JOHANNES KEPLER
UNIVERSITY LINZ

Author
Stefan Luger, BSc

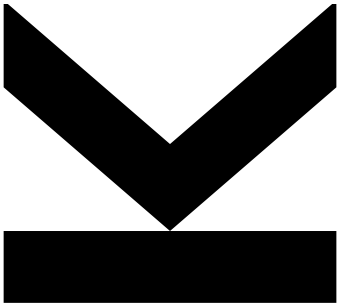
Submission
Institute of Computer Graphics

Thesis Supervisor
Dipl.-Ing. Dr.techn. Marc Streit

Assistant Thesis Supervisor
Nils Gehlenborg, PhD

February 2016

Interactive Visualization of Provenance Graphs for Reproducible Biomedical Research



Master's Thesis
to confer the academic degree of
Diplom-Ingenieur
in the Master's Program
Computer Science

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Str. 69
4040 Linz, Austria
www.jku.at
DVR 0093696

Abstract

Recent advances in cancer research potentially translate into new drugs and treatment methods. Through new sequencing technologies, cancer researchers can now acquire the biomedical samples needed for pre-clinical studies more quickly. A study often requires the integration of many external scripts and programs, resulting in numerous workflows that contain hundreds of tools and files. Due to the sheer size and complexity of such data-intensive experiments, many labs struggle to replicate and guarantee the trustworthiness of the published results. Recent efforts, such as *The Refinery Platform*, emerged to manage biomedical data, support automated scientific workflow execution, and provide the visual exploration tools that let analysts review their results and ensure the reproducibility of a study. Meta-data and the collected provenance information about files, tools, and analysis results of every workflow execution over time are typically stored in a provenance graph for each study. Existing provenance graph visualizations are mostly static, not on par with modern visualization techniques, and do not scale well. The thesis' goal is to deliver an interactive provenance graph visualization for Refinery that handles provenance graphs at the file level over time. In collaboration with the Refinery team, we elicited the six most important user tasks and requirements. The two major challenges when dealing with provenance graphs are to handle large and quickly growing graphs while they simultaneously evolve over time. We address these issues by using hierarchical aggregation to organize the graph into multiple hierarchy levels. Based on network motifs, we further aggregate similar and redundant analyses, which adds an even more abstract representation on top of this hierarchy. The graph dynamically adjusts the visibility of these levels for every node based on user interest through the application of a modular degree-of-interest function. Our combined approach enables researchers to review and communicate time-varying provenance graphs containing numerous workflows of hundreds of tools and files. The visualization was evaluated using real datasets acquired from the *Stem Cell Commons* database. The provenance graph grew up to 100 analyses, resulting in more than 1100 files and tools. We demonstrate the effectiveness of our visualization by means of task-driven use cases.

Keywords: dynamic graph drawing, hierarchical aggregation, motif-based aggregation, degree-of-interest, reproducible biomedical research, provenance

Zusammenfassung

Die jüngsten Fortschritte in der Krebsforschung sind von großer Bedeutung für die Herstellung neuer Medikamente und Behandlungsmethoden. So gelangen Krebsforscher durch neue Sequenziertechnologien schneller an präklinische Studien. Häufig benötigt eine Studie die Integration von verschiedenen externen Skripten und Programmen, welche wiederum in zahlreichen Workflows mit hunderten Tools und Dateien resultieren. Aufgrund der Größe und Komplexität dieser daten-intensiven Experimente, haben viele Labore Probleme die Ergebnisse zu replizieren und können folglich nicht mehr für die Vertrauenswürdigkeit der publizierten Resultate garantieren. Die Entwicklung neuer Software-Systeme, wie beispielsweise *The Refinery Platform*, gewährleisten die Reproduzierbarkeit der Studien durch die Verwaltung der biomedizinischen Daten, automatisierte Ausführung wissenschaftlicher Workflows und die Bereitstellung von Werkzeugen für die visuelle Datenanalyse. Bei jeder Workflow-Ausführung werden Metadaten und Provenance-Information von Dateien, Tools und Analyse-Ergebnisse in einem Provenance-Graph abgespeichert. Bestehende Ansätze für die visuelle Analyse der Provenance-Graphen sind meist statisch, nutzen weder die modernen Techniken der Visualisierung, noch skalieren sie gut. Der Schwerpunkt dieser Arbeit ist daher eine interaktive Provenance-Graph-Visualisierung für Refinery, auf Basis der gespeicherten Tools und Dateien des Provenance-Graphs. In Zusammenarbeit mit dem Refinery-Team haben wir die sechs wichtigsten Anwendungsfälle und deren Anforderungen ermittelt. Die beiden größten Herausforderungen für die Visualisierung von Provenance-Graphen sind eine gute Skalierbarkeit hinsichtlich der Größe und der Entwicklung des Graphen über die Zeit. In unserer Lösung unterteilen wir den Graphen mittels hierarchischer Aggregation in mehrere Hierarchieebenen und fassen ähnliche und redundante Analysen durch motif-abhängige Aggregation in einer zusätzlichen abstrakten Ebene zusammen. Darauf basierend passt sich die Sichtbarkeit dieser Ebenen mithilfe einer Degree-Of-Interest Funktion dynamisch an das Interesse des Benutzers an. Der kombinierte Einsatz unserer angewandten Techniken ermöglicht Benutzern zeitabhängige Provenance-Graphen, welche aus zahlreichen Workflows und hunderten von Tools und Dateien bestehen, besser zu verstehen und zu präsentieren. Für die Auswertung der Visualisierung wurden reale Datensätze aus der *Stem Cell Commons*-Datenbank herangezogen. Der ausgewählte Provenance-Graph umfasste mehr als 100 Analysen mit 1100 Tools und Dateien. Den Erfolg unserer Visualisierung zeigen wir anhand der aufgestellten Anwendungsfälle.

