in data, i.e., querying event sequences through ordinal patterns. However, there has been much less emphasis on how to specify time as a quantitative data dimension in temporal queries. Motivated by the advent of the *Internet of Things* (IoT), we present a formal model that can be used to represent complex time specifications. Our model is the first step in an effort to enhance temporal user interfaces that enables discrete time specifications through a visual query interface.

5.2 Introduction

Various research efforts have identified and formulated both goals and tasks for information visualization, many of which include *interaction* in addition to visual tasks. Shneiderman [123], for instance, defined a task by data type taxonomy that comprises interaction tasks such as zoom, filter, and extract. Similarly, Yi et al [150] derived categories of interaction tasks, including "show me a different arrangement," "show me a different representation," and "show me something conditionally." While many interactive data exploration systems [34, 43] have built features to support these kinds of tasks for various data types, most offer only limited support for performing such tasks on time-oriented data.

An exception to this observation is the large body of work in the area of event sequence analysis, which has produced user interfaces that overcome the limited expressiveness of ordinal time specification techniques (e.g., [151, 98, 99, 148, 57]). However, such systems mainly focus on ordinal time, i.e., the relationship between events. Other applications treat discrete time as a regular, linear, and quantitative data dimension, where interactive parametrization is often limited to defining the start and end point of a time interval. Motivated by an IoT scenario, we argue that such systems are not suitable to express queries that exceed the complexity of simple interval- or calendar-based filters, since they either lack expressiveness or have a steep learning curve due to dialog-heavy interfaces or complex query languages.

As part of our endeavor to create a gestural interface for specifying time for temporal queries while also encouraging iterative and fluid exploration, this paper investigates a compact and expressive structure to parametrize discrete time using the basic time building blocks *instants*, *intervals*, and *spans*.

5.3 Motivating Examples

Our work is motivated by a recent experiment we conducted in the computer science building at Brown University. Throughout the building, we installed a number of IoT devices that record measurements and events, such as the current energy consumption of printers and vending machines, as well as motion in classrooms and when doors are opened or closed. An example of recorded data can be found in Table 5.1. In the following, we provide use cases to illustrate the importance of temporal interfaces that enable users to specify complex queries.

5.3.1 Example 1

The facilities department is responsible for management, maintenance, and scheduling of the building. In regular intervals, a facilities manager opens a web-based monitoring application, which displays information for all motion and temperature sensors as well as energy monitoring devices installed in the building. He is interested in finding anomalies in the recorded temperatures of all rooms in the building. While the application displays a linearly re-sampled representation of all temperature time series so they visually fit on the screen, he wants to zoom into last week's records in order to look at a more fine-grained resolution and detect unusual patterns.

Time	**Type**	**Value**	**Dev Id**	**Dev Type**
1483030839180	open	NULL	d1:front	door
1483030842534	watts	669	v1:coke	outlet
1483030843573	closed	NULL	d2:front	door
1483030844573	watts	669.7	v1:coke	outlet
1483030852564	watts	668.7	v1:coke	outlet
1483030853525	watts	669.2	v1:coke	outlet
1483030857530	watts	668.2	v1:coke	outlet
1483030858033	open	NULL	d1:front	door
1483030858549	watts	669.1	v1:coke	outlet
1483030861628	watts	668.4	v1:coke	outlet
1483030863981	closed	NULL	d1:front	door
1483030866552	watts	669.4	v1:coke	outlet
1483030867478	watts	668.8	v1:coke	outlet
1483030873516	watts	667.7	v1:coke	outlet
1483030874589	watts	668.2	v1:coke	outlet
1483030876625	watts	667.7	v1:coke	outlet
1483030878471	watts	666.4	v1:coke	outlet

Table 5.1: A sample from our IoT dataset showing measurements made for two doors (d1,d2) and a vending machine (v1).

5.3.2 Example 2

The facilities manager notices that the energy consumption of one of two co-located vending machines has remained steady over the course of the past 24 hours, indicating that no sales have been recorded. Since this is abnormal, he then decides to manually examine the machine and determines that there is indeed a technical issue. Because repair work on either vending machine can disrupt the service of the other, he wants to find suitable time slots for scheduling a repairman at times where the working vending machine is used the least. Knowing that repair work typically takes a maximum of two hours, he wants to find times on workdays between 9am and 3pm with low vending machine usage (i.e., low energy consumption). Based on the visualization shown in Figure 5.1, he schedules an appointment with the repairman.

5.4 Background

The goal of our work is to provide a formal description of a structure that can be used to parameterize time as an argument for temporal queries. We first recapitulate important time modeling design aspects that are typically considered when visualizing time-oriented data. We then give a brief description of a few selected systems that support data selection by time, before formally defining our model in the next section.

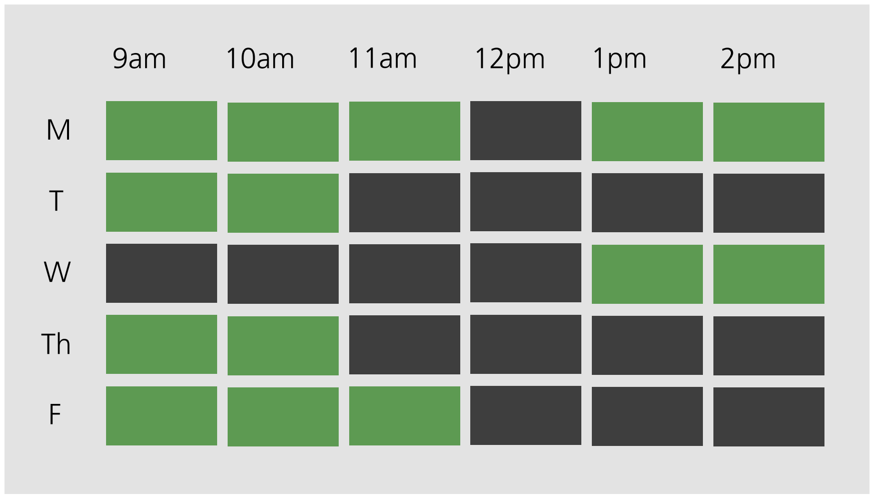


Figure 5.1: A sample visualization where a facilities manager executed a query to find time slots of two hours in length where the usage of a vending machine is below some threshold (shown in green).

5.4.1 Time Modeling

A wide variety of models have been proposed in past research. In the following, we provide a brief overview (based on [7] and [56]) of time-related objects and the properties most relevant to our work.

Time Domain Time can be categorized into either an ordinal, discrete, or continuous domain. Ordinal time only defines relative order relations, such as before and after. Continuous time most closely models physical time, where an infinite number of points in time exist between any other two points. Discrete time uses a discrete time unit (e.g., a UNIX timestamp denoting the number of elapsed seconds since January 1, 1970).

Time Primitives Time Primitives are the building blocks of a time model. We distinguish between *instants*, *intervals*, and *spans*. An instant refers to a point in time with finite precision in the discrete time domain and infinite precision in the continuous time domain. We define an interval as a closed interval between two instants. Both instants and intervals are anchored, meaning that they have a fixed position in time. Spans represent a duration of time (e.g., three seconds) and are, therefore, unanchored.

Granularity Time can be represented at different levels of granularity. The granularity of time determines how temporal primitives are transformed to other conceptual units of time. For instance, the Gregorian calendaric elements Saturday and Sunday can be summarized as a weekend, or a day can be represented as 24 hours.

Rind et al. presented similar work on a software library called TimeBench [114], that provides data structures and algorithms for visual analytics of time-oriented data. In particular, they developed a data model that captures the complexity of time-oriented data by mapping data objects to a set of time primitives and vice-versa. Our work, in turn, focuses on a more formal and expressive notation for time specifications that parametrize time as an argument for temporal queries.

5.4.2 Time Specification Techniques

A common technique to drill down in time series data (used by systems such as ChronoLenses [155] and KronoMiner [154]) is to select regions of interest using rubber band selections. In both systems, selected time ranges can be shifted using a drag-and-drop gesture. With KronoMiner, start and end values can be adjusted precisely using a pop-up dialog. TimeSearcher [62] uses a concept called timeboxes, which are similarl to the rubber band selection. Timeboxes are rectangles that can be drawn directly on a line graph visualization of a time series. While the extent of a timebox along the horizontal axis defines the time range of interest, the location on the vertical axis and height of the rectangle specifies the desired amplitude range. Queries can be refined in either dimension using range sliders. A more recent example of an application that allows users to explore time-oriented data and offers similar basic operations is LiveRAC [94], a system tailored to the visual analysis of large collections of system management time series. Tableau [130] offers a way to create parametrization controls, which can be applied as filters. A parameter (e.g., a start date) can be defined with a minimum and maximum value as well as a step size. An interactive slider control can then be used to set the parameter.

5.5 Time Specifications

In this section, we introduce a model that can be used to parametrize temporal queries, such as the ones outlined in the previous section. Because time in information technology is usually represented in the discrete domain, we will assume discrete time for all primitives.

We first describe a time-oriented query function Q that we aim to parametrize.

$$Q(\{t_{s0}, ..., t_{sn}\}, \lambda, d) \tag{5.1}$$

Queries on time-oriented data are typically defined by temporal and non-temporal parameters, $\{t_{s0}, ..., t_{sn}\}$ and λ, respectively, as well as the data d for which the query is defined. We call each t_s a *Time Specification.* A *Time Specification* is a set of *Time Primitives* that each can represent an *instant*, *interval*, or *span*, as we will show. More formally, we define a *Time Primitive* t_p as a tuple $\langle S, \delta, \sigma \rangle$, where the start time S and duration δ represent a time interval $[S, S + \delta)$. σ indicates a stride or time step that we will use to support temporal partitioning. All values in t_s are defined in a time unit (e.g, milliseconds or ticks). Typically, S is defined in $\mathbb{N}$, δ is in $\mathbb{Z}$, and σ can be set to either a value in $\mathbb{N}^+$ or to ∞. The non-temporal argument λ is defined as a function that takes in a finite set of data satisfying the temporal constraint given by all t_s. This could be an aggregate function, such as an average for a particular data attribute. Conceptually, Q first partitions the data based on the temporal specifications given by t and then applies λ to each partition.

With t_p, an instant can expressed by setting S to the timestamp of the *instant*, the duration δ to zero, and the stride σ to ∞. To describe *intervals*, we set S to the start time, δ to the difference between the start and end times, and σ to ∞. By definition, spans are unanchored, i.e., they are not specified by a start time and end time, but rather only by duration. However, spans are typically used to parametrize windowed functions in order to extract intervals of a given span duration (time span) at discrete steps (temporal partitioning). This is reflected in our model by the stride parameter σ. Setting a stride to infinity, as done for instants and intervals, means that λ is only applied to a single partition of data (an instant or interval). Conversely, setting a stride to a positive integer σ, can lead to one or many potentially overlapping partitions, which can be seen as defining multiple Time Primitives at once, each of duration δ but with varying start times. We formally define t_p as

$$\langle S, \delta, \sigma \rangle = \bigcup_{i=0}^{n-1} \langle S + i \cdot \sigma, \delta, \infty \rangle \tag{5.2}$$

where

$$n = \begin{cases} \text{floor}(\frac{\delta}{\sigma}), & \text{if } \sigma \neq \infty \\ 1, & \text{otherwise} \end{cases} \tag{5.3}$$

To illustrate the applicability of *Time Specifications*, we provide a number of examples that show how both simple and more complex time constraints can be represented using our model. For simplicity and readability, we do not use a particular time unit in the following specifications.

5.5.1 Examples

11am, January 1st, 2017

$$\downarrow$$

$$\{\langle \text{11am, January 1st, 2017, } 0, \infty \rangle\}$$

A single instant can be represented by a *Time Specification* with a single time primitive.

January 1st and January 3rd, 2017

$$\downarrow$$

$$\{\langle \text{January 1st, 2017, } 0, \infty\rangle\}$$
$$\cup$$
$$\{\langle \text{January 3rd, 2017, } 0, \infty\rangle\}$$

Consequently, n non-consecutive instants are represented by n time primitives.

10am to 2pm, February 20st, 2017

$$\downarrow$$

$$\{\langle \text{February 20st 10am, 4h}, \infty\rangle\}$$

An single interval is represented by setting the duration. Analogous to the previous example, non-consecutive intervals can be expressed by defining multiple time primitives.

Past hour

$$\downarrow$$

$$\{\langle \text{Current time, -1h}, \infty\rangle\}$$

Negative durations can be used to specify an interval from $S - \delta$ to S.

All weekends

$$\downarrow$$

$$\{\langle \text{First Saturday 1970 in Unix Time, 2d, 7d}\rangle\}$$

A span is defined by setting the stride to a positive integer, in this case seven days. Since "all weekends" does not constrain the time range in which weekend intervals are to be queried, the start time of this specification is set to the first occurrence of a Saturday in the given time unit.

All 24h windows with stride of 1h in the past week

$$\downarrow$$

$$\{\langle \text{Current Time, -7d}, \infty\rangle\}$$
$$\cap$$
$$\{\langle \text{First Unix day, 24h, 1h}\rangle\ \}$$

In this example, the user is looking for a span (24h) within a certain interval (past week). Since the query has to satisfy each of the time constraints imposed by a time specification, we can limit the temporal search space for a span by adding an additional interval primitive to the time specification.

5.6 Discussion and Future Work

Time Specifications compactly describe discrete time instants and intervals as well as periodic intervals and can easily be translated into queries over data (e.g., using SQL). Their expressiveness has been particularly useful to formulate sample queries, which we used to determine and compare

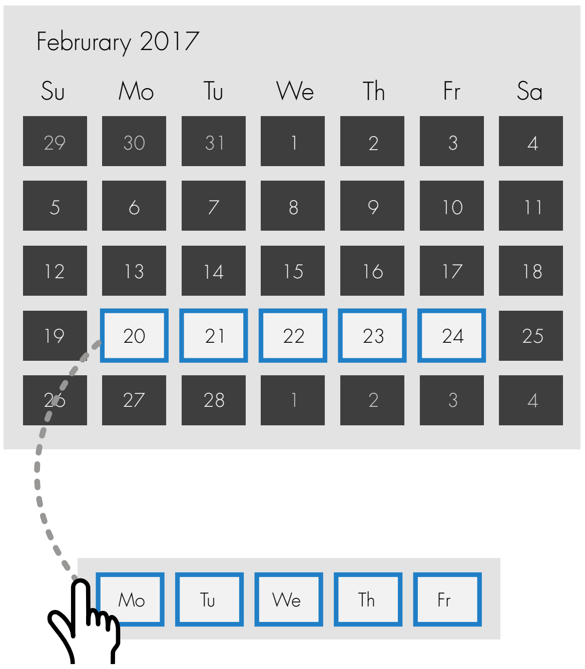


Figure 5.2: Depicts a possible scenario of how a time primitive can be turned into a new time specification. In this case, an interval (Mo, Feb 20 to Fr, Feb 24) is dragged out of its original context (the calendar) and thereby converted to *all weekdays* (query-by-example).

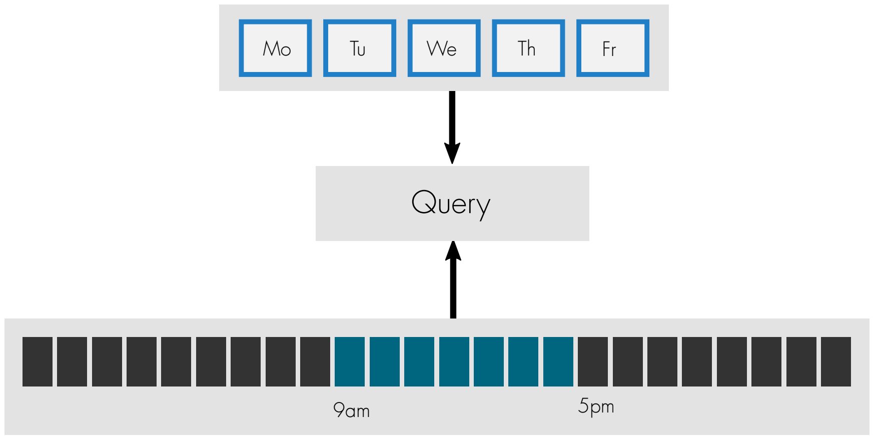


Figure 5.3: Shows two time specifications (Mo-Fr and 9am-5pm) that fully specify the temporal argument of a query.

the capabilities of existing interactive data exploration systems that support time-based data. Consider our motivational examples: a possible time specification for *last week* in Example 1 could be $\langle$Current time, -7d, $\infty\rangle$. The time constraints in Example 2 could be described by first specifying weekdays as $\langle$First Monday in Unix Time, 5d, 7d$\rangle$ and then intersecting it with the daily time range $\langle$First Unix day 9am, 8h, 1d$\rangle$.