Keywords: Darstellung von dynamischen Graphen, Hierarchische Aggregation, Motif-abhängige Aggregation, Degree-Of-Interest, Reproduzierbarkeit in der biomedizinischen Forschung, Provenance

Acknowledgements

First and foremost, I would like to express my gratitude to my advisors. My greatest thanks go to Marc Streit who made this thesis possible as well as for his continued support and guidance throughout all stages of my work. I am really thankful to Nils Gehlenborg, who invested some extra hours to get myself familiar with the biomedical background and the Refinery framework. I would also like to take the opportunity to thank Oliver Bimber for granting me access to institute facilities.

I am very glad and grateful that both the Institute of Computer Graphics and Nils Gehlenborg gave me the opportunity to present my work at the IEEE VIS Conference on Information Visualization in Chicago—it was truly an extraordinary experience. Moreover, I was able collect invaluable feedback from domain experts which I could use in this thesis.

Furthermore, I want to thank both PhD students for their consistent and valuable feedback at the bi-weekly progress meetings. Holger Stitz, an experienced web developer, who could give me some excellent advices on how to tackle specific issues in web development. And Samuel Gratzl, thanks for the hours of discussions on how to solve some of the trickiest problems.

Special thanks goes to all the members of the Refinery development team. Especially Ilya, who helped me out on the deployment and integration side with Refinery.

Thanks to my colleagues who became friends. Leo, Stefan, and Manuel, these years of study would not have been the same without you.

Further I want to thank all my friends, and in particular Clez, Lewa, Richard, Thomas, Woifi, Alex, and Klaus for being the people that they are. Thanks for cheering me up when times are tough, but especially when they are good.

Thank you Manuela and Christian for your continued motivation and support. Finally, I am grateful to my parents Margarete and Robert, for enabling me my studies.

Contents

Chapter 1

Introduction

Today's better medical care for stem cell related diseases such as cancer is the direct result of advances in drug development and new treatment methods. Spearheaded by next-generation sequencing technologies, that accelerate the data acquisition of biomedical samples, researchers are now able to carry out studies that lay the necessary groundwork responsible for this achievement more frequently. The analysis of such heterogeneous data typically utilizes bioinformatics workflows that contain many distinct steps and tools. Moreover, the complexity increases with sample size and repeated workflow execution. In recent years, it became increasingly difficult for researchers to maintain, compare, and reproduce such analytical studies. As a consequence, scientific workflow systems emerged to address these issues. They provide meta-data annotations and automatically collect provenance information about files, tools, and their operating workflows for every step, decision, and changes made by analysts. However, they lack visual exploration tools, with which analysts would be able to explore and navigate the provenance of a study, enabling them to review results, ensure reproducibility, or even discover new insights in a visual environment. Efforts to create provenance visualizations have been made, but the existing solutions are not on par with modern visualization techniques. They do not scale well for hundreds of workflows as their solutions are mostly static and lack novel user interaction tools.

1.1 Motivation and Background Information

In the following sections, we provide essential background information on reproducible biomedical research and explain the meaning of provenance. These topics give a deeper understand why biomedical research is dealing with a crisis. We also introduce the necessity of scientific workflow systems. What motivates data-management, analysis, and visualization systems such as *The Refinery Platform*? Why and how can visual provenance exploration tools help solve important problems in biomedical research?

1.1.1 Biomedical Research

In 2003, the *Human Genome Project (HGP)*[1] was stated complete as the human Desoxyribonucleic Acid (DNA) was decoded. The DNA comprise the sequence of all base pairs for every gene in the human genetic code. It is considered the biggest achievement in biological science in human history. Many applied research fields such as (bio)medical research take advantage of these recent advances. Improved sequencing technologies and computational aid let scientists experiment with big data as it never could be done before. With the use of bioinformatics tools and interactive visualizations, analysts are finally able to process, present, and explore large datasets. The insights gained from explorative data analysis directly benefits (pre)clinical studies about drug development and new treatment methods. With cancer leading the charts on causes of death worldwide, the disease evolved into one of the most popular research topics to date. The *World Health Organization (WHO)*[2] also expects that the number of annual cancer cases will increase to about 22 million within the next two decades. In order to better understand the origin of this disease, the *The Cancer Genome Atlas (TCGA)*[3] project was initiated in 2005 and recently came to an end. Scientists tried to examine the genetic changes and mutations that are responsible for cancer cell growth. The project covered more than 20 cancer types with about 500 patient samples each. By comparing cancerous and normal tissue samples, researches set up a genomic profile for all cancer types. The gained results help to develop more individualized studies and in the long run personalize cancer treatment. One step and usually years before such study finds its application in real world scenarios, pre-clinical (testing on non-human organisms) and clinical trials (testing in humans) must be conducted and approved.

1.1.2 The Reproducibility Crisis

Today's record breaking number of cancer survivors are a result of robust cancer treatment methods made possible through the continuous advances in pre-clinical studies. While our knowledge on this disease broadens, scientists often struggle to provide documented and transparent studies for their findings. In Begley *et al.* [BE12], scientists attempted to reproduce 53 published findings related to drug development research. Unfortunately only six pre-clinical studies could be reproduced and confirmed. Whether the results where reproducible or not, patients volunteered in clinical studies that maybe caused side effects and probably never were to work. With the patient's determination to take a risk by volunteering for new therapies, the standard in pre-clinical research has to be raised.

Three years later Begley *et al.* reviewed the ongoing reproducibility crisis in biomedical research again [BI15]. Since his first publication the awareness within the scientific community has been risen greatly. Begley *et al.* urge funding agencies and institutions

[1] `http://www.genome.gov/10001772`
[2] `http://www.who.int/`
[3] `http://cancergenome.nih.gov/`

to reward reproducible and robust rather than flashy studies. Stakeholders must care for open data access, guidelines and good quality science in general. Eventually it will yield higher quality results in the long run.

With efforts of open-science organizations such as *Science Exchange*, *Center for Open Science* and recently the *Reproducibility Project: Cancer Biology* [Kai15][4], independent labs and university core facilities tried to replicate the results of 50 key cancer papers. The project earned a lot of controversial criticism by the listed authors as they felt insulted by letting scientists without expert knowledge reproduce and scrutinize their pre-clinical results. Although the authors are partly right, details given on experiment parameters and the conductive process itself should suffice to confirm at least the key conclusions of a paper, even when replicated by independent contract labs.

1.1.3 Scientific Workflow Systems

Most certainly, the cause of the crisis leads back to sheer amount of data that needs to be processed. For example, the TCGA project examines 500 samples for each of the 20 cancer types. The data collected alone sums up to about ten thousand heterogeneous datasets. Analysts, who experiment with this data regularly, experiment on studies containing thousands of processes (e.g., scripts or manual inputs). Usually the analysts are not computer scientists themselves but rather researchers in the fields of molecular biology. In order to deal with hundreds of workflows, which again contain dozens of tools and results, analysts are in the need of software suits which can handle large and complex experiments for them. Such systems are referred to as workflow management systems which support the creation, execution and monitoring of scientific workflows. More specialized scientific workflow systems are for example *VisTrails* [BCC+05, CFS+06, FSC+06, FS12][5], *Kepler* [ABJ+04, ABJF06, LAB+06][6], *Taverna* [WHF+13][7], *Galaxy* [BKC+01, GRH+05, GNT10][8], or *GenePattern* [RLG+06][9]. They were built to manage, track, share, and conduct large scale scientific studies. These systems share some common terms such as an analysis that is the product of an executed workflow template. Every workflow is executed upon a series of input files, contains execution parameters, and produces result files. A typical study often counts hundreds of such analyses that are connected to each other in a combined graph. Instead of manually transferring the results from one program or script to another, these systems automate this process and as a result save time and minimize the risk of manual errors.

[4] `https://osf.io/e81xl/`
[5] `http://www.vistrails.org/`
[6] `https://kepler-project.org/`
[7] `http://www.taverna.org.uk/`
[8] `https://galaxyproject.org/`
[9] `http://www.genepattern.org/`

1.1.4 Provenance to the Rescue

Scientific workflow systems offer fully automated analysis processing pipelines and with the addition of meta-data, which describes the steps how a result was produced, researchers are now able to handle large genomic data generated by dozens of connected workflows.

> *"The two important features of the provenance of a data product are the ancestral data product(s) from which this data product evolved, and the process of transformation of these ancestral data product(s), potentially through workflows, that helped derive this data product."* [SPG05]

This descriptive information is called data provenance and means lineage or pedigree information about a data result. In general, it captures (1) every process involved in the processing pipeline at any stage, (2) input files for each process, and (3) the produced results [DF08]. Many have defined provenance to the notion of data product causality, but provenance means more than just the lineage of a file.