Although our model makes it easy to reason about and express time-dependant queries, there are many open questions that must be answered so that users can easily and naturally express queries in a visual interface. While we showed that a wide set of time-based constraints can be specified using just three parameters, a remaining challenge is to determine how users can define the parameters to construct a time specification in a user interface, and which operations they can use to derive parameters from existing specifications. A key aspect thereof is the notion of granularity, since humans use abstractions when they talk about time. The parameters in the example time specifications shown in the previous section are defined in different time units such as hours, days, weekends, and years. Although many systems provide mappings from higher level granularities to a discrete time domain (e.g., Unix time and vice versa), many support only a fixed set of pre-defined granularities, as Rind et al. [114] point out. While useful in some some cases, we believe that an expressive visual query language should facilitate intuitive ways to define, modify, and derive the parameters used in our model at both pre-defined as well as custom granularities. Figure 5.2 schematically depicts a case where an anchored interval can be turned into a span by moving it out of its original context.

Based on this work, we are currently designing a novel gestural interface that allows for incremental specifications of time-oriented queries. We envision an iterative query building workflow that resembles the iterative build-up of formal specifications. As described in the previous paragraph, a user with the intention to constrain a temporal query to hours 9am - 5pm on weekdays would split up the query into (1) specifying weekdays, (2) specifying the hours, and (3) connecting the two specifications to fully define the temporal arguments of the query. Figure 5.3 schematically illustrates such a scenario.

Chapter 6

Forecasting for Non-Experts

6.1 Abstract

Time series forecasting is a common data science procedure used for an abundance of tasks such as inventory and staff planning, prediction of financial assets, projecting electricity consumption, as well as anomaly detection. However, developing time series forecasting models is technically challenging and requires domain expertise. In this chapter, we present a system that facilitates an end-to-end, mix-initiative forecasting workflow for non-experts users. It exposes a set of tools that allow users to identify and explore time series in regular tabular datasets, and enables them to quickly specify and find solutions for common forecasting problems. Users can iteratively add and adjust parameters such as exogenous regressors, upper and lower bounds, as well important events that may impact the forecast. Furthermore, users can explore the effect of custom interventions, and inspect the impact of different variables on any predicted data point.

6.2 Introductory Use Case

Time series forecasting is complex and requires domain expertise to develop high-quality models. To provide some intuition of the complexity of the steps involved in an end-to-end time series forecasting workflow, consider the following fictional use case.

Emily is a crypto-currency enthusiast. To optimize her gains she wants to see if she can develop a price forecasting model that will help her optimize her trading strategy. She downloads a dataset containing historic real-time Ether prices. The dataset contains two variables: a UNIX timestamp, and the price of 1 Ether in US Dollars. As a first step, she must concisely define the problem she wants to solve. She decides that she wants to predict daily closing values, using a week worth of hourly closing prices. However, the raw data is not isochronous, i.e., does not occur at equal time intervals, and was recorded at a much finer grained resolution (multiple times per second). She therefore needs to prepare the data accordingly, which includes aggregating the data per hour and day using an aggregation function such as the average or median. She then decides to start off with a basic autogressive integrated moving average (ARIMA) model. As a trained statistician, Emily knows that ARIMA requires the input time series to be stationary (a time series that does not have trend or seasonal effects). Yet, given its recent upward trend and the obvious daily periodicity, Ether prices clearly exhibit non-stationary characteristics. To make it stationary, Emily decides to reformulate the problem. Rather than predicting the price, she now wants to predict the difference of the price at time t vs. $t-1$. She then performs a statistical test called Dickey-Fuller test to validate that the transformed time series is indeed stationary. To see how well her model performs, Emily decides to use a Walk-forward validation, i.e. to re-train the model after each prediction step, and to use root mean squared error RMSE as error metric. Based on her first model she decides to train a new one but using a different set of hyper-parameters. Success: the new model leads to a slightly lower RMSE. But is this model statistically superior than her first one? To find out Emily applies the Diebold-Mariano test statistic, which is commonly used to compare the forecast accuracy of two forecast methods. Finally, knowing that model 2 does not perform significantly better than model 1, Emily wants to train a new model, but this time with more input data than just historic price values. Specifically, she hypothesizes that adding prices of other crypto-currencies, as well as stock prices to the training set might lead to better forecasts. However, because ARIMA cannot be used for multivariate time series forecasting problems, she decides to use Vector Autoregression algorithms, Long-Short-Term-Memory (LSTM) models, etc.

6.3 Desiderata

6.3.1 Tasks

To assist non-scientists in scenarios such as the one described above, we break down the forecasting process into the following tasks:

- *Problem Definition:* What is the forecasting target? A single variable, multiple variable, single time steps, multiple time steps? What are alternate definitions of the problem?

- *Training data selection:* Which data should I use for my predictions? Historical values of univariate time series, or multivariate time series? Is the data in the right format, e.g., is it isochronous? With which datasets can I augment my training data?

- *Model Selection:* Which models are suitable candidates for my problem? Which models work well for my data size? Are there model-specific assumptions on the data?
- *Evaluation:* What is a reasonable train/test split? Which metric should I use? How should I interpret the metric? How can I determine which forecast is better?

Although some of these steps can be automated, the role of domain experts is crucial for incorporating useful assumptions and heuristics [133]. For instance, they can formulate and re-formulate forecasting problems to facilitate automated search for solutions. They can add hints to make it easier for AutoML components to find solutions faster. For instance, many time series exhibit trend changes, seasonality dependent value increases or decreases. Domain experts are able to articulate such phenomena in advance. They are also aware of calendar-based events that may affect the trajectory of a time series, such as holidays, product launches, political events, etc. Finally, they know what other variables might be a useful in predicting the variable they want to forecast, or can define reasonable lower and upper bounds for their predictions.

6.4 Exploring Time Series in Tabular Datasets

6.4.1 Time Series Characteristics

While there various different kinds of time series, such as high-frequency sensor readings, or low frequency census updates, in this chapter we particularly focus on what Tylor and Lethman refer to has *business time series* [133]; time series that are generated by human action. According to their analysis, such time series have a number of common characteristics. On one hand, they typically exhibit one or many seasonal effects, e.g., daily or weekly cycles. On the other hand such time series contain one or many segments with a visible trend. And finally, the often contain outliers.

6.4.2 Time Series Cardinality

In their simplest form, time series are series of tuples containing a time and a numerical value. Consider, for instance, Table 6.1, showing one-dimensional time series of total sales per month.

In most real-world use cases, however, a single table contains multiple series. For example, assume that a single table contains multiple numerical measurements, each of which has a number of tags such as the branch in which the sales was recorded, the sales person, product category, product name, etc. (Table 6.2). In database terms, the number of possible tag combinations is referred to

Date/Time	Total Sales
2019-01-01	266.0k
2019-02-01	145.9k
2019-03-01	183.1k
2019-04-01	119.3k
2019-05-01	180.3k
2019-06-01	168.5k
2019-07-01	231.8k
2019-08-01	224.5k
2019-09-01	192.8k
2019-10-01	224.5k
2019-11-01	192.8k
2019-12-01	165.9k

Table 6.1: An example time series containing the total sales per month in 2019.

Date/Time	Price	Branch	Employee ID	Product ID
2019-01-01 8:02:21	3.50	Boston	D291	P201
2019-01-01 8:02:21	1.10	New York	D509	P371
2019-01-01 8:02:22	34.30	Atlanta	D008	P746
2019-01-01 8:02:47	8.70	Boston	D048	P847
2019-01-01 8:02:47	9.00	Boston	D048	P746
2019-01-01 8:02:48	1.20	Atlanta	D834	P371
2019-01-01 8:02:49	2.90	Atlanta	D732	P201
2019-01-01 8:03:11	2.45	Atlanta	D981	P913
2019-01-01 8:03:13	7.65	New York	D416	P821
2019-01-01 8:03:14	3.05	Boston	D948	P991
2019-01-01 8:03:14	6.40	New York	D487	P334
2019-01-01 8:03:14	1.10	Boston	D691	P872
...	...	...	...	

Table 6.2: An example time series containing the total sales per month in 2019.

as the *series cardinality* of a dataset. However, the number of series is not limited by the tags. In many cases datasets grow so big that individual recordings become less important than *aggregates* thereof, such as a monthly sum as shown in Table 6.1. This means that the series cardinality of a single datatable is defined by the number of combinations of all tags, as well as all aggregation possibilities. Given the large space of different possibilities, it is key to provide tools that make it easy to isolate and explore these different series.

6.4.3 Exploring Time Series

In Northstar, a data science platform we built as part of a DARPA's D3m program, we added a set of tools facilitate the exploration of the many time series that may be present in a tabular dataset. When loading a new dataset, the system shows all attributes (columns) in the menu on the left

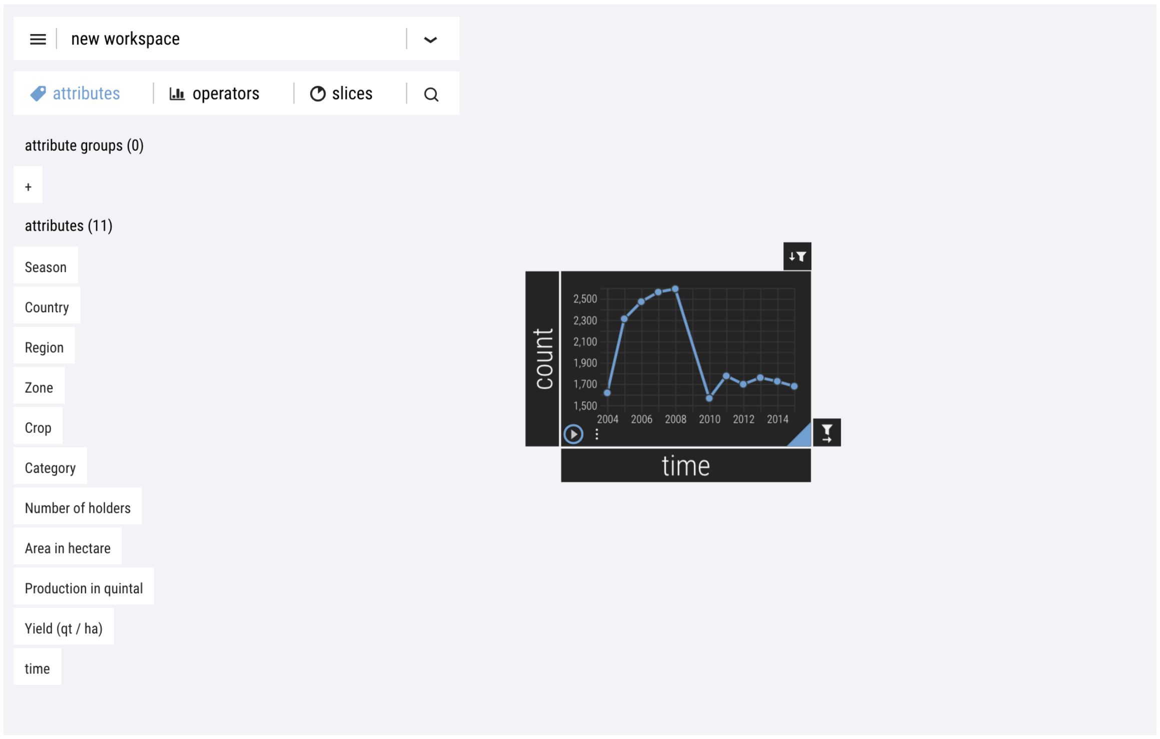


Figure 6.1: Shows the user interface of Northstar and a time series plot showing the number of records in the current dataset per year.

hand side (Figure 6.1). To visualize any time-based attribute, users can drag them out and drop them onto the workspace. Doing so adds a linegraph to the workspace, which, by default, depicts the number of records (rows) in the dataset for a given time step (year in the case of figure 6.1). To replace the count of rows with a different measurement, users can drag and drop any other numeric attribute on the y-axis (Figure 6.2). By default, the system chooses a suitable time granularity given the space constraints of the visualization. In Figure 6.2, for example, the system decided that aggregating the data on a per-month basis would lead to too many data points given the size of the visualization. Therefore, it uses a yearly aggregation level instead. The time granularity can be configured by the user and they can choose how data is aggregated across time by picking an aggregate function. For instance, in 6.2 a user might be interested in plotting the total, i.e. the sum of all crop production values, rather than an average or count.

In Northstar, any plot can be used as an interactive filter to perform visual linking and brushing. Figure 6.3, for instance, shows how the plot from Figure 6.2 can be constrained to only visualize the total production of “Cereals” in the “Amhara” region of Ethiopia: a user first drags out the “category” and “region” attribute, for which the system creates an interactive bar chart, by default. The user visually connects the two plots, and selects the relevant tags directly in the plot. Optionally,

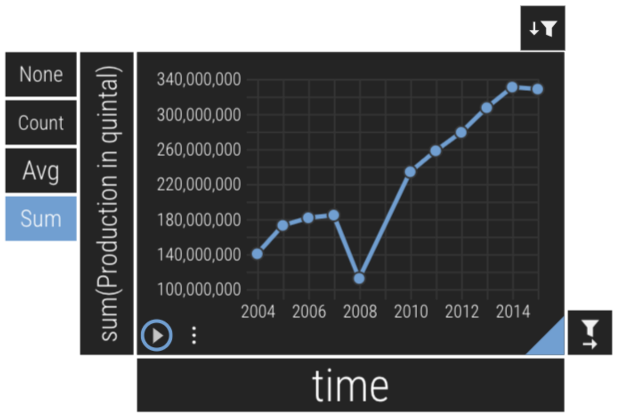


Figure 6.2: A time series plot showing how the total crop production in Ethiopia has developed over time. The aggregation function (sum, in this case) can be changed through the menu on the left.

the logical boolean operator that defines how these two sets are merged can be configured by tapping on the icon where the links meet.

Similarly, any time series plot can be "brushed", i.e. selected portions in surrounding plots are highlighted in different colors. In Figure 6.4, for instance, a user compares the production development for two different crop categories, cereals and root crops, which are highlighted with their respective color.

6.5 Forecasting

6.5.1 Univariate Forecast

The forecasting process works as follows: First, users use the exploration tools introduced above to identify a time series of interest. From the operators menu on the left, they drag and drop a forecast operator onto the workspace. To initiate a forecast, users can then connect the time series plot to the forecasting operator, making that time series the target of the operator. When pressing the play button in the bottom left, the system computes a univariate forecast based on all historical values.

Our system uses the algorithm proposed by Taylor and Letham [133], a decomposable time series model using the following three components: trend, seasonality, and (possibly irregular calendar-based events that may affect the time series and its forecast. The algorithm is a generative additive model defined as:

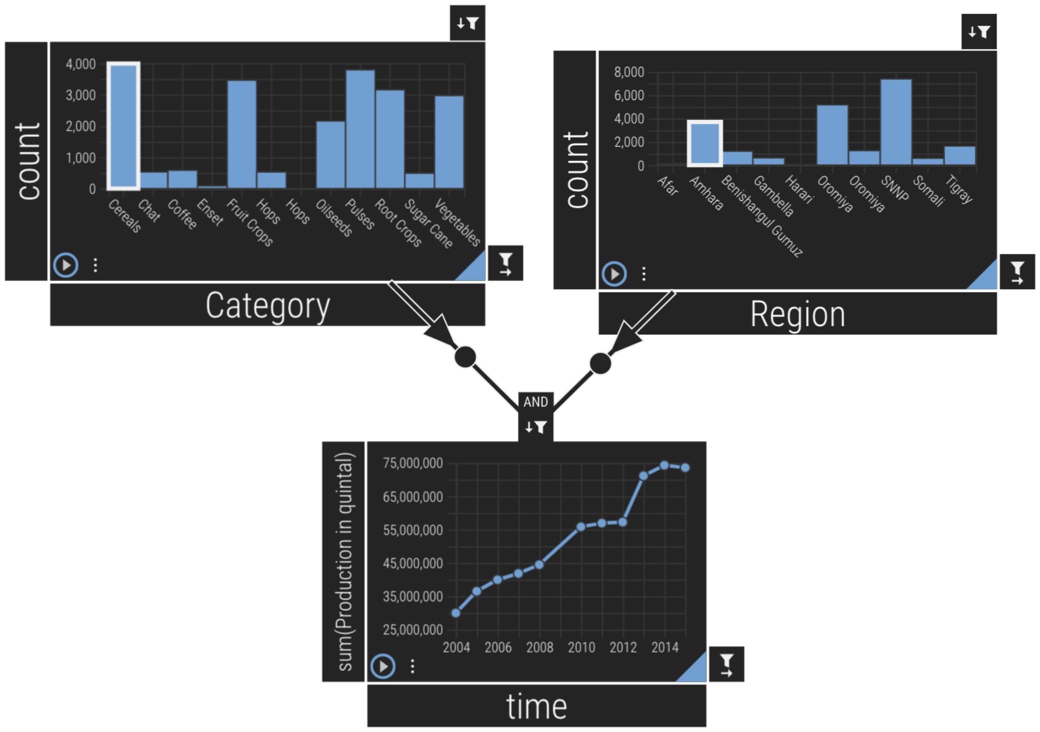


Figure 6.3: Users can isolate arbitrary time series in a dataset by applying filters to a time series plot. In this example, two tag constraints are applied to the time series at the bottom.

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t \tag{6.1}$$

where $g(t)$ models non-periodic trends, $s(t)$ models periodic changes, and $h(t)$ models any other, possibly irregular events such as holidays.

The results of the forecast are displayed as a continuation of the input time series (Figure 6.5. By default the operator predicts the next 15 data points, spaced by then currently selected time granularity (e.g., days, or months).

The horizon can be manually configured by the user; by dragging the vertical bars they can choose to perform a forecast in a out-of-sample or in-sample fashion, or a mix of the two.

Each prediction can be inspected by hovering over it. When doing so, a pop-up appears showing the predicted value as well as the decomposed trend and seasonal terms leading to this prediction.

Furthermore, to give users a sense of the confidence for each prediction, our system also shows the confidence boundaries in gray.

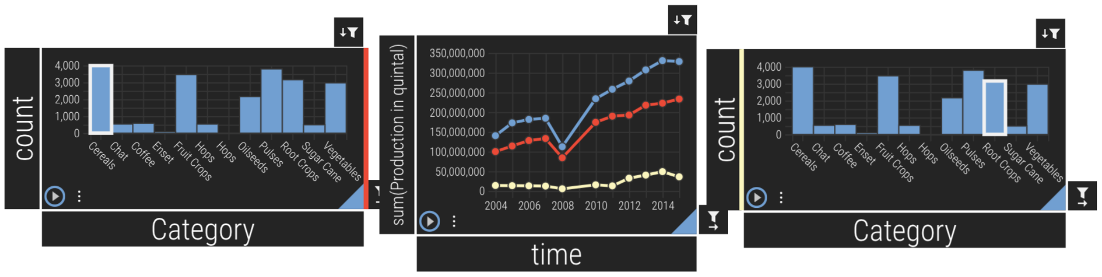


Figure 6.4: Time series can be brushed, i.e. highlighted with color, to put them in context. In this example, a user compares the development of total crop production levels of two crop categories.

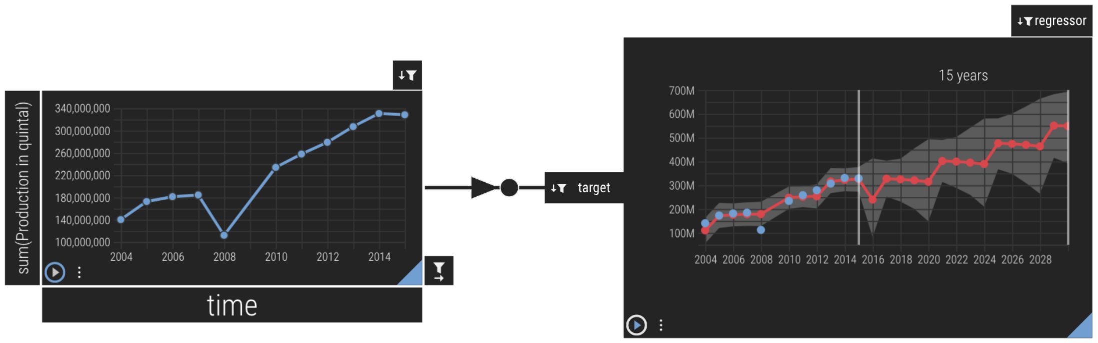


Figure 6.5: A univariate forecast of the average crop production levels in Ethiopia, with a horizon of 15 years

6.5.2 Multivariate Forecast

In many cases, forecasts can be improved by adding additional time series data that might be indicative of the progression of some other time series. Northstar allows users to add an arbitrary number of such independent variables, i.e. regressors through the user interface. To do so, users can define other time series using the steps described in the previous section. They can connect them to the "regressor" input and then re-run the operator (Figure 6.6. Formally, an additional regressor $r(t)$ extends the model introduced in the previous section as follows:

$$y(t) = g(t) + s(t) + h(t) + r(t) + \epsilon_t \tag{6.2}$$

This change has an important implication on the forecasting process: Predicting $y(t)$ requires a value of r at point t, which in it of itself is a forecasting problem. Our system resolves this prediction order problem by first performing a univariate forecast on r named $r\hat{(t)}$ it to solve for $y(t)$ instead:

$$y(t) = g(t) + s(t) + h(t) + \hat{r}(t) + \epsilon_t \tag{6.3}$$

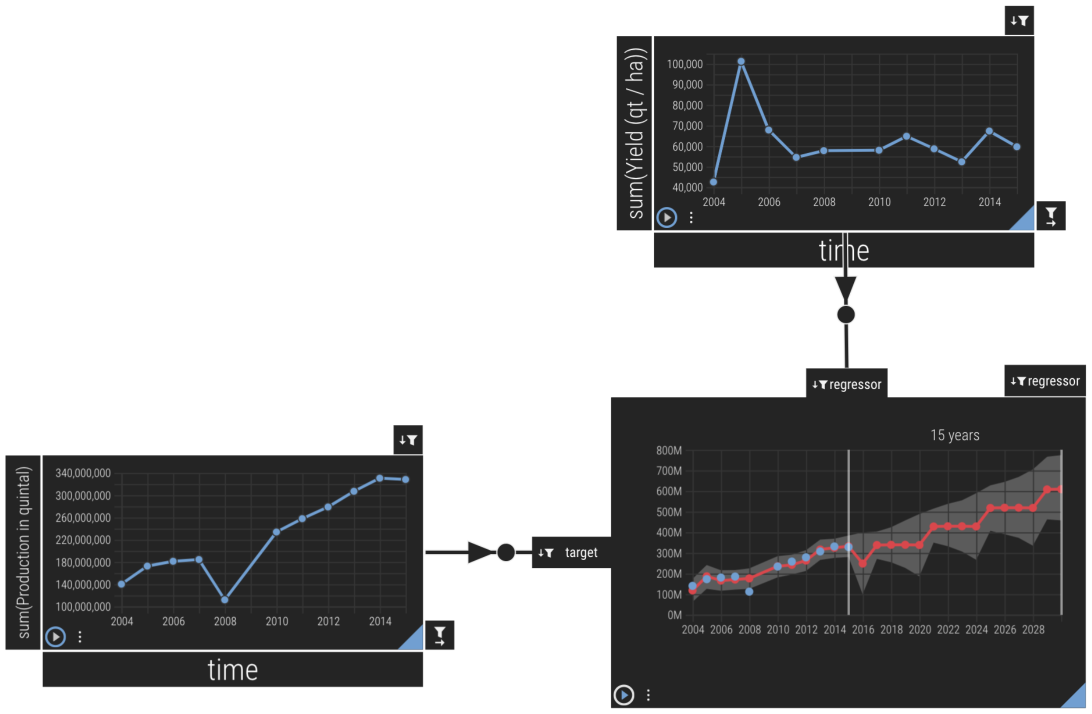


Figure 6.6: To improve a forecast users can add an exogenous regressors.

6.5.3 Visual Intervention Analysis

In many cases it is helpful for users to understand the effect of certain interventions. Imagine the crop dataset contained additional data such as weather information and fertilization recordings. Users might ask themselves: how would an increase in precipitation or decrease in the amount of applied fertilizer affect the yield and the production of crops? Similarly, when including exogenous regressors in a forecast, there is a chance that domain experts have access to a more accurate model than a simple univariate forecast on that regressor. To perform this kind of what-if analysis, users can directly modify the forecasts of any regressor by sketching (Figure 6.7.

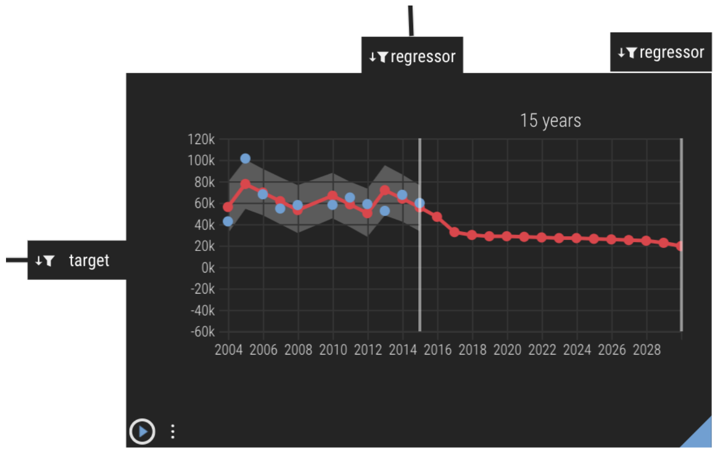


Figure 6.7: Shows the original data points (blue) and the model (red) for a regressor. In this case the user replaced the forecast for the regressor with their own hand-drawn progression of the time series (right-hand portion of the plot.

6.5.4 Related Work

A related system is TimeFork [11], which uses a mixed-initiative approach to perform time series forecasting using pre-trained models. In TimeFork users are presented with multiple possible outcomes of a prediction, allowing the users to iteratively perform a forecast with increasingly large horizons. TimeFork assumes that the time series and models used are pre-defined, i.e., it only supports users in the actual prediction step of an end-to-end forecasting workflow. Another system, TimOvA [19], is a visual analytics tool for time series model selection that was build to help domain experts build ARIMA and Seasonal ARIMA models by enabling them to tweak the parameters and inspect residual plots. However, TimOvA has several limitations: it only supports two basic statistical models (no machine learning algorithms), the skills required to manually optimize the parameters of these two models clearly exceed the ability of domain experts.

Chapter 7

IDEBench: A Benchmark for Interactive Data Exploration

7.1 IDEBench: A Benchmark for Interactive Data Exploration

n recent years, many query processing techniques have been developed to better support interactive data exploration (IDE) of large structured datasets. To evaluate and compare database engines in terms of how well they support such workloads, experimenters have mostly used self-designed evaluation procedures rather than established benchmarks. In this paper we argue that this is due to the fact that the workloads and metrics of popular analytical benchmarks such as TPC-H or TPC-DS were designed for traditional performance reporting scenarios, and do not capture distinctive IDE characteristics. Guided by the findings of several user studies we present a new benchmark called IDEBench, designed to evaluate database engines based on common IDE workflows and metrics that matter to the end-user. We demonstrate the applicability of IDEBench through a number of experiments with five different database engines, and present and discuss our findings.

7.2 Introduction

Interactive data exploration (IDE), i.e., user-guided exploration of a database is an important task in data analysis. IDE tools such as Tableau assist users in exploring databases by allowing them to filter, group, and aggregate data through a visual user interface. The speed of computation when performing such operations is crucial. A study by Liu et. al. [87] showed that latencies greater than 500ms can have a significant negative impact on user performance. Thus, relational DBMSs are challenged to provide responses at interactive speeds for ad-hoc queries that users issue as they explore data through a visual user interface. However, traditional analytical DBMSs, such as MonetDB [97] or SAP HANA [51], often take seconds or even minutes to compute results on increasingly large databases. To mitigate this problem various query processing techniques have been proposed for DBMSs [5, 86, 34, 35, 89, 65, 127]. Examples include re-using previously computed results [35, 54, 49, 32], employing specialized data structures, on-demand creation of stratified samples [35], or performing speculative pre-computation based on previous interactions [68].

As the space of such techniques grows rapidly, comparing and evaluating their utility in the context of IDE becomes increasingly important. Yet, the workload used in traditional analytical SQL benchmarks such as TPC-H [136], TPC-DS [135], or the Star Schema Benchmark (SSB) [105] are not representative of the unique characteristics of IDE, and their metrics are not primarily intended to capture what matters to the end-user, rendering them unsuitable for IDE benchmarking. Our analysis of IDE user behavior shows that workloads are typically composed of sequences of related queries that are separated by time gaps. Users incrementally refine filters and visualize subsets of data using multiple plots for different attributes and aggregate functions. Conversely, queries found in existing benchmark are largely independent and unrelated, and meant to be executed back-to-back. Furthermore, metrics used in existing benchmarks are unable to fully capture aspects that are important to the end-user. They do not take into account different execution models, such as *blocking* execution, where exact results are returned once all data is processed, and *approximate* and *progressive* models, which can be configured to balance the trade-off between result quality and query run-time. Moreover, the time a system takes to start up is largely neglected even though some engines heavily rely on pre-processing (e.g. to create samples offline).

We argue that an IDE benchmark must not be solely based on the end-to-end run-time of individual queries. Instead it should report on metrics that reflect that trade-off between speed and quality, as well as on the time it takes to pre-process data. Such metrics enable benchmark users and systems builders to answer questions like, “can a system provide responses within interactive latencies”, “at what cost in terms of quality of the result”, “at which data-size would a traditional column-store like MonetDB outperform an approximate engine?”, “how much overhead do approximate query processing techniques introduce?”, etc.

In this paper we present IDEBench, a new benchmark to facilitate comparison of database engines tested under common IDE conditions. The design of IDEBench is driven by the findings of five user studies we have conducted over the course of the past few years. First, contrary to existing benchmarks, IDEBench generates and uses time-spaced sequences of aggregate queries as its workload, which are representative of common query patterns we observed in the logs of our user studies. Our goal is not to accurately simulate users, which is arguably impossible, but instead to focus on a common denominator of IDE query patterns in query workloads that result from IDE systems. Second, IDEBench uses real-world data. This is crucial since reporting the quality of approximated results, i.e., accuracy and completeness, using synthetic uniformly distributed data, for instance, would be meaningless. To that end, we built a data generator that can be used to scale real-world datasets to any size while maintaining the original characteristics. Third, inspired by prior work of the data visualization community IDEBench reports on metrics that capture the trade-off between quality of results and query run-time. Finally, our benchmark is designed in an extensible way that enables users to add custom datasets and generate workflows that match their IDE use case.

In summary, the main contributions of this paper are: (1) We report on the results of an extensive workload analysis of five user studies. (2) We present the design of IDEBench, an extensible benchmark that facilitates the evaluation and comparison of different database engines for IDE workloads. The source-code of IDEBench is made available to the research community[1]. (3) We evaluated five different database engines using IDEBench and present and discuss their results: two commercial database engines, two research prototypes (IDEA, approXimateDB/XDB), as well as MonetDB.

7.3 Interactive Data Exploration

In this section we first describe an IDE scenario that exemplifies how users visually explore data using IDE tools such as Tableau and Power BI. We then define the scope of our benchmark and present a selection of DBMSs that are within this scope.