> *"Specifically, provenance should include not only the standard data lineage information but also information about the context in which the workflow was used, execution that processed the data, and the evolution of the workflow design."* [ABJF06]

Hence, the term provenance also includes information about executed workflows in different levels of detail and over time—the evolution and changes during the development of a new study. Ragan *et al.* [RESC15] reviewed 50 papers referring to provenance in the visual analytics domain. According to their categorization of provenance by *type* and *purpose*, a biomedical study, as introduced in this chapter, operates on *data provenance*. Data provenance helps to *recall* the analysis history, *replicate* intermediate and final results, and *communicate* the development of the whole study over time. And more specifically, data provenance captures (1) every workflow and all processes and tools including all configuration parameters (e.g., execution date and time, author- and ownership, version number), (2) all datasets and input files, (3) every intermediate and final analysis result.

Benefits

Provenance provides many benefits to researchers who run their studies using workflow systems [DF08]. The additional level of documentation serves as a guideline for others to not only help validate, but most importantly, let them reproduce the published studies. When a study is reproduced, both the original and the replicate are investigated at the highest level of detail. Their analysis graphs should only differ in time or by its creators. In case a study has been modified, provenance tracks these changes at the workflow level including parametrization and all results to help the observer understand the cause. Further, provenance is mostly represented as a graph data structure which makes it easier to investigate the causality, the derivation of tools and files alongside a path. Analysts interactively explore a provenance graph to answer the questions: who, what, where, when and

why. Provenance also offers to reuse stored workflows for re-executions (also referred to as workflow *reruns*) in or outside the current study. Over the course of multiple workflow reruns, provenance keeps track of the different inputs and parameters used to produce each individual result. When developing new studies, many iterations of such reruns might occur until a desired result is achieved. Given that the varying inputs and parameters for each iteration are stored in the provenance model, the recurring and actual workflow only need to be stored once. Whereas general provenance is generated automatically, analysts should also be able to manually annotate their own decisions or add text notes.

Provenance to Ensure Reproducibility

Sandve *et al.* established ten rules on how to utilize provenance in scientific workflow systems to ultimately ensure reproducibility [SNTH13].

1. *For Every Result, Keep Track of How It Was Produced*
   The main idea of data provenance is to capture every computational step, including pre- and postprocessing, the execution configuration, and any other details involved in the result creating process. Workflow reruns generate multiple iterations of execution with each of them having their distinct input and parameter configuration. Former workflow runs may produce different results and thus need to be stored.
2. *Avoid Manual Data Manipulation Steps*
   The purpose of scientific workflow systems is to fully automate data manipulation steps within workflows. Each tool must provide a standardized interfaces for data input, execution parameters, and results. This requirement applies to any data processing step to maintain reproducibility, efficiency, and to remove the risk of human error. Nevertheless, manual text annotations are encouraged as they do not interfere with the computational tasks but rather add potential important information.
3. *Archive the Exact Versions of All External Programs Used*
   Workflows may change over time as their tools are being regularly updated. Newer tools may change standardized file formats or dependencies. It is inevitable to backup earlier tool versions for the case it is required again.
4. *Version Control All Custom Scripts*
   Extending rule 3, workflow provenance needs to keep track of the changes and by request a version control system must allow a rollback to the tool version needed.
5. *Record All Intermediate Results, When Possible in Standardized Formats*
   Through standardized file formats, low-level text and string comparisons are made possible. These comparisons can be measured with metrics that help to quantify the relevance of a change.
6. *For Analyses That Include Randomness, Note Underlying Random Seeds*
   In order to eliminate uncertainty, random data must be recorded and stored for reuse in the replication process.

7. *Always Store Raw Data behind Plots*
   In order to create visual representations of large datasets, raw data is often preprocessed into a model with less complexity. It is important, that the underlying raw data is immutable and the actual data transformation process is documented.
8. *Generate Hierarchical Analysis Output, Allowing Layers of Increasing Detail to Be Inspected*
   This rule is similar to rule 1 with the addition of providing a variable level of detail when necessary. Provenance heavily depends on the granularity of the data or process that is described. A final analysis step might contain multiple reruns over time, and in the end just one is picked out of a dozens. It is important to initially present an overview of high-level results. In turn, it is important to let fine-grained details such as raw files or other run-specific attributes be explored on demand.
9. *Connect Textual Statements to Underlying Results*
   Manual annotation may help others to better understand a certain outcome. It is important that claims are linked to their underlying results.
10. *Provide Public Access to Scripts, Runs, and Results*
    Key to reproducibility is transparency, meaning that all versions of data, scripts, parameters, and results within a study are made accessible to the public or at least privately shared with reviewers.