7.3.1 An IDE Example

Jean, a research staff member at a major hospital, wants to get an overview of the hospital's patient population and their health problems. To do so, she looks at electronic health records from the past 20 years. Jean starts out by examining demographic information of patients and, for example, finds that patients ages are normally distributed. She then continues to look for interesting patterns in admission times and dates. Jean creates a query that shows the number of new admits per hour of the day. The result reveals that most admissions are during business hours, but there is an interesting bump from 7 to 10 pm. She filters down to admits from the emergency center and

[1]http://github.com/IDEBench/IDEBench-public

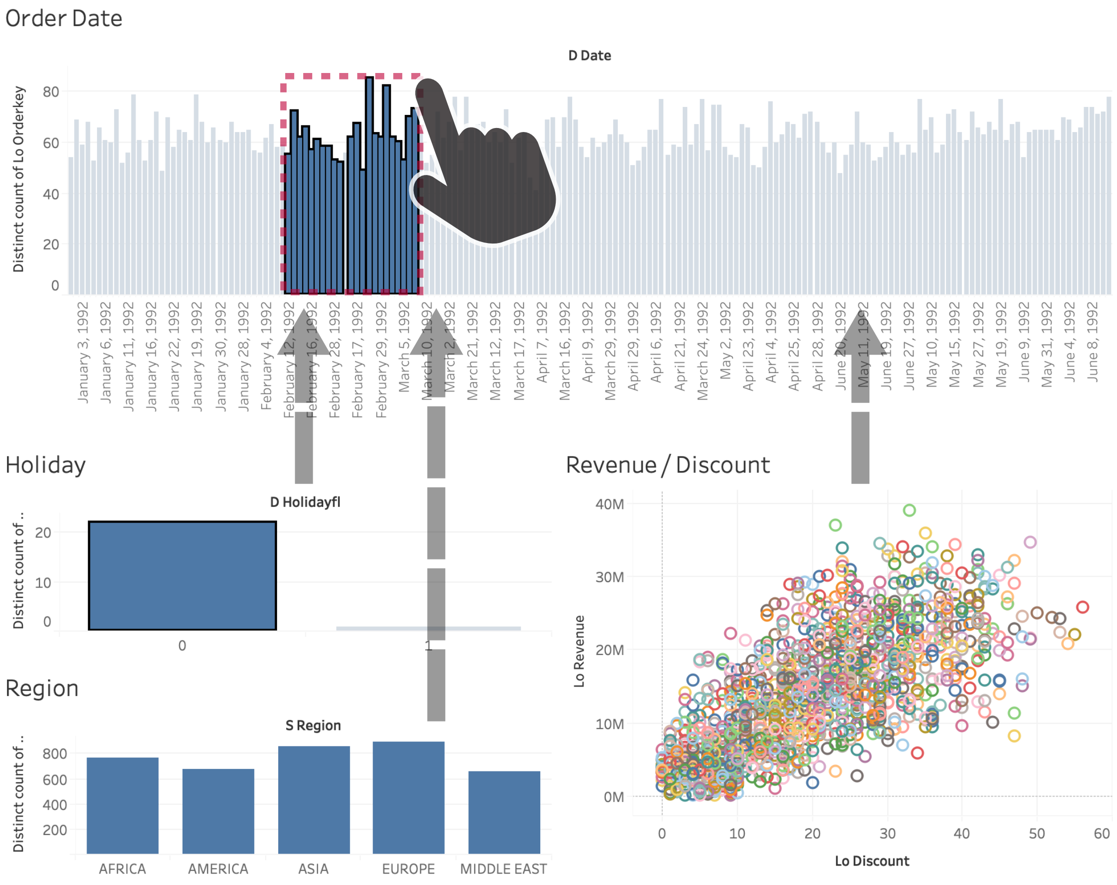


Figure 7.1: A dashboard created by one of our user study participants using Tableau. She laid out multiple visualization in a coordinated view (the dashboard), where all visualizations are implicitly linked. In this example she selects a range of data points using a mouse gesture, filtering the three linked visualizations (shown by the grey dashed arrows). This selection results in three concurrent database queries.

notices that most of the admissions between 7 and 10 pm were recorded there. Is this trend identical on all days of the week? She refines her query to display admits on weekends and sees that the bump shifted towards 10 to 12 pm. Who are these patients? Jean filters her previous age query by patients admitted on weekends between 10 and 12 pm. She finds that patients ranging from 20 to 35 are over-represented in this subset when compared to the overall age distribution. Now Jean wants to see which health problems are common among this sub-population. She finds that head traumas occur frequently and decides to check with the administration if the hospital's duty rota accommodates for this by making sure a trauma specialist is on call during weekend nights.

7.3.2 Scope of Benchmark

The above scenario illustrates a visual data exploration paradigm, which is commonly referred to as the *Visual Information Seeking Mantra*: "Overview first, zoom and filter, then details-on-demand" [123]. Modern IDE tools such as Tableau and Power BI are built to support this paradigm. On demand, users can create visualizations of attributes of interest, look at the distribution of associated values, and zoom into subsets of interest before inspecting specific instances.

Often visualizations can be turned into active filters; a technique known as *linking*. Most modern IDE tools can be configured to interpret selections of value ranges in a plot as filter that is subsequently applied to other plots presented to the user. As a result, visualizations can be used to conveniently slice and dice data, and to examine sub-populations.

Unlike traditional dashboards, in which all plots are pre-defined and (implicitly) linked [89], modern IDE tools like Tableau and Power BI allow users to create custom dashboards, where plots can be arbitrarily added, removed, modified and linked. Interactions such as creating, modifying and linking plots, as well as selecting value ranges on an existing plot (filtering), trigger database queries. For instance, a user selection on a plot that is linked to n other plots might results in n concurrent queries. This paper focuses on IDE scenarios in which interactions like the ones outlined above are supported, and on structured (i.e., relational) datasets that are too large to be processed by a DBMS within interactive thresholds.

7.3.3 Database Landscape

Several commercial and academic DBMSs are either specifically built for IDE workloads, or can be used to support them. In the following, we summarize this landscape through three categories and provide examples for each.

Analytical DBMSs This category represents database systems that efficiently execute SQL queries in an exact manner. It includes column-stores and main-memory systems such as MonetDB [97], SAP HANA [51], Hyper [71] as well as database management systems that are designed for online analytical processing (OLAP) type workloads [26]. Their execution model cannot guarantee interactive response times on large datasets.

Approximate DBMSs Contrary to analytical DBMSs, approximate DBMSs use either offline or online sampling techniques to compute approximate answers. Most systems allow users to configure the trade-off between computation time and the quality of the result through a time or quality constraint as part of a SQL query. Examples include AQUA [4], VerdictDB [140], BlinkDB [5], as well as approXimateDB [86].

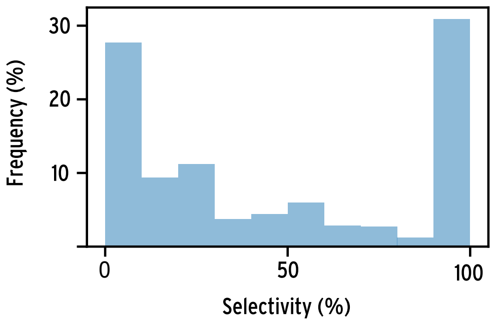
Specialized IDE DBMSs Prominent commercial examples of this category include Tableau [130] and its research predecessor Polaris [128], whose middle-layers are optimized for IDE workloads. ImMens [89] uses pre-computation over the entire query space to keep query run-times at a minimum. IDEA [54] is a middleware that provides interactive query execution on top of existing DBMSs such as Postgres or raw data sources such as CSV files. IDEA uses an online aggregation-based execution scheme to progressively compute and push query results to subscribers. Finally, DICE [65] is optimized for exploratory cube analysis and leverages interaction delays (i.e., "think-times") and a user interaction model to predict future queries.

7.4 The Need for an IDE Benchmark

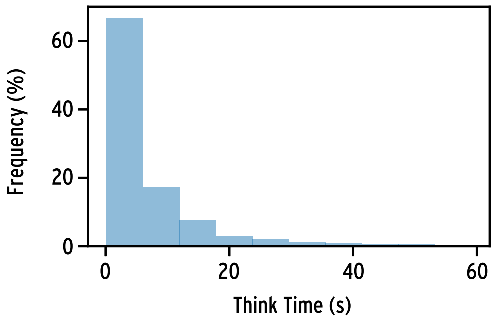
Although there are a number of existing analytical database benchmarks, this section highlights why they should not be used for IDE, why a new benchmark for IDE is needed, and which requirements it must meet.

7.4.1 Existing Analytical Benchmarks

Traditional analytical database benchmarks (TPC-H, TPC-DS, SSB) define a data-warehouse based workload with a fixed set of pre-defined SQL queries: TPC-H consists of 22 business oriented SQL queries, with a schema containing 8 tables (1 fact and 7 dimensions tables). TPC-DS comes with a much more complex schema containing 24 tables (7 fact and 17 dimensions tables) and 99 queries. SSB is based on TPC-H and re-organizes the database as a star schema with 5 tables (1 fact and 4 dimensions tables). SSB features 13 different queries, only some of which resemble TPC-H queries. Yet, as we explain in the next section, *none* of these benchmarks are representative of IDE workloads.



(a) Selectivity



(b) Think-Times

Figure 7.2: a) Shows the selectivity for all queries of user study participants. b) shows the distribution of think-times of all user study participants.

7.4.2 IDE Workload Analysis

To evaluate the applicability of existing analytical benchmarks, and to define benchmark requirements for IDE workloads, we draw on five independent user studies conducted with tools that are representative of what we outlined in Section 7.3.2. The five studies were conducted with 109 participants in total. Each study followed an open-ended (not tasked-based) protocol where participants were asked to explore real-world and synthetic datasets, and instructed to think-aloud and report insights that were noted down during the study. The only exception is user study 5, where we did not have the ability to create think-aloud protocols. However, we were able to use the interaction logs and query traces of the user study as basis of our analysis.

User Study 1 : Interactive Data Exploration Accelerators (IDEAs) For this study 35 participants were recruited to explore a subset of the 1994 US census, using a visual IDE system called Vizdom that supports progressive visualizations using a pen-and-touch interface. Its frontend was designed to interactively explore data by arranging 1D or 2D aggregate plots of attributes on an unbounded canvas. Plots in the tool are interactive, meaning that users can link plots and select arbitrary bin ranges to create filter chains, and apply boolean operations (AND, OR, NOT) to combine multiple filters, etc.

User Study 2: Tableau / Star-Schema Benchmark In this study, conducted as part of this paper, we invited 12 participants – all graduate students in computer science – to explore a dataset using Tableau. We used the SSB benchmark schema and generated a database with a scale factor of 0.1 ($\tilde{1}$00MB) to ensure that query results could be computed near-instantaneously. The goal of this study was to find out what whether the queries in SSB are representative of the traces that real users generate, and whether these traces overlap with the ones from study 1-3. Of the 12 participants, 8 stated to have some experience with data analysis, 4 reported to be advanced with data analysis (e.g., they have used Tableau, Power BI, or a similar visual exploration frontends multiple times). Each test subject was first given an overview of the database schema and a brief overview of the basic functionality of Tableau, e.g., how data can be plotted, filtered, and how multiple visualizations can be arranged in interactive dashboard views. The experimenter acted as a mediator between the participant and the software, i.e., the participants instructed the experimenter on what to do with the tool. We decided not to let the participants operate Tableau themselves as most pilot users struggled to remember the somewhat involved steps required to create plots, filters and dashboards.

User Study 3: Effect of Progressive Visualizations The goal of this study was to investigate how progressive visualizations affect users during exploratory data analysis. The authors recruited 24 participants from a research university, all of whom had prior experience in data exploration or analysis tools (e.g., Excel, R, Pandas). The authors created a dashboard-like user interface with four visualization slots, that users can configure by assigning them any attribute from a dataset. All visualization are 1D/2D aggregate plots with configurable aggregate functions (count, min, max, avg). The four visualizations were implicitly linked; selecting values ranges in one visualization would filter/brush the other visualizations.

User Study 4: Multiple Comparisons Problem (MCP) in Visual Analysis The aim of this work was to investigate the effect of MCP in visual analysis by evaluating the accuracy of user reported insights when exploring a dataset. This study required users to broadly explore datasets since they were explicitly asked to find interesting correlations in the data. All 28 participants were students who had some experience with data exploration or analysis tools (e.g., Tableau, Pandas, R) and have taken at least introductory college-level statistics and probability classes.

User Study 5: DARPA Competitions As part of a larger DARPA program, we participated with our own data exploration tool in a competition where 10 government data analysts were asked to explore new datasets. We gathered and analyzed the query traces that resulted from the exploration sessions.

Workload Analysis Given the results of user studies 1-5, we were interested to see (1) how users interact with IDE systems to make sense of data, how these interactions translate to SQL queries for a DBMS, and (2) if the workloads of existing benchmarks are representative of these queries. Our method involved analyzing multiple days worth of video and audio recordings, as well as the analysis

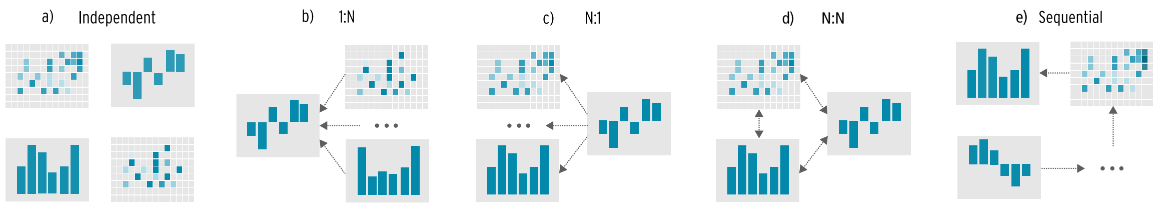


Figure 7.3: Illustrates five different dependency patterns (see 7.5.1 for details) we observed during the IDE workload analysis of five user studies. a) Independent: four independent visualizations, changing the specification of any of the visualizations does not affect the others as there are no dependencies. b,c,d and e) an arrow represents a dependency from source to target. For instance, if a selection in the visualization on the left in b) changes, all N visualizations on the right need to be updated.

of query logs from the underlying DBMSs. In the following, we discuss our main observations (O1 - O6).

O1: User actions that trigger database queries User actions that trigger database queries can be grouped into three abstract categories (independent of the IDE tool being used): 1) *creating/modifying visualizations*: the creation and modification of a plot involves settings parameters, such as the attributes that are being visualized, filters to apply to the data that is being visualized, as well as instructions on how to group/bin and aggregate the data. 2) *linking visualizations*: defining dependencies between plots so that they can be used as interactive filters 3) *selecting bins*: selecting a subset of the visualized data to filter linked plots.

O2: Aggregation dominates. Visualizing large datasets inevitably leads to over-plotting, which overwhelms users' perceptual and cognitive capacities. Aggregating data is the only practically reasonable way to support visual exploration of big data [89].

O3: Selectivity varies significantly. Queries formulated by study participants strongly varied in selectivity. We summarize the query selectivity of all queries from user study 3 in Figure 7.2, showing that approximately 30% of all queries either have a selectivity of $0 - 10\%$ while another 30% have a selectivity of $90 - 100\%$ respectively. Similar distributions emerged in user study 1.

O4: Queries are built incrementally. Users explore data and answer questions incrementally. Intermediate visualizations are used to inform further exploration steps. In all studies this was manifested by users either replacing parameters of an existing plot (e.g., the attribute being visualized), applying filters, etc. In user study 1, for instance, we noticed that the selections on plots were modified 6.91 times on average ($\sigma = 9$) to change a filter. In some cases users even changed the filter up to 50 times, trying to find interesting correlations for multiple minutes. In addition, we found that the number of attributes used to specify a query ranged from 1 to 5 different attributes (composed by AND and OR operators). We observed 5 different patterns of how the study participants used links to define dependencies between visualizations (see Figure 7.3):

- *Independent* (Figure 7.3a), where a user explores data by creating visualizations using individual queries and applying filters that only affect a single visualization at a time, as well as by altering the aggregation function (e.g., switching from `SUM` to `AVG`). This pattern was often used as a first step to browse through different attributes in a dataset.
- *1:N* (Figure 7.3b), where selecting bins one visualization triggers N other queries for all dependent plots. This pattern, as well as the N:1 and N:N, typically apply after using *independent* visualizations to see how slicing and dicing the data affects previously plotted data.
- *N:1* (Figure 7.3c), where a user links N visualizations to a single visualization. Selecting value ranges in any of the N plots adds a filter predicate to the linked plot, forcing it to update.
- *N:N* (Figure 7.3d), where all visualizations are inter-linked. This pattern is typically found in interactive dashboards, where every visualization is implicitly linked with all other plots.
- *Sequential* (Figure 7.3e), is a browsing pattern where users create multiple visualizations that are sequentially linked. This pattern was often used when users drill down (zoom in) into subsets of the data by building increasingly selective filter expressions, while keeping track of intermediate results.

O5: Single interactions can lead to multiple concurrent queries. Linked visualizations can lead to multiple concurrent queries. When users were free to arbitrarily create and link plots, they created dependency chains of up to 5 links (see Figure 7.3e, for example), meaning that a selection change on one plot would trigger up to 5 concurrent queries. Similarly, in user study 3, users were interacting with a pre-defined *N:N* dependency pattern between four visualizations in total. A selection in one plot led to an update in the other three visualizations.

O6: Think-Time between interactions.

We observed time gaps between user interactions (think-time) with high variance, ranging from just a few hundred milliseconds up to over 200 seconds in individual cases. Figure 7.2b shows the think-times recorded in user study 1 (outliers of more 60s were omitted). Most think-times were between 0 and 10s (mean=8.08s, $\sigma = 13.97$), which could be leveraged by DBMSs to speculatively execute SQL queries to prepare for the next user interaction.

O7: Real-data matters. Users are less interested in actual values, but more in the distribution of values and/or irregularities/outliers that can only be found in real datasets. Similar observations were made by Amar et al. in a related study [8]. This became especially apparent in user study 2 and 4 where synthetic data was used. With synthetic data, users found it difficult to formulate meaningful queries as all attribute values were uniformly distributed.

	TPC-H	*TPC-DS*	*SSB*	*IDEBench*
Schema	snowflake	snowflake	star	star (default)
Data Origin	synthetic	synthetic	synthetic	real-world
Data Distributions	uniform	skewed	uniform	real-world
Data Scaling	yes	yes	yes	yes
Iterative Query Formulation	no	4 out of 99	no	yes
Multi-Query Execution	no	no	no	yes
Think Time	no	no	no	yes
Metrics	Time-based	Time-based	Time-based	Quality, Time

Table 7.1: A comparison of traditional analytical database benchmarks and IDEBench

Using these observations we derived a set of key requirements (R1 - R8) for an IDE benchmark. The list of requirements are an attempt to capture crucial aspects of measuring performance of a data engine in the context of interactive data exploration, and goes beyond what traditional benchmarks are designed for.

Workload A benchmark for IDE should feature queries that resemble common IDE patterns. First, we note that based on O2 and O3, queries for aggregated plots have a common structure involving bins (group-bys), aggregated values (COUNT, AVG, etc.) as well as a set of filter predicates. For example, consider a dataset containing flight delays of domestic flights in the US [104]. A query for a plot showing the average flight delay per airport for flights originating in California can be structured as shown in Figure 7.4.

Second, it is pivotal that the benchmark not only runs individual queries in isolation (as done by TPC-H, TPC-DS, and SSB). Instead, as described in O4, the workload should contain time-spaced sequences of related queries mapping to the addition or modification of plots or selections

```
SELECT   AVG(DELAY) AS VALUE, AIRPORT AS BIN_AIRPORT,
FROM     FLIGHTS_DELAYS
WHERE    ORIGIN = 'CA'
GROUP BY BIN_AIRPORT
```

Figure 7.4: An example query for a plot showing the average flight delay per airport for flights originating in California.

therein (**R1**). Third, based on O5 the workload should representative of the fact that a single user interaction can lead to multiple concurrent queries (**R2**). Fourth, as stated in O6 and unlike TPC-H, TPC-DS, and SSB, a benchmark for IDE must honor that fact that there are think-times between user interactions, which data engines can leverage for speculative execution or other tasks (**R3**).

Data Contrary to existing benchmarks, a benchmark for IDE should use a real-world dataset with outliers (O7). The benchmark must provide tools to scale the data while maintaining correlations and outliers (**R4**). Finally, as some data engines (e.g., BlinkDB [5] or IDEA [35]) only support single-table schemas, it is important that the data is available in both normalized and de-normalized form (**R5**).

Metrics It is crucial that metrics go beyond measuring end-to-end runtimes of queries, and capture what matters to the end-user. Prior research suggests that there are two main factors that directly impact human performance in interactive data exploration: response time and accuracy. [61, 59, 20, 16, 119, 102, 22, 122]. For instance, Liu et. al. [87], showed that even response times of more than 500ms could lead to poor user performance. Researchers have also found evidence that poor accuracy or incomplete results (e.g., those of approximate and progressive systems) can lead to poor user performance [152] and false conclusions [36, 67]. Therefore, it is important that the metrics used in a benchmark for IDE reflect the trade-off between processing speed and the quality of the returned results: measures of responsiveness (**R6**), as well as measures of accuracy (**R7**).

Customizablity Like traditional analytical database benchmarks, an IDE benchmark must define a default configuration for workload and dataset. However, as there is no one-size-fits-all solution for all parameters of a benchmark, we argue that it is important to provide the ability to customize the workload and data to match different IDE usage scenarios (**R8**). While some may argue that the ability to customize workloads, datasets and other parameters is undesirable as it hinders comparability, we, like [14] believe that customizability is crucial for adoption by different communities. Being able to configure such settings in a benchmark allows users to publish their configurations along with the benchmark results, and enable others to reason about the applicability, benefits and drawbacks of database designs and processing techniques in particular usage scenarios.

7.5 The IDEBench Design

To address the shortcomings of existing benchmarks we designed IDEBench, a collection of tools that can be used to evaluate the performance of databases on generated workloads that closely resemble the ones we observed in the logs of various different user studies (see Section 7.4.2).

7.5.1 Workflow Generator

Inspired by the observations we made when users visually explore data, we propose a customizable workload generator capable of generating series of common time-spaced user actions in IDE (*work-flows*) which, directly or indirectly, trigger database queries. The workflow generator simulates user actions to create/modify and link visualizations (O1). The set of visualizations and actions used by the generator are abstract specifications that are independent of a concrete visualization tool.

Visualization Specification (VizSpec)

A VizSpec is an abstract definition of a visualization, and is identified by an `id`. For instance, every visualization created in a Tableau can be thought of being specified by a VizSpec. It determines how data is grouped (`binning`) using one or many data `dimensions` and details how to perform numerical binning, e.g., using a `bin_width` or list of numerical `boundaries`. Using the `aggregates` attribute, an arbitrary number of aggregates per bin (e.g., the average of a numerical column and the total count of item in a bin) can be defined. The `selection` attribute specifies which data ranges have been selected on a visualization. The `depends-on` attribute specifies a dependency on selections of other visualizations. Finally, the time of execution of a VizSpec can be specified using the `time` attribute, (e.g., the time of a plot creation or modification). The time is specified in milliseconds relative to the start time of a workflow to simulate users' think-time.

User Actions

Based on observation O1 (Section 7.4.2), the workflow generator defines the following user actions.

- *Create/Modify:* An action to either *create a new* or *modify an existing* visualization. Each action is defined by a full VizSpec for creating a new visualization or a partial VizSpec for modifying a visualization (see Figure 7.5 and 7.8) respectively.

```
Visualization:
    id:          (string)
    time:        (number)
    binning:     (BinningDimension[])
    aggregates:  (Aggregate[])
    selection:   (string)
    depends-on:  (string)

BinningDimension:
    column:      (string)
    width:       (optional number)
    boundaries:  (optional number[])

Aggregate:
    type:        (enum COUNT|DCOUNT|AVG|SUM|MIN|MAX)
    dimension:   (string)
```

Figure 7.5: A Visualization Specification

- *Link:* An action to link two existing visualizations, i.e. to establish a dependency from one visualization to another. It can be expressed by a partial VizSpec containing an `id` and one or more `id`s of visualizations it depends on (`depends-on`).
- *Select:* An action to select a subset of the data in a visualization can be expressed by partial VizSpec containing its `id` and a set of selection predicates as a `selection` string.

Query Model

Any VizSpec v can be mapped to a database query $Q(v)$ as shown in Figure 7.6 which is an abstract specification of SPJA queries. A query uses v's bin (group-by) and aggregates specification, and its `filter` attribute is recursively computed as union of all selections of visualizations v depends on. The `filter` attribute can be thought of as the `WHERE` clause of a SQL statement, comprising all relevant filter predicates. Joins between tables are implicitly defined in our query model by the attributes which are selected; i.e., we create equi-joins between those tables based on the given database schema of our data generator (see Section 7.5.2).

```
Query:
    binning:    (BinningDimension[])
    aggregates: (Aggregate[])
    filter:     (string)
    time:       (number)
```

Figure 7.6: IDEBench's Query Model

Using such an abstract query specification, a concrete database driver of our benchmark renders an executable query. For example, if the database provides a SQL-like interface we generate queries as shown in Figure 7.8.

Creating Workflows by Simulating User Actions

Our workflow generator is designed to model interactive data exploration workflows, i.e. actions to create new and modify existing visualizations (**R1**). It uses as a Markov chain comprising three user actions: *create/modify*, *link*, and *select bins* (see Figure 7.7). We derived the transition probabilities between any pair of these actions empirically through the analysis of the study logs described in Section 7.4. The workload generator comes with pre-configured but customizable (**R8**) transition probabilities that can be used to generate variants of workflows mimicking the five dependency patterns shown in Figure 7.3 (an independent workflow, for example, has zero transition probability from create/modify to either link or select bins). To create a new workflow, the workflow generator samples n actions from the Markov chain and adds a fixed inter-query pause (think-time) to consecutive actions by setting the `time` attribute of a VizSpec (**R3**) accordingly. We use fixed rather than dynamic think-times to be able to measure its effect on the accuracy of query results, as we shall see in the Section 7.7.4. An example workflow that was created by our generator for a 1:N workflow type is shown in Figure 7.8.

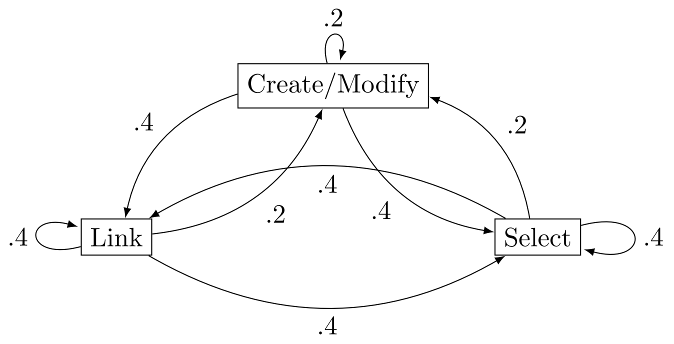


Figure 7.7: Example of a Markov Chain used for 1:N workflows.

Every user action exposes a set of parameters that are drawn randomly from a set of configurable probability distributions when creating a workflow (**R8**):

- *Create/Modify*: the number and types of binning dimensions (e.g., whether is a 1D or 2D plot), the bin width (i.e., the group-by granularity) or bin boundaries (i.e., minimum and maximum value used for each dimension), as well as the number and types of aggregates to use in the visualization.
- *Link*: linking strategy (none, 1:N, N:1, etc.).
- *Select Bins*: the bins to be selected as filter predicates.

Analogous to the transition probabilities between any of these actions, the defaults for the probability distributions of these parameters were informed by findings in the empirical analysis our user study logs.

When simulating an action the workflow generator re-computes the `filter` attribute for all affected queries and schedules affected queries for execution by setting the `time` attribute. To that end it keeps track of all VizSpecs or modifications thereof, by maintaining a directed dependency graph $G = (V, E)$. A directed edge in G defines a dependency (link) between two visualizations, e.g., $v_1 \to v_2$ indicates that $Q(v_1)$ must be re-executed when the specification of v_2, in particular its selection attribute is modified (**R2**).

7.5.2 Data Generator

Using real-world datasets in a benchmark for IDE is important as non-synthetic distributions make it harder for DBMSs to retrieve a representative sample, which consequently affects the quality of the results. Currently, IDEBench uses the U.S. domestic flight delay dataset [104] as default. However, users can customize the benchmark and plug in custom datasets.

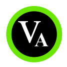
To scale a dataset to different sizes, IDEBench comes with a data generator that can resize a given dataset to any size (**R4**). The generator tries to maintain distributions in the data and relationships between attributes when scaling by simulating a Gaussian Copula [101]. The intuition behind this procedure is that every multivariate joint distribution can be expressed in terms of its marginal distributions and a function (the copula) which describes their relationship [125]. This decomposition allows for the creation (simulation) of new correlated samples.

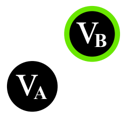
In order to simplify the resizing process, IDEBench requires that the dataset is provided in a star schema-like format, in de-normalized form. This, however, is no limitation since typical IDE queries (i.e., SPJA queries) are natively supported on a star-schema. The algorithm to scale a dataset works as follows:

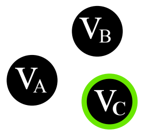
1. First, we draw a random sample from the dataset.
2. Second, we compute the covariance matrix Σ and perform a Cholesky decomposition on $\Sigma = A^T A$.
3. Finally, we resize the data. To create a new tuple, we generate a vector $X \sim \mathcal{N}(0, 1)$ of random normal variables and induce correlation by computing $\widetilde{X} = AX$. Furthermore, we transform $\widetilde{X}$ to a uniform distribution and finally use the CDF from the sample to transform the uniform variables to a correlated tuple.

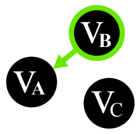
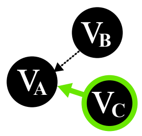
Optionally, as a last step the data generator can split the generated data into multiple tables (fact and dimensions) based on the user-provided schema (**R5**).

	VizSpec	Query Graph	Queries (SQL)

create	id: "A"
	time: 0
	binning:
	    dimension: "CARRIER"
	aggregates:
	    type: "avg"
	    dimension: "DEP_DELAY"
			SELECT
			    CARRIER,
			    AVG(DEP_DELAY)
			FROM
			    FLIGHTS
			GROUP BY
			    CARRIER

create	id: "B"
	time: 2000
	binning:
	    dimension: "CARRIER"
	aggregates:
	    type: "avg"
	    dimension: "DEP_DELAY"
			SELECT
			    CARRIER,
			    AVG(DEP_DELAY)
			FROM
			    FLIGHTS
			GROUP BY
			    CARRIER

create	id: "C"
	time: 4500
	binning:
	    dimension: "CARRIER"
	aggregates:
	    type: "avg"
	    dimension: "DEP_DELAY"
			SELECT
			    CARRIER,
			    AVG(DEP_DELAY)
			FROM
			    FLIGHTS
			GROUP BY
			    CARRIER

link	id: "B" time: 8000 depends_on: "A"		[no query]
link	id: "C" time: 11000 depends_on: "A"		[no query]

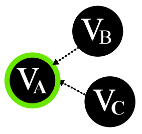
select bins	id: "A"
	time: 13300
	selection:
	    "CARRIER = ‘AA’ OR
	    CARRIER = ‘DL’ OR
	    CARRIER = ‘UA’"
			SELECT
			    ORIGIN_STATE,
			    COUNT(*)
			FROM
			    FLIGHTS
			GROUP BY
			    CARRIER
			WHERE
			    CARRIER = 'AA' OR
			    CARRIER = 'UA'
			
			SELECT
			    FLOOR(ARR_DELAY/30)
			    AS BIN_ARR_DELAY,
			    AVG(DEP_DELAY)
			FROM
			    FLIGHTS
			GROUP BY
			    BIN_ARR_DELAY
			WHERE
			    CARRIER = 'AA' OR
			    CARRIER = 'UA'

Figure 7.8: An example of a generated 1:N workflow (Figure 7.3c.), the corresponding VizSpecs and translations to SQL. The highlighted item in the query graph corresponds to the current user action. Note that the final interaction triggers two SQL queries.

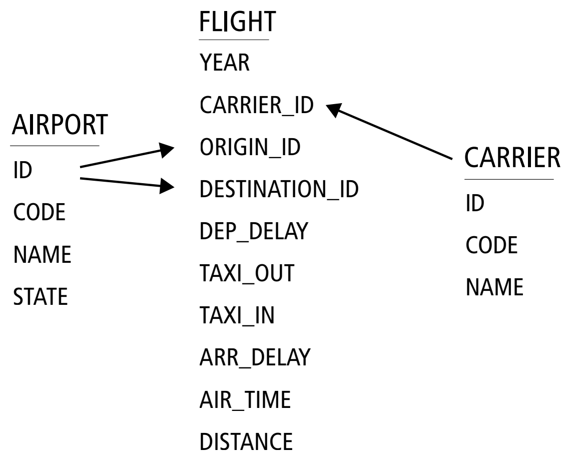


Figure 7.9: The schema of IDEBench's default dataset.