Each rule or guideline is significant to the implementation, visualization, and analysis of the provenance in order to make science more reproducible in the future.

Challenges and Opportunities

Having discussed the benefits of provenance thoroughly, some open problems as pointed out by Davidson *et al.* in [DF08] have to be addressed.

Scientific workflow systems still have room for improvement. Although they manage large and heterogeneous data well in its raw state, data transformation processes including provenance generate a multiple of it and cause a study to grow quickly. We need smart solutions to maintain scalability (e.g., prune data and provenance overhead).

Studies often include collaborators in multiple research labs, resulting in distributed work and data. Different systems may use other methods of collecting and annotating provenance. In order to combine distributed study resources a standardized provenance model is required. Institutions such as the *Provenance Challenge* created *The Open Provenance Model (OPM)*[10], or the *World Wide Web Consortium (W3C)* who created *PROV*[11].

It is important to track provenance for both workflows and their underlying files that are stored in a database. Database entries might get manipulated manually at the database

[10] `http://openprovenance.org/`
[11] `http://www.w3.org/2011/prov/`

level too. Therefore, it is suggested to connect provenance across databases and workflows to keep track of all inputs and results at both database and workflow system level.

The biggest benefit and motivation for this thesis is the potential of visual provenance exploration and navigation. More insightful visualizations would enable researchers to better understand their results or gather new insights from workflow runs. Aside the discovery aspect, novel visualization tools could support analysts to present a simplified proof of success of their study, ultimately ensuring reproducibility. However, the workflow systems we introduced in Section 1.1.2 lack the visualization tools to fully accomplish such tasks.

1.1.5 The Refinery Platform

The *Refinery Platform* [ref15][12] (in the following referred to as *Refinery*) is a web-based data management, analysis and visualization system that supports reproducible biomedical research by collecting provenance about scientific workflow executions of heterogeneous datasets in large genomic studies. It is developed by the *Park* and *Gehlenborg Labs* at *Harvard Medical School*[13] in collaboration with the *Hide Lab* at *Harvard T.H. Chan School of Public Health*[14].

Refinery, in combination with the contribution of this work, specifically aims to present complex provenance through the support of visualization tools. Additional goals of Refinery are to share provenance, datasets, and results in public and private groups. Furthermore, Refinery aims to support seamless workflow execution on top of provenance visualization tools to rerun previously executed analyses with modified input files or workflow parameters.

Key application fields for Refinery are projects like the *Stem Cell Commons*[15]. Developed by the *Harvard Stem Cell Institute (HSCI)*[16] it is already using milestone releases of Refinery to manage their data repositories on stem cell biology to develop new treatments and cures for a wide range of genetic diseases (e.g., Cancer, ALS, Alzheimer, Diabetes). The Stem Cell Commons database contains hundreds of stem cell experiments, microarray and *Next-Generation Sequencing (NGS)* data from human, mouse, rat and zebrafish. Their cell types and disease models of them are linked and annotated with meta-data as well as provenance. This data typically is obtained by advanced methods in biotechnology such as *mRNA Expression*, *microRNA expression*, *DNA methylation*, *protein expression*, *copy number variants*, *mutation calls*, and *clinical parameters*.

[12] `http://refinery-platform.org/`
[13] `http://hms.harvard.edu/`
[14] `http://www.hsph.harvard.edu/`
[15] `http://stemcellcommons.org/`
[16] `http://hsci.harvard.edu/`

Repositories

Refinery imports datasets from tab-delimited text or *Investigation-Study-Assay (ISA-tab)* [RSBM+10][17] based files. ISA-tab is a hierarchically structured and general-purpose tabular format that enables descriptive annotation of studies to support reproducible biomedical research. As shown in Figure 1.1, Refinery's data model is built on top of ISA-tab to support meta-data annotation and provenance tracking.