7.5.3 Main Metrics

IDEBench computes metrics that capture the trade of between the runtime of a query and the quality/usefulness of the results.

Data Preparation Time In IDEBench users are required to report on all actions taken to prepare for a benchmark run (called "data preparation time" in our report). This includes the time it takes to copy the dataset into the system, to create sample tables/views offline, perform pre-processing, execute warm-up queries, etc.

Then, for every query, we compute the following metrics:

Time Requirement Violations In IDEBench we measure the responsiveness of a DBMS using Time Requirements (TR). `TR Violated` is a boolean value indicating whether a query exceeded the time requirement specified in the settings (see Section 7.6.4). TR is violated if time TR after initiating the query no result is present or can be fetched. In practice, this means, that for batch-processing and AQP systems, TR is violated if the run-time of a query is greater than TR, and no intermediate result is present. For progressive systems, TR is violated if time TR after initiating a query no result can be fetched. Because IDEBench focuses on interactivity, it does not measure the actual query duration, i.e. it does not wait for query results to return if the computation takes longer than TR. If a query exceeds TR, database drivers can abort the query.

Mean Relative Error (MRE) Approximate and progressive results deviate from the ground-truth. To understand how much they deviate we measure the error between the *latest* result of an aggregate query and its ground-truth by computing the relative error, i.e., the ratio between the difference of the *latest* estimated result for a query F_i and the actual result A_i (**R7**).

$$\text{MRE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|F_i - A_i|}{|A_i|}$$

Missing Bins/Groups Missing Bins/Groups is the percentage of all bins missing in the *latest* result for query. It is a measure of completeness for an aggregate query result, irrespective of the number of tuples processed by a system. The intuition is that the faster a system can enumerate all groups, the more insightful this will be to a user (**R7**).

$$\text{Missing Bins} = \frac{|bins\_missing|}{|bins\_in\_groundtruth|}$$

Missing bins are reported for exact and non-exact systems. While for exact systems the percentage of missing bins is either 0% or 100% (depending on whether or not a result was present in time), for non-exact systems we use the latest result that was returned within the given time-threshold. The mean relative error is only computed for results of non-exact systems.

Cosine Distance Based on the observation that often users are more interested in characterizing data by its distribution rather than by the exact aggregated values [8], we additionally measure the cosine distance to test how much the “shape” of an aggregate result deviates from the shape of the ground-truth.

7.6 The IDEBench Implementation

IDEBench comprises two main components: a *benchmark driver*, which is responsible for simulating previously generated workflows, and *database adapters* which translate and delegate instructions from the benchmark driver to a database.

7.6.1 Benchmark Driver

The core of IDEBench is the benchmark driver, a command line application configured to load and simulate workflows created by the workflow generator. Given a workflow, the driver reads VizSpecs and builds and modifies an internal query graph structure, delegating queries and query updates to a database adapter (cf. Section 7.6.2). The database adapter then translates the VizSpecs from their JSON specification to the query language supported by the DBMS (e.g. SQL).

Listing 7.1: A stub for a database adapter.

```
class SampleAdapter:

    def workflow_start(self):
        # called before a benchmark run starts
        # do pre-processing, if applicable

    def run_query(self, viz_spec, result_queue):
        # 1. translate viz_spec into query
        # 2. execute query

    def workflow_end(self):
        # do clean-up, if applicable
```

7.6.2 Database Adapters

In order to benchmark an IDE system using IDEBench, a *database adapter* must be implemented, which acts as a proxy between the benchmark and the system that is being tested. The benchmark driver delegates changes made to the query graph to a database adapter, which then translates them into queries supported by the database (e.g., SQL). There are three core methods that must be implemented by a database adapter: `workflow_start`, which can be used to run pre-processing tasks before a workflow starts, `run_query` to initiate new or update existing VizSpecs (e.g., when adding a filter, see Listing 7.1), and `workflow_end` which is called once the workflow has ended, allowing the database adapter to perform clean-up tasks.

7.6.3 Benchmark Defaults

By default IDEBench uses 10 generated workflows using a mix of the four query graph patterns described in Section 7.5.1. Table 7.2 shows a summary of the most important parameters of IDEBench and their defaults, which we defined based on findings in our user studies. It uses the flight delay dataset (introduced in Section 7.5.2) scaled to 500M tuples in de-normalized form (i.e., only one large fact table). It also sets the time requirement TR for each query to 0.5 seconds, uses a think-time of 3 seconds between each user interaction, and a confidence level of 95%.

Upon completion of a benchmark run, IDEBench generates two reports: (1) An aggregated summary report listing how frequently the time requirement was violated, how many bins are missing on average, and the distribution of mean relative errors for all queries which did not violate the time requirement. Figure 7.10 shows an example of such a summary report. (2) A detailed report listing

Setting	**Description**	**Default**
Workflows	The workflows to run in the benchmark	10
Time Requirement (TR)	The maximum execution duration for a query	0.5s
Dataset and Size	The dataset to run the benchmark on and the number of rows in the fact table	Flights 500M
Think Time	The delay between two consecutive interactions	3s
Using Joins	Whether a star schema is used where dimension tables are pre-joined to fact table	False
Confidence Level	The confidence level at which an AQP/progressive systems return confidence intervals	95%

Table 7.2: Benchmark Settings

all settings and metrics on a per-query basis.

7.6.4 Customizing the Benchmark

For the sake of comparability, we encourage users of IDEBench to use the default configuration. However, all parameters can be modified so that users of IDEBench can test their database systems with settings that match a specific target use-case. For instance, some DBMS might be specifically optimized for data exhibiting certain distributions, for very low latencies, or for a workload that only supports specific types of queries.

7.7 An Experimental Study

To demonstrate the applicability of IDEBench and show the effects on the different classes of query processing engines (7.3), we conducted an experimental study using following DBMSs:

(1) *MonetDB:* a state-of-the-art open-source analytical DBMS, which uses a blocking query execution model that requires users to wait until an exact query result is computed. Thus, upon initiating a query, the run-time of a query is unknown. (2) *approXimateDB/XDB:* a PostgreSQL-based DBMS that supports online aggregation using the wander join algorithm [86]. It allows for a maximum run-time to be set in SQL. In addition, a "report interval" can be set so that intermediate results can be retrieved at fixed time intervals. XDB has some limitations in terms of query support, which we describe in detail in Section 7.7.2. (3) *IDEA:* a system that supports online aggregation and uses a fully progressive computation model; after initiating a query, results can be polled at any point in time. (4) *System X:* a commercial in-memory AQP engine operating on stratified sample tables that are created offline. The run-time of queries cannot be specified explicitly but must be indirectly adjusted by varying the size of samples tables. (5) *System Y:* a commercial specialized

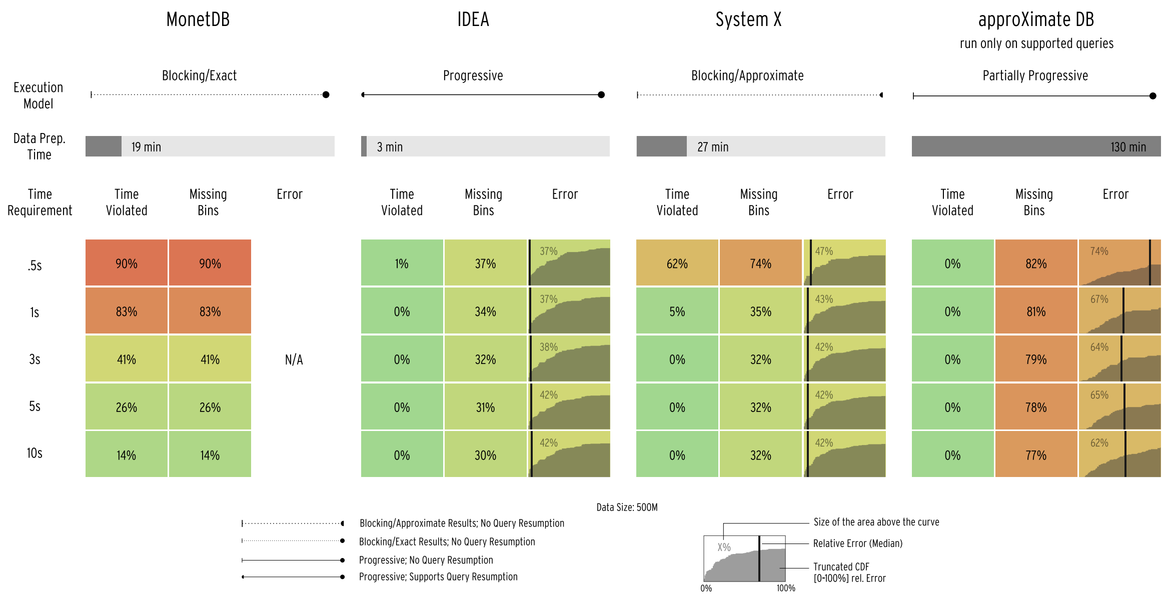


Figure 7.10: Shows the aggregated benchmark results for four systems in a summary report. The benchmark was run for five time requirements on a dataset with 500M rows. It shows the mean percentage of time violations and missing bins, as well as a CDF of the mean relative errors (MREs) truncated to errors less or equal to 100%. Thus, the greater the proportion of small errors, the smaller the area above the curve (shown as percentage above the CDF).

SQL engine for IDE that uses a blocking execution model, which is designed as an in-memory optimization layer on top of various different DBMSs.

In the remainder of this section, we describe the configuration and setup of our experiments, and discuss the results and findings.

7.7.1 Configuration and Setup

Configuration We used the default flight delay dataset (see Section 7.5.2), with S=100 million, M=500 million (default), and L=1 billion tuples in de-normalized form (i.e., a single pre-joined table). Furthermore, we used five different time requirements (TR) 0.5s, 1s, 3s (default), 5s, and 10s, and a fixed confidence level of 95% (default). While various Human-Computer-Interaction studies recommend time requirements of less than $1s$ (see Section 7.4), we also included larger ones to get a better sense of how fast results converge. Because none of the systems use speculative query execution in their default configuration, we set the think-time to a fixed value of $1s$; we analyze the effect of varying the think-times in a separate experiment (see Section 7.7.4).

Setup We ran IDEBench on *MonetDB*, *approXimateDB*, *IDEA* and *System X*, but were unable to run the full benchmark on *System Y* due to the absence of an API that could be used in a

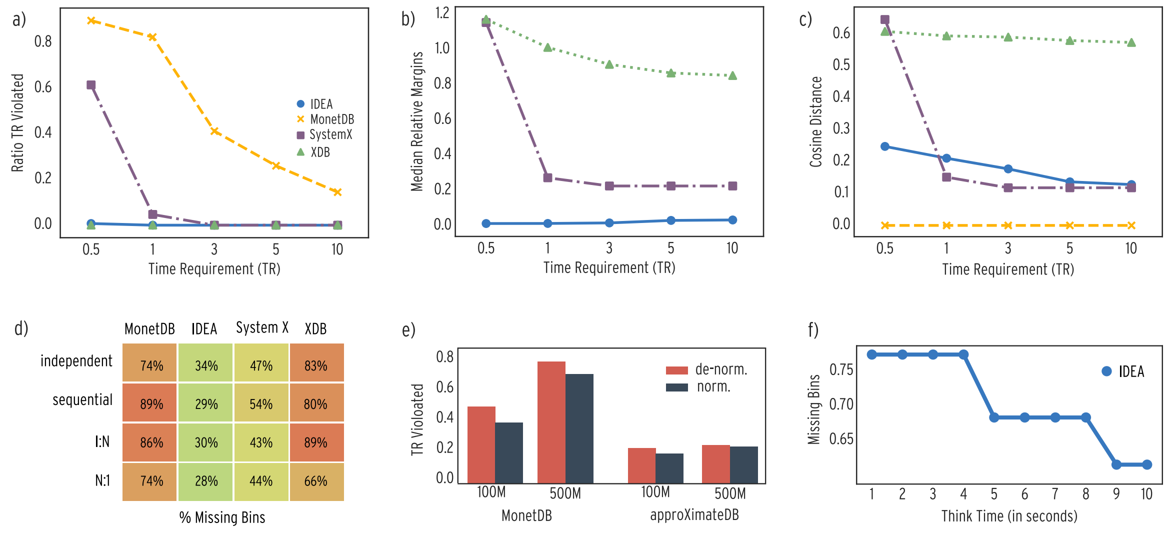


Figure 7.11: a, b, and c) show how the ratio of TR violations, the median of the mean relative margins, and the cosine distance behave for increasing time requirements. d) Compares how the percentage of missing bins differs depending on which system and workflow type is used. e) A comparison of the percentage of violated time requirements for MonetDB and *approXimateDB*, using a normalized and de-normalized dataset of size 500M. f) Shows the effect of varying think-times on missing bins for *IDEA*.

database adapter. However, we executed selected queries of our benchmark manually through its user interface in a separate experiment (see Section 7.7.5). All experiments were conducted on a computer with two Intel E5-2660 CPUs (2.2GHz, 10 cores, 25MB cache) and 256GB RAM. We used the default parameters for all DBMSs without hand-tuning them and did not tweak or optimize any of the system parameters.

7.7.2 Exp. 1: Overall Results

In our main experiment (Figure 7.10 and 7.11) we analyzed how the four systems behave with respect to different time requirements (see Figure 7.10). We show the results for 10 workflows of the default configuration, a data size of 500M tuples, de-normalized schema, and a confidence level of 95%.

Data Pre-Processing In *MonetDB* data stored in a CSV file can be loaded into the database through an SQL interface, which takes approximately 19 minutes for 500M records. *IDEA* expects data in a single randomized CSV file. On start-up the system loads a configurable number of tuples into main memory, which took approximately 3min in our experiment. With *System X* data stored in a CSV file can be loaded into the database through a SQL interface. In order to be able to execute approximate queries, stratified sample tables have to be created offline. We used a sample size of 1% of the dataset size. *System X* further requires that each connection executes a warm-up query when the system is restarted. For 500M records, we measured a data preparation time of 27min. Finally, *approXimateDB* took 130min to load and prepare the data. While this system provides support to pre-load relations and indexes into the database buffer in main memory, we did not make use of this feature for our experiments.

Speed and Quality Metrics As expected for an exact execution model, *MonetDB*'s TR violations decrease roughly linearly with the time requirement, and so does the percentage of missing bins (see also Figure 7.11a). On the contrary, *approXimateDB* never violates the TR as the query runtime can be specified accordingly. However, it is important to note that this system only supports online aggregation for `COUNT` and `SUM`, but does not support `AVG` and multiple aggregates in a single query. Therefore, we excluded all unsupported queries (34% of all queries) for *approXimateDB*. For comparison, running the experiment for all queries (falling back to regular Postgres for unsupported queries) leads to TR violations of 66% for all TRs, i.e., all non-approximate queries exceeded TR.

With *System X* more than 60% of all queries violate TR=0.5s. Interestingly, for TR=1s only 5% are violated, and for TR=3s all query results are returned on time. The percentage where TR is violated is therefore a good indicator of how large a sample table needs to be, if speed is more important than result quality. *IDEA* does not violate any TR, with the exception of 1% of all queries for TR=0.5. The authors confirmed that this is due a slightly higher overhead for the first query after a restart of the system. *IDEA* also starts off with significantly less missing bins (37%) for TR=0.5 than any other system, but achieves similar performance as *System X* for TR¿=1s. Furthermore, *IDEA* manages to perform better than other systems in terms of mean relative error of all returned results. The median of all mean relative errors is significantly less than *approXimateDB*'s and *System X*.

Another interesting fact can be observed by inspecting the cumulative distribution of mean relative errors. In Figure 7.10 (column *Error*), *approXimateDB*'s "area above the curve" is significantly greater than the one of *IDEA* and *System X*, indicating that high mean relative errors occur more frequently. A similar conclusion can be drawn by looking at the end of the curve. For instance, *approXimateDB*'s curve ends, below 50%, indicating that more than 50% of all mean relative errors are greater than 100%.

Figure 7.11b and 7.11c show how the median of the mean relative margins, and the cosine distance changes for the different DBMSs with increasing time requirements. Again, *approXimateDB* has significantly greater relative margins than both *IDEA* and *System X*. Moreover, while *System X*'s median is close to 120% for TR=0.5s and drops to approximately 20% for TR=1s, *IDEA*'s median remains constant around zero for all TRs. Finally, Figure 7.11d compares how the proportion of missing bins differs based on the system and query graph patterns. As none of the systems we used in the evaluation use speculative execution by default, there are only few notable differences per column. For instance, *MonetDB* has fewer missing bins on average for "independent browsing" and N:1 patterns, which may be attributed to the fact that interactions of these workflows only trigger a single query.