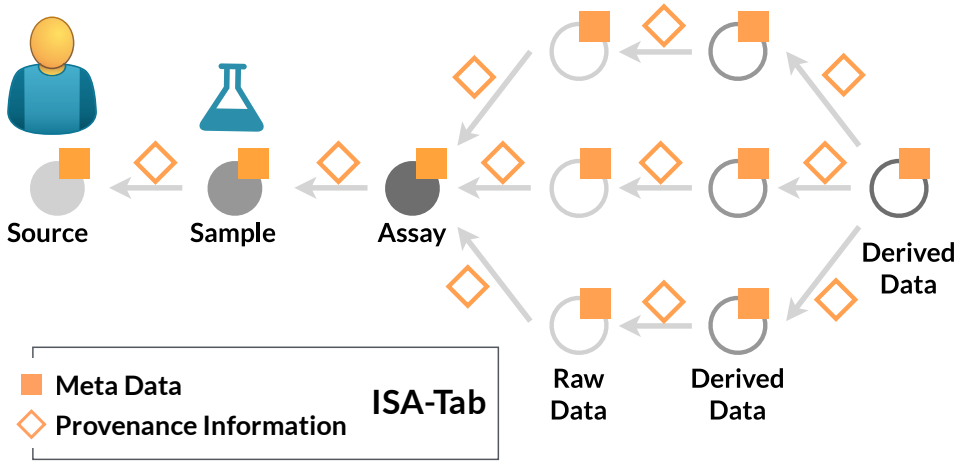


Figure 1.1: *The annotation of meta-data and provenance in Refinery enable users to understand the computational steps executed in an analysis pipeline that led to a certain result. The graph's nodes represent files such as raw data and derived results. The graph's edges represent the executed workflows (analyses) that manipulate and connect files. (Figure courtesy of Nils Gehlenborg.)*

Workflow Execution

While imported datasets already describe an experiment on their own, Refinery offers to run (or execute) workflow on the dataset through the *Galaxy* bioinformatics workbench [BKC+01, GRH+05, GNT10][18], resulting in an analysis creation. Workflow templates (also referred to as workflow types) are manually created with the Galaxy workflow editor and imported back into Refinery. Figure 1.2 shows a typical Galaxy workflow that is visualized with Refinery's own workflow visualization. We have implemented the workflow visualization module for Refinery in the course of a preliminary project to this thesis.

Every workflow execution in Galaxy has their workflow template and input files assigned to it. Galaxy then processes the workflow step by step and produces the intermediate and final results that are sent back to Refinery (see Figure 1.4). From these files, Refinery

[17] `http://isacommons.org/` and `http://www.isa-tools.org/`
[18] `http://galaxyproject.org/`

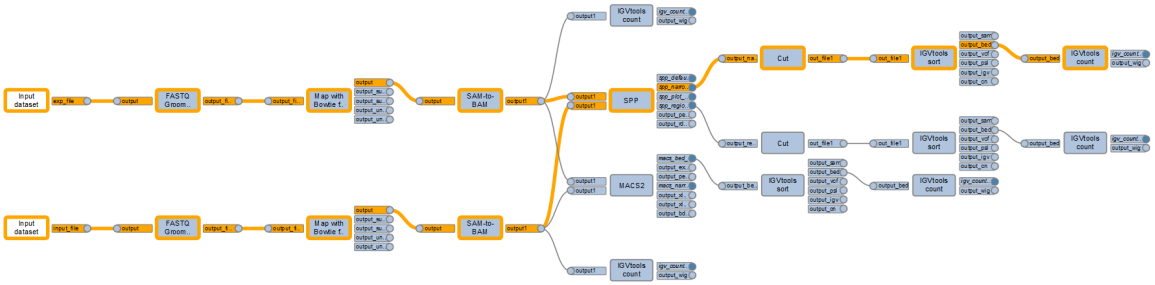


Figure 1.2: *A typical workflow is created in Galaxy, imported back to Refinery as workflow template, and rendered as node-link diagram. This workflow takes* 2 *inputs (white nodes at the left) and produces* 10 *outputs (dark-blue colored circles). The bigger rectangles represent tools while the slim ones represent the in- or output files.*

generates an analysis and adds it to the experiment graph (as described in Figure 1.1). Every workflow execution is automatically annotated with meta-data and provenance about files, workflow, execution time, creator of the workflow, and tool versions. Every analysis is strictly restricted to exactly one workflow template, but may contain multiple subanalysis where the same workflow is executed for every set of input files. Imagine a simple workflow template which takes 2 input files and produces 10 output files. Applying 8 inputs means that the analysis would result in $8/2 = 4$ subanalyses and $10 * 4 = 40$ outputs (see Figure 1.2). These are basically the core assets of Refinery (datasets, analyses, and workflows) which are depicted in Figure 1.3, showing the Refinery landing page in a web browser.

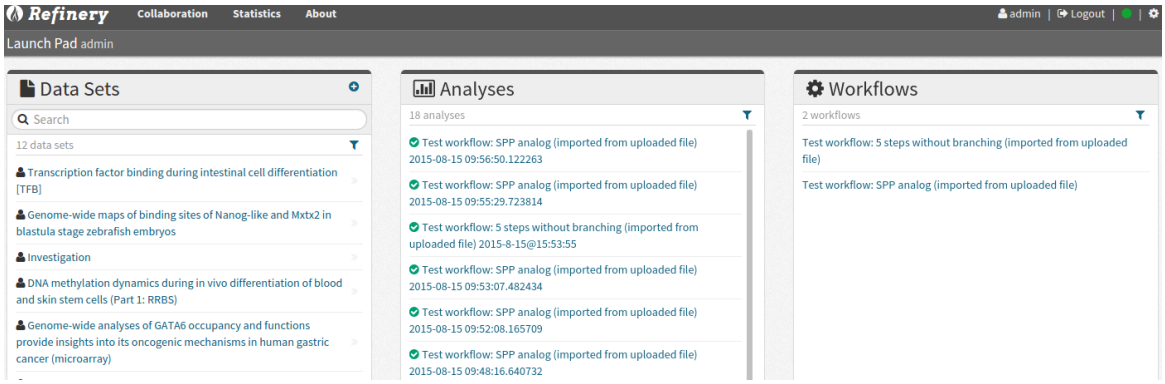


Figure 1.3: *A screenshot of the Refinery landing page lists imported datasets on the left, analyses in the middle and workflow templates on the right.*