7.7.3 Exp. 2: Varying Schema Complexity

In this experiment, we compare the performance of *MonetDB* and *approXimateDB* using a normalized and de-normalized schema. Contrary to experiment 1, we set up *approXimateDB* so that any query that cannot be executed online will fall back to a regular Postgres query. We exclude *IDEA* and *System X* as they do not support joins. Using the data generator we created two datasets of 100M and 500M tuples and normalized the data so that the fact table holds foreign keys to two dimension tables (airports and carriers). Interestingly, as can be seen in Figure 7.11e, both *MonetDB* and *approXimateDB* perform slightly better in terms of time requirement violations with a normalized schema since the overall scan volume of data significantly decreases. *MonetDB*'s proportion of TR violations grows with the size of the normalized dataset, while *approXimateDB*'s TR violations remain steady due its ability to perform joins online.

7.7.4 Exp. 3: Varying Think-Time

In this experiment, we evaluated the impact of increasing think times between interactions (see Figure 7.11f). We used an experimental extension of *IDEA* that speculatively executes queries when two visualizations are linked. For this experiment, we used a fixed data size of 500M tuples, a time requirement of 3 seconds, and created a custom workflow comprising two linked visualizations. Internally, *IDEA* uses a simple speculative execution procedure. While user think IDEA starts queries for every possible single bin selection in the source visualization. If too many bins exist, it selects only the k visually most dominant ones. If upon the next interaction one of the bins is selected, *IDEA* can return a potentially better estimate of the results, as the query was given more processing time. Figure 7.11f shows the results of this experiment with the proportion of missing bins for ten different think times (1s - 10s).

7.7.5 Exp. 4: Experiment with System Y

In the last experiment, we manually executed a selected subset of our workflows in a commercial IDE System Y and used MonetDB as a backend. We used a fixed data size of 500M and performed three variants of the 1:N workflow type. In particular, we were interested to see whether *System X*'s middle layer that pre-fetches and pre-computes query results leads to significant benefits over using a vanilla MonetDB setup. However, we did not find this to be the case. System Y renders and updates the visualizations in the workload roughly at the same speed as if *MonetDB* is used directly.

7.7.6 Discussion

Our experiments with IDEBench and five systems have shown that the performance on IDE workloads in terms of data preparation time, responsiveness as well as the quality of the results can vary significantly, and thus these systems are not equally suitable for IDE.

Preparation Time Creating representatives samples offline is challenging: users need to find a suitable sample size to balance the trade-off between processing speed and quality of the results. Although the time overhead of offline-sampling approaches can be reduced by employing online sampling instead, systems using online sampling such as *approXimateDB* may still require much time to load data into the DBMS. *IDEA*, for instance, bypasses the loading problem by reading samples from disk at runtime. The system, however, requires the data to be randomized prior to ingestion.

Time Violations We found that progressive and AQP systems like *IDEA* and *System X* were able to keep time violations at a minimum while maintaining low error rates with increasing data sizes and time requirements. This is in stark contrast to traditional analytical databases represented by *MonetDB* where time violations increase for larger datasets and time requirements. We also found that *approXimateDB* can only execute a subset of the queries in our workload online. It has to revert to executing a significant number of queries in a blocking fashion, which leads to notably more TR violations. Finally, given the log of queries that a user executes during an exploration workflow, systems can leverage think-times to speculatively execute queries in order to provide faster responses upon the next user interaction.

Error Metrics Progressive systems progressively refine results and therefore lower the errors and confidence intervals over time, converging to an exact result. AQP systems that create sample tables offline, on the other hand, have constants error rates and confidence intervals, irrespective of the time requirement. To optimize the error rates for such systems, users must find representative sample that is small enough not to violate the time requirement, which can be challenging. Finally, we observed weak completeness scores, i.e. high missing-bin values for low time requirements and in workflows containing concurrent queries (such as sequential or 1:N).

7.8 Conclusion

In this chapter, we presented IDEBench, a new benchmark designed to evaluate systems for interactive data exploration (IDE). Unlike traditional analytical database benchmarks, IDEBench's workloads and metrics are inspired by the requirements and usage patterns of real IDE systems that were derived through a number of different user studies. Using our benchmark we conducted an evaluation that included five different DBMSs (approximateDB, IDEA, MonetDB as well as two commercial systems) representing three different system categories (traditional exact DBMSs, approximate/progressive DBMSs, as well as specialized engines for IDE). The results showed that especially for low latency requirements approximate and progressive query processing engines outperform traditional databases.

7.9 Acknowledgements

This research is funded by the DARPA Award 16-43-D3M-FP040, NSF Award IIS-1562657 and NSF Award IIS-1514491 and supported by Google, Intel, and Microsoft as part of the MIT Data Systems and AI Lab (DSAIL).

Bibliography

[1] Chicagolobbyists.org. `http://chicagolobbyists.org/about`, 2019. Accessed: 2019-09-06.

[2] Opensecrets.org. `https://www.opensecrets.org/`, 2019. Accessed: 2019-09-06.

[3] Daniel J. Abadi et al. The Beckman Report on Database Research. *ACM SIGMOD Record*, 43(3):61–70, 2014.

[4] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. The aqua approximate query answering system. In *ACM SIGMOD*, pages 574–576, 1999.

[5] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 29–42. ACM, 2013.

[6] Jae-wook Ahn, Catherine Plaisant, and Ben Shneiderman. A task taxonomy for network evolution analysis. *IEEE transactions on visualization and computer graphics*, 20(3):365–376, 2013.

[7] Wolfgang Aigner, Silvia Miksch, Heidrun Schumann, and Christian Tominski. *Visualization of time-oriented data*. Springer Science & Business Media, 2011.

[8] Robert Amar, James Eagan, and John Stasko. Low-level components of analytic activity in information visualization. In *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*, pages 111–117. IEEE, 2005.

[9] Saleema Amershi et al. ModelTracker: Redesigning Performance Analysis Tools for Machine Learning. In *ACM CHI Conference*, pages 337–346, 2015.

[10] Henrik André-Jönsson and Dushan Z Badal. Using signature files for querying time-series data. In *Principles of Data Mining and Knowledge Discovery*, pages 211–220. Springer, 1997.

[11] Sriram Karthik Badam, Jieqiong Zhao, Shivalik Sen, Niklas Elmqvist, and David Ebert. Timefork: Interactive prediction of time series. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 5409–5420. ACM, 2016.

[12] David Bamman, Brendan O'Connor, and Noah A Smith. Learning latent personas of film characters. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 352–361, 2013.

[13] Gustavo EAPA Batista, Xiaoyue Wang, and Eamonn J Keogh. A complexity-invariant distance measure for time series. SIAM.

[14] Leilani Battle, Remco Chang, Jeffrey Heer, and Michael Stonebraker. Position statement: The case for a visualization performance benchmark. *IEEE Internet Computing*, 13(3):48–55, 2009.

[15] Dominikus Baur, Bongshin Lee, and Sheelagh Carpendale. Touchwave: kinetic multi-touch manipulation for hierarchical stacked graphs. In *Proceedings of ACM ITS '12*, pages 255–264, 2012.

[16] Tom Beigbeder, Rory Coughlan, Corey Lusher, John Plunkett, Emmanuel Agu, and Mark Claypool. The effects of loss and latency on user performance in unreal tournament 2003®. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 144–151. ACM, 2004.

[17] Anastasia Bezerianos, Fanny Chevalier, Pierre Dragicevic, Niklas Elmqvist, and Jean-Daniel Fekete. Graphdice: A system for exploring multivariate social networks. In *Computer Graphics Forum*, volume 29, pages 863–872. Wiley Online Library, 2010.

[18] Sourav S Bhowmick, Byron Choi, and Shuigeng Zhou. Vogue: Towards a visual interaction-aware graph query processing framework. In *CIDR*, 2013.

[19] Markus Bögl, Wolfgang Aigner, Peter Filzmoser, Tim Lammarsch, Silvia Miksch, and Alexander Rind. Visual analytics for model selection in time series analysis. *IEEE transactions on visualization and computer graphics*, 19(12):2237–2246, 2013.

[20] Jake Brutlag. Speed matters for google web search. `https://services.google.com/fh/files/blogs/google_delayexp.pdf`, 2009.

[21] Nan Cao, Yu-Ru Lin, Liangyue Li, and Hanghang Tong. g-miner: Interactive visual group mining on multivariate graphs. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 279–288. ACM, 2015.

[22] Stuart K Card, George G Robertson, and Jock D Mackinlay. The information visualizer, an information workspace. In *ACM SIGCHI*, pages 181–186. ACM, 1991.

[23] Varun Chandola et al. Anomaly Detection: A Survey. *ACM Computing Surveys*, 41(3):15:1–15:58, 2009.

[24] Duen Horng Chau, Christos Faloutsos, Hanghang Tong, Jason I Hong, Brian Gallagher, and Tina Eliassi-Rad. Graphite: A visual query system for large graphs. In *2008 IEEE International Conference on Data Mining Workshops*, pages 963–966. IEEE, 2008.

[25] Duen Horng Chau, Aniket Kittur, Jason I Hong, and Christos Faloutsos. Apolo: making sense of large network data by combining rich user interaction and machine learning. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 167–176. ACM, 2011.

[26] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *ACM Sigmod record*, 26(1):65–74, 1997.

[27] Lei Chen, M Tamer Özsu, and Vincent Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 491–502. ACM, 2005.

[28] Yueguo Chen, Mario A Nascimento, Beng Chin Ooi, and A Tung. Spade: On shape-based pattern detection in streaming time series. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 786–795. IEEE, 2007.

[29] Eun Kyoung Choe, Raimund Dachselt, Petra Isenberg, and Bongshin Lee. Mobile data visualization (dagstuhl seminar 19292). *Dagstuhl Reports*, 2019.

[30] Sophie Chou, William Li, and Ramesh Sridharan. Democratizing data science. In *Proceedings of the KDD 2014 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA*, pages 24–27. Citeseer, 2014.

[31] Jonathan Cornelissen. The democratization of data science. `https://hbr.org/2018/07/the-democratization-of-data-science`. Accessed: 2019-04-27.

[32] Andrew Crotty, Alex Galakatos, Kayhan Dursun, Tim Kraska, Carsten Binnig, Ugur Çetintemel, and Stan Zdonik. An architecture for compiling udf-centric workflows. *PVLDB*, 8(12):1466–1477, 2015.

[33] Andrew Crotty, Alex Galakatos, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. Vizdom: interactive analytics through pen and touch. *Proceedings of the VLDB Endowment*, 8(12):2024–2027, 2015.

[34] Andrew Crotty, Alex Galakatos, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. Vizdom: Interactive analytics through pen and touch. In *PVLDB*, 2015.

[35] Andrew Crotty, Alex Galakatos, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. The case for interactive data exploration accelerators (IDEAs). In *HILDA@SIGMOD*, page 11. ACM, 2016.

[36] Geoff Cumming and Sue Finch. Inference by eye: confidence intervals and how to read pictures of data. *American Psychologist*, 60(2):170, 2005.

[37] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. *Computational geometry*. Springer, 2000.

[38] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment*, 1(2):1542–1552, 2008.

[39] Marian Dörk, Nathalie Henry Riche, Gonzalo Ramos, and Susan Dumais. Pivotpaths: Strolling through faceted information spaces. *IEEE transactions on visualization and computer graphics*, 18(12):2709–2718, 2012.

[40] Finale Doshi-Velez and Been Kim. Towards a Rigorous Science of Interpretable Machine Learning. *ArXiv e-prints*, abs/1702.08608, 2017.

[41] Steven M Drucker, Danyel Fisher, Ramik Sadana, Jessica Herron, et al. Touchviz: a case study comparing two interfaces for data analytics on tablets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2301–2310. ACM, 2013.

[42] Cody Dunne, Nathalie Henry Riche, Bongshin Lee, Ronald Metoyer, and George Robertson. Graphtrail: Analyzing large multivariate, heterogeneous networks while supporting exploration history. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1663–1672. ACM, 2012.

[43] Cody Dunne, Nathalie Henry Riche, Bongshin Lee, Ronald A. Metoyer, and George G. Robertson. Graphtrail: analyzing large multivariate, heterogeneous networks while supporting exploration history. In *CHI*, 2012.

[44] Darren Edge, Jonathan Larson, Markus Mobius, and Christopher White. Trimming the hairball: Edge cutting strategies for making dense graphs usable. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 3951–3958. IEEE, 2018.

[45] Darren Edge, Jonathan Larson, and Christopher White. Bringing ai to bi: enabling visual analytics of unstructured data in a modern business intelligence platform. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, page CS02. ACM, 2018.

[46] Philipp Eichmann et al. Discrete Time Specifications in Temporal Queries. In *ACM CHI Conference*, pages 2536–2542, 2017.

[47] Philipp Eichmann and Emanuel Zgraggen. Evaluating Subjective Accuracy in Time Series Pattern-Matching using Human-Annotated Rankings. In *IUI Conference*, pages 28–37, 2015.

[48] Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM Trans. Graph.*, 31(4):44, 2012.

[49] Muhammad El-Hindi, Zheguang Zhao, Carsten Binnig, and Tim Kraska. Vistrees: fast indexes for interactive data exploration. In *ACM SIGMOD*, page 5, 2016.

[50] Christos Faloutsos, Mudumbai Ranganathan, and Yannis Manolopoulos. *Fast subsequence matching in time-series databases*, volume 23. ACM, 1994.

[51] Franz Färber, Norman May, Wolfgang Lehner, Philipp Große, Ingo Müller, Hannes Rauhe, and Jonathan Dees. The SAP HANA database – an architecture overview. *IEEE Data Eng. Bull.*, 35(1):28–33, 2012.

[52] Jeff Feng, Erin Coffman, and Elena Grewal. How airbnb democratizes data science with data university. `https://medium.com/airbnb-engineering/how-airbnb-democratizes-data-science-with-data-university-3eccc71e073a`. Accessed: 2019-04-27.

[53] Tak-chung Fu, Fu-lai Chung, Robert Luk, and Chak-man Ng. Stock time series pattern matching: Template-based vs. rule-based approaches. *Engineering Applications of Artificial Intelligence*, 20(3):347–364, 2007.

[54] Alex Galakatos, Andrew Crotty, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. Revisiting reuse for approximate query processing. *PVLDB*, 10(10):1142–1153, 2017.

[55] Google. What-If Tool. `http://pair-code.github.io/what-if-tool/`, 2018.

[56] Iqbal A Goralwalla, M Tamer Ozsu, and Duane Szafron. A framework for temporal data models: exploiting object-oriented technology. In *Technology of Object-Oriented Languages and Systems, 1997. TOOLS 23. Proceedings*, pages 16–30. IEEE, 1998.

[57] David Gotz and Harry Stavropoulos. Decisionflow: Visual analytics for high-dimensional temporal event sequence data. *IEEE transactions on visualization and computer graphics*, 20(12):1783–1792, 2014.

[58] Machon Gregory and Ben Shneiderman. Shape identification in temporal data sets. pages 305–321, 2012.

[59] Pat Hanrahan. Analytic database technologies for a new kind of user: the data enthusiast. In *ACM SIGMOD*, pages 577–578. ACM, 2012.

[60] Jeffrey Heer and Adam Perer. Orion: A system for modeling, transformation and visualization of multidimensional heterogeneous networks. *Information Visualization*, 13(2):111–133, 2014.

[61] Jeffrey Heer and Ben Shneiderman. Interactive dynamics for visual analysis. *Queue*, 10:30, 2012.

[62] Harry Hochheiser and Ben Shneiderman. Interactive exploration of time series data. In *International Conference on Discovery Science*, pages 441–446. Springer, 2001.

[63] Harry Hochheiser and Ben Shneiderman. Dynamic query tools for time series data sets: timebox widgets for interactive exploration. *Information Visualization*, 3(1):1–18, 2004.

[64] Eyke Hüllermeier, Johannes Fürnkranz, Weiwei Cheng, and Klaus Brinker. Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172(16):1897–1916, 2008.

[65] Prasanth Jayachandran, Karthik Tunga, Niranjan Kamat, and Arnab Nandi. Combining user interaction, speculative query execution and sampling in the dice system. *PVLDB*, 7:1697–1700, 2014.