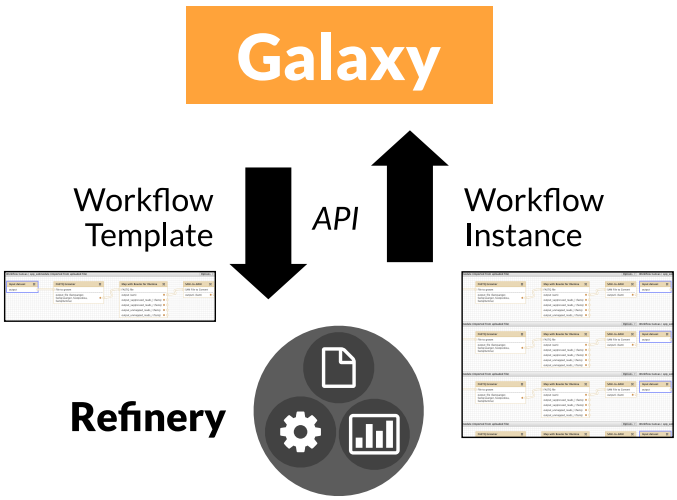


Figure 1.4: *Galaxy workflows are imported into Refinery, executed in Galaxy, and the workflow execution results are sent back to Refinery again. (Figure courtesy of Nils Gehlenborg.)*

Prerequisite Terms

The provenance graph in Refinery is organized into three **hierarchy levels**, which we also refer to as graph node types throughout this thesis: An **analysis** is the product of a workflow execution (or run) that includes one or multiple **subanalyses** where each of them again contains exactly one **workflow** instance (see Figure 1.5a). Note, within the hierarchy of a single analysis, all subanalysis children are restricted to the same workflow template. A workflow comprises **files** and **tools** that represent the atomic building blocks. Whereas analyses represent the most abstract view and the lowest level of detail, workflow files and tools reveal the highest (e.g., files carry a set of attributes and are annotated with meta-data; tools connect incoming with outgoing files and contain specific tool state parameters). Connecting the analyses through their workflow inputs and outputs generates a **provenance graph** on top of the study as illustrated in Figure 1.5b.

1.2 Goals and Contribution

The thesis proposes an interactive provenance graph visualization on top of Refinery's provenance data model. The key objective is to make the visualization scale to complex provenance graphs at all stages over time. We address this by the combined use of multiple visualization techniques among multiple views. Ultimately, the provenance visualization should enable analysts to review and ensure reproducibility of their studies. Letting them navigate and investigate provenance, they can get a better understanding on how a study develops over time.

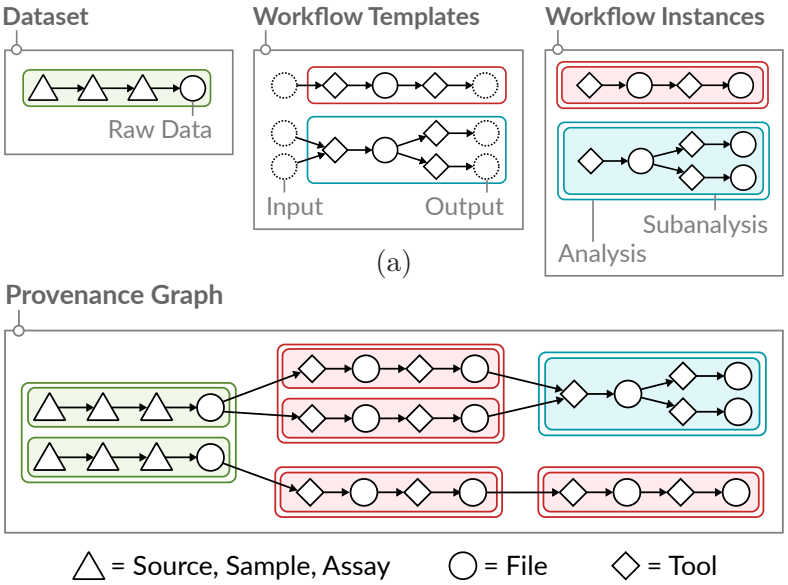


Figure 1.5: *The provenance graph in Refinery consists of connected workflow instances over time. (a) The workflow instance is the product of a workflow execution that conforms to a single workflow template of files (inputs, outputs, and intermediate results) and tools. A single workflow instance is also referred to as subanalysis, where all subanalyses which were produced at the same workflow execution (in Galaxy), are part of the same analysis. (b) Adding workflow instances to the raw dataset or previously added analyses generates the provenance graph.*