[66] Jaemin Jo, Sehi L'Yi, Bongshin Lee, and Jinwook Seo. Touchpivot: blending wimp & post-wimp interfaces for data exploration on tablet devices. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 2660–2671. ACM, 2017.

[67] Susan Joslyn and Jared LeClerc. Decisions with uncertainty: the glass half full. *Current Directions in Psychological Science*, 22(4):308–315, 2013.

[68] Niranjan Kamat, Prasanth Jayachandran, Karthik Tunga, and Arnab Nandi. Distributed and interactive cube exploration. In *ICDE*, pages 472–483. IEEE, 2014.

[69] Hyunmo Kang, Catherine Plaisant, Bongshin Lee, and Benjamin B Bederson. Netlens: iterative exploration of content-actor network data. *Information Visualization*, 6(1):18–31, 2007.

[70] Amy K Karlson, George G Robertson, Daniel C Robbins, Mary P Czerwinski, and Greg R Smith. Fathumb: a facet-based interface for mobile search. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 711–720. ACM, 2006.

[71] Alfons Kemper and Thomas Neumann. Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In *ICDE*, pages 195–206. IEEE, 2011.

[72] Eamonn Keogh and Shruti Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. *Data Mining and knowledge discovery*, 7(4):349–371, 2003.

[73] Eamonn J Keogh and Michael J Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *KDD*, volume 98, pages 239–243, 1998.

[74] Eamonn J Keogh and Michael J Pazzani. Relevance feedback retrieval of time series data. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 183–190. ACM, 1999.

[75] Zhu Q. Hu B. Hao. Y. Xi X. Wei L. & Ratanamahatana Keogh, E. *The UCR Time Series Classification/Clustering Homepage*, 2011. `www.cs.ucr.edu/~eamonn/time_series_data/`.

[76] Tim Kraska. Northstar: An interactive data science system. *Proceedings of the VLDB Endowment*, 11(12):2150–2164, 2018.

[77] Matthieu Latapy, Clémence Magnien, and Nathalie Del Vecchio. Basic notions for the analysis of large two-mode networks. *Social networks*, 30(1):31–48, 2008.

[78] ICIJ Offshore Leaks. Panama papers. `http://offshoreleaks.icij.org`, 2019. Accessed: 2019-09-06.

[79] Bongshin Lee, Matthew Brehmer, Petra Isenberg, Eun Kyoung Choe, Ricardo Langner, and Raimund Dachselt. Data visualization on mobile devices. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–8, 2018.

[80] Bongshin Lee, Eun Kyoung Choe, Petra Isenberg, Kim Marriott, and John Stasko. Reaching broader audiences with data visualization. *IEEE Computer Graphics and Applications*, 40(2):82–90, 2020.

[81] Bongshin Lee, Mary Czerwinski, George Robertson, and Benjamin B Bederson. Understanding research trends in conferences using paperlens. In *CHI'05 extended abstracts on Human factors in computing systems*, pages 1969–1972. ACM, 2005.

[82] Bongshin Lee, Petra Isenberg, Nathalie Henry Riche, and Sheelagh Carpendale. Beyond mouse and keyboard: Expanding design considerations for information visualization interactions. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2689–2698, 2012.

[83] Bongshin Lee, Catherine Plaisant, Cynthia Sims Parr, Jean-Daniel Fekete, and Nathalie Henry. Task taxonomy for graph visualization. In *Proceedings of the 2006 AVI workshop on BEyond time and errors: novel evaluation methods for information visualization*, pages 1–5. ACM, 2006.

[84] Bongshin Lee, Greg Smith, Nathalie Henry Riche, Amy Karlson, and Sheelagh Carpendale. Sketchinsight: Natural data exploration on interactive whiteboards leveraging pen and touch interaction. In *2015 IEEE Pacific Visualization Symposium (PacificVis)*, pages 199–206. IEEE, 2015.

[85] Bongshin Lee, Greg Smith, George G Robertson, Mary Czerwinski, and Desney S Tan. Facetlens: exposing trends and relationships to support sensemaking within faceted datasets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1293–1302. ACM, 2009.

[86] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. Wander join: Online aggregation via random walks. In *ACM SIGMOD*, pages 615–629. ACM, 2016.

[87] Zhicheng Liu and Jeffrey Heer. The effects of interactive latency on exploratory visual analysis. *IEEE transactions on visualization and computer graphics*, 20:2122–2131, 2014.

[88] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. immens: Real-time visual querying of big data. *Computer Graphics Forum (Proc. EuroVis)*, 32, 2013.

[89] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. immens: Real-time visual querying of big data. In *Computer Graphics Forum*, volume 32, pages 421–430. Wiley Online Library, 2013.

[90] Zhicheng Liu, Shamkant B Navathe, and John T Stasko. Ploceus: Modeling, visualizing, and analyzing tabular data as networks. *Information Visualization*, 13(1):59–89, 2014.

[91] Miro Mannino and Azza Abouzied. Expressive Time Series Querying with Hand-Drawn Scale-Free Sketches. In *ACM CHI Conference*, pages 388:1–388:12, 2018.

[92] Gary Marchionini. Exploratory search: from finding to understanding. *Communications of the ACM*, 49(4):41–46, 2006.

[93] Hugo Mathien. European soccer database, Oct 2016.

[94] Peter McLachlan, Tamara Munzner, Eleftherios Koutsofios, and Stephen North. Liverac: interactive visual exploration of system management time-series data. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1483–1492. ACM, 2008.

[95] Mary Meeker. Internet trends report. `https://www.kleinerperkins.com/files/INTERNET_TRENDS_REPORT_2017.pdf`, 2017. Accessed: 2020-03-21.

[96] Theophano Mitsa. *Temporal data mining*. CRC Press, 2010.

[97] Monetdb. `http://www.monetdb.org`. Accessed: 2019-11-02.

[98] Megan Monroe, Rongjian Lan, Hanseung Lee, Catherine Plaisant, and Ben Shneiderman. Temporal event sequence simplification. *IEEE transactions on visualization and computer graphics*, 19(12):2227–2236, 2013.

[99] Megan Monroe, Rongjian Lan, Juan Morales del Olmo, Ben Shneiderman, Catherine Plaisant, and Jeff Millstein. The challenges of specifying intervals and absences in temporal queries: A graphical language approach. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2349–2358. ACM, 2013.

[100] Michael D Morse and Jignesh M Patel. An efficient and accurate method for evaluating time series similarity. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 569–580. ACM, 2007.

[101] Roger B Nelsen. *An introduction to copulas*. Springer Science & Business Media, 2007.

[102] Jakob Nielsen. Powers of 10: Time scales in user experience. *Retrieved January*, 5:2015, 2009.

[103] Carolina Nobre, Marc Streit, and Alexander Lex. Juniper: A tree+ table approach to multivariate graph visualization. *IEEE transactions on visualization and computer graphics*, 25(1):544–554, 2018.

[104] Bureau of Transportation Statistics. Bureau of transportation statistics. `http://www.transtats.bts.gov`, 2017. Accessed: 2019-10-21.

[105] Patrick E O'Neil, Elizabeth J O'Neil, and Xuedong Chen. The star schema benchmark (ssb). *Pat*, 200(0):50, 2007.

[106] Themistoklis Palpanas, Michail Vlachos, Eamonn Keogh, Dimitrios Gunopulos, and Wagner Truppel. Online amnesic approximation of streaming time series. In *Data Engineering, 2004. Proceedings. 20th International Conference on*, pages 339–349. IEEE.

[107] Panagiotis Papapetrou, Vassilis Athitsos, Michalis Potamias, George Kollios, and Dimitrios Gunopulos. Embedding-based subsequence matching in time-series databases. *ACM Transactions on Database Systems (TODS)*, 36(3):17, 2011.

[108] Kayur Dushyant Patel. *Lowering the barrier to applying machine learning.* PhD thesis, 2013.

[109] Robert Pienta, Fred Hohman, Acar Tamersoy, Alex Endert, Shamkant Navathe, Hanghang Tong, and Duen Horng Chau. Visual graph query construction and refinement. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1587–1590. ACM, 2017.

[110] Johannes Pretorius, Helen C Purchase, and John T Stasko. Tasks for multivariate network analysis. In *Multivariate Network Visualization*, pages 77–95. Springer, 2014.

[111] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 262–270. ACM, 2012.

[112] Chotirat Ratanamahatana, Eamonn Keogh, Anthony J Bagnall, and Stefano Lonardi. A novel bit level time series representation with implication of similarity search and clustering. In *Advances in Knowledge Discovery and Data Mining*, pages 771–777. Springer, 2005.

[113] Donghao Ren et al. Squares: Supporting Interactive Performance Analysis for Multiclass Classifiers. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):61–70, 2017.

[114] Alexander Rind, Tim Lammarsch, Wolfgang Aigner, Bilal Alsallakh, and Silvia Miksch. Timebench: A data model and software library for visual analytics of time-oriented data. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2247–2256, 2013.

[115] Benedek Rozemberczki, Ryan Davies, Rik Sarkar, and Charles Sutton. Gemsec: Graph embedding with self clustering, 2018.

[116] Jeffrey M Rzeszotarski and Aniket Kittur. Kinetica: naturalistic multi-touch data visualization. In *Proceedings of ACM CHI '14*, pages 897–906, 2014.

[117] Ramik Sadana and John Stasko. Designing multiple coordinated visualizations for tablets. In *Computer Graphics Forum*, volume 35, pages 261–270. Wiley Online Library, 2016.

[118] Sebastian Schmidt, Miguel A Nacenta, Raimund Dachselt, and Sheelagh Carpendale. A set of multi-touch graph interaction techniques. In *ACM International Conference on Interactive Tabletops and Surfaces*, pages 113–116. ACM, 2010.

[119] Steven C Seow. *Designing and engineering time: The psychology of time perception in software.* Addison-Wesley Professional, 2008.

[120] Zeqian Shen, Kwan-Liu Ma, and Tina Eliassi-Rad. Visual analysis of large heterogeneous social networks by semantic and structural abstraction. *IEEE transactions on visualization and computer graphics*, 12(6):1427–1439, 2006.

[121] Jitesh Shetty and Jafar Adibi. The enron email dataset database schema and brief statistical report. *Information sciences institute technical report, University of Southern California*, 4(1):120–128, 2004.

[122] Ben Shneiderman. Response time and display rate in human performance with computers. *ACM Computing Surveys (CSUR)*, 16(3):265–285, 1984.

[123] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343. IEEE, 1996.

[124] Patrice Y Simard, Saleema Amershi, David M Chickering, Alicia Edelman Pelton, Soroush Ghorashi, Christopher Meek, Gonzalo Ramos, Jina Suh, Johan Verwey, Mo Wang, et al. Machine teaching: A new paradigm for building machine learning systems. *arXiv preprint arXiv:1707.06742*, 2017.

[125] M Sklar. Fonctions de repartition an dimensions et leurs marges. *Publ. inst. statist. univ. Paris*, 8:229–231, 1959.

[126] Greg Smith, Mary Czerwinski, Brian Meyers, Daniel Robbins, George Robertson, and Desney S Tan. Facetmap: A scalable search and browse visualization. *IEEE Transactions on visualization and computer graphics*, 12(5):797–804, 2006.

[127] Snappy data. `https://www.snappydata.io/`. Accessed: 2019-11-02.

[128] Chris Stolte, Diane Tang, and Pat Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Trans. Vis. Comput. Graph.*, 8(1):52–65, 2002.

[129] Yizhou Sun and Jiawei Han. Mining heterogeneous information networks: principles and methodologies. *Synthesis Lectures on Data Mining and Knowledge Discovery*, 3(2):1–159, 2012.

[130] Tableau. `http://www.tableau.com`, 2017. Accessed: 2017-01-01.

[131] Nesime Tatbul. Tools for Advanced Time Series Analytics: Enabling the Future. In *CIDR Conference*, 2019.

[132] Nesime Tatbul et al. Precision and Recall for Time Series. In *NeurIPS Conference*, pages 1924–1934, 2018.

[133] Sean J Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

[134] John Thompson, Arjun Srinivasan, and John Stasko. Tangraphe: interactive exploration of network visualizations using single hand, multi-touch gestures. In *Proceedings of the 2018 International Conference on Advanced Visual Interfaces*, page 43. ACM, 2018.

[135] TPC-DS. `http://www.tpc.org/tpcds/`, 2016. Accessed: 2019-11-02.

[136] TPC-H. `http://www.tpc.org/tpch/`, 2016. Accessed: 2019-11-02.

[137] Stef Van den Elzen and Jarke J Van Wijk. Multivariate network exploration and presentation: From detail to overview via selections and aggregations. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2310–2319, 2014.

[138] Frank Van Ham and Adam Perer. "search, show context, expand on demand": supporting large graph exploration with degree-of-interest. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):953–960, 2009.

[139] Han Varian. Hal varian on how the web challenges managers. `https://www.mckinsey.com/industries/high-tech/our-insights/hal-varian-on-how-the-web-challenges-managers`. Accessed: 2019-04-27.

[140] VerdictDB. Verdictdb. `https://www.verdictdb.com`. Accessed: 2018-05-30.

[141] Michail Vlachos, Marios Hadjieleftheriou, Dimitrios Gunopulos, and Eamonn Keogh. Indexing multidimensional time-series. *The VLDB Journal—The International Journal on Very Large Data Bases*, 15(1):1–20, 2006.

[142] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. Discovering similar multidimensional trajectories. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 673–684. IEEE, 2002.

[143] Tatiana Von Landesberger, Arjan Kuijper, Tobias Schreck, Jörn Kohlhammer, Jarke J van Wijk, J-D Fekete, and Dieter W Fellner. Visual analysis of large graphs: state-of-the-art and future research challenges. In *Computer graphics forum*, volume 30, pages 1719–1749. Wiley Online Library, 2011.

[144] Xiaojun Wan, Jianwu Yang, and Jianguo Xiao. Towards a unified approach to document similarity search using manifold-ranking of blocks. *Information Processing & Management*, 44(3):1032–1048, 2008.

[145] Martin Wattenberg. Sketching a graph to query a time-series database. In *CHI'01 Extended Abstracts on Human factors in Computing Systems*, pages 381–382. ACM, 2001.

[146] Martin Wattenberg. Visual exploration of multivariate graphs. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 811–819. ACM, 2006.

[147] Ryen W White and Resa A Roth. Exploratory search: Beyond the query-response paradigm. *Synthesis lectures on information concepts, retrieval, and services*, 1(1):1–98, 2009.

[148] Krist Wongsuphasawat, John Alexis Guerra Gómez, Catherine Plaisant, Taowei David Wang, Meirav Taieb-Maimon, and Ben Shneiderman. Lifeflow: visualizing an overview of event sequences. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1747–1756. ACM, 2011.

[149] Ka-Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. Faceted metadata for image search and browsing. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 401–408. ACM, 2003.

[150] Ji Soo Yi, Youn ah Kang, John Stasko, and Julie Jacko. Toward a deeper understanding of the role of interaction in information visualization. *IEEE transactions on visualization and computer graphics*, 13(6):1224–1231, 2007.

[151] Emanuel Zgraggen, Steven M Drucker, Danyel Fisher, and Robert Deline. (s— qu) eries: Visual regular expressions for querying and exploring event sequences. 2015.

[152] Emanuel Zgraggen, Alex Galakatos, Andrew Crotty, Jean-Daniel Fekete, and Tim Kraska. How progressive visualizations affect exploratory analysis. *IEEE transactions on visualization and computer graphics*, 23(8):1977–1987, 2017.

[153] Emanuel Zgraggen, Robert Zeleznik, and Steven M Drucker. Panoramicdata: Data analysis through pen & touch. *IEEE transactions on visualization and computer graphics*, 20(12):2112–2121, 2014.

[154] Jian Zhao, Fanny Chevalier, and Ravin Balakrishnan. Kronominer: using multi-foci navigation for the visual exploration of time-series data. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1737–1746. ACM, 2011.

[155] Jian Zhao, Fanny Chevalier, Emmanuel Pietriga, and Ravin Balakrishnan. Exploratory analysis of time-series with chronolenses. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2422–2431, 2011.

[156] Jian Zhao, Christopher Collins, Fanny Chevalier, and Ravin Balakrishnan. Interactive exploration of implicit and explicit relations in faceted datasets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2080–2089, 2013.