In Section 1.1.4 we discussed the challenges and opportunities on visualizing provenance. According to [DF08], the problem of mining and extracting knowledge from provenance data has been largely unexplored. The state-of-the-art of provenance visualizations substantiates this statement. Existing solutions lack basic focus+context techniques and are implemented as static node link diagrams. In a single view, the provenance graph is laid out in its highest level of detail, showing every workflow in a single view all at once. With overly heavy use of text labels, they struggle to scale well for hundreds of nodes and edges. All these issues create visual complexity that results in visual clutter, making it impossible to understand and explore the graph at moderate size. As in Refinery, the provenance graph is growing quickly, resulting in hundreds of workflow files and tools.

Hence, we need to make use of semantic aggregation techniques (e.g., hierarchical or temporal) with which we are able to compress the graph. Initially, the analyst is presented with an abstract but sufficient overview of the graph at manageable size. We have to compute an aesthetically pleasing layout that responds to dynamic changes when navigating through the graph's hierarchy levels. A degree-of-interest function incorporates user interest and hence is well suited to dynamically adapt the level of semantic aggregation to each node in the graph. Another common visualization technique to add details without the

use of text labels is to encode attributes (e.g., time) as color, shape and size—as visual variables.

Through the course of long-term experiments workflow reruns result in file, tool, and parameter changes over time. We pinpoint these changes and introduce measurements to quantify their significance and meaning. For that purpose, we propose to stack similar analyses, which only differ in time but share the same workflow, on top of each other. With this motif-based aggregation technique, redundant reruns are summarized while dissimilarity is indicated through separation.

Refinery annotates files with meta-data attributes and provides a facet-browsing interface as well as a matrix visualization to simplify browsing through large and heterogeneous datasets. We integrate the browsing interface to allow attribute based filtering of the provenance graph.

The thesis does not include actionable provenance, meaning user-driven workflow executions by modifying parameters or previous results, both within the provenance visualization interface itself. Workflow editing could significantly speed up exploration tasks as workflow changes could be applied to existing templates inside the visualization interface. However, this feature is beyond the scope of this thesis.

While we provide meaningful change metrics for workflow reruns based on high-level characteristics, a comparison of the actual raw data is not considered.

Existing provenance visualizations do not satisfy the requirements stated above due to the lack of a well thought out provenance model or their limited set of visualization techniques. The proposed provenance graph visualization contributes to visual presentation, navigation, and investigation tasks of workflow provenance while achieving the goals and requirements stated above.

Conference Contributions

In the course of this work, I co-authored several conference contributions. The collaborating authors are: Holger Stitz, Samuel Gratzl, Nils Gehlenborg, and Marc Streit.

- During the concept phase of this work, we submitted a poster to the annual *IEEE Conference on Visualization (InfoVis)*. The poster includes a mockup of our conceptual design (see Chapter 4). It was presented at *InfoVis '14* in Paris, France in November 2014 [SGL+14].
- At the later implementation stage of this work, we submitted a poster abstract to the annual *Symposium on Biological Data Visualization (BioVis)*. The poster demonstrates our overall concept and early implementation results (see Chapter 6) were presented in the form of screenshots at *BioVis'15* in Dublin, Ireland in July 2015 [LSG+15b].

- After the implementation was done, we submitted a poster to the annual *IEEE Conference on Visualization (InfoVis)*. The poster summarizes this work in an overview. It briefly describes user tasks (see Chapter 2), the main concept (see Chapter 4), and implementation results (see Chapter 6). Our submission had been accepted and awarded with the best poster of the conference (of 81 submissions). I have presented our work at *InfoVis '15* in Chicago, Illinois, USA in October 2015 [LSG+15a].
- To conclude our work, we also submitted a full paper to the annual *VGTC Conference on Data Visualization (EuroVis)*, which is currently under review. We described our work in the form of a design study paper that gives an overview on all chapters of this work. Most importantly, the effectiveness of our solution is demonstrated by the means of a biomedical case study.

1.3 Outline

The thesis is structured as follows. In collaboration with the Refinery team members we identified six user tasks in Chapter 2. Next, visualization principles, related work, and the status quo of provenance visualizations is discussed in Chapter 3. The core of this thesis is presented in Chapter 4, where we explain our applied aggregation techniques, the specification of a modular degree-of-interest function, and the multiple view concept. Chapter 5 briefly introduces Refinery from a technical perspective followed by our implementation details that introduce used technologies, the data model, and selected topics in detail. We discuss the implemented features, practical use cases, and limitations in Chapter 6. Chapter 7 concludes and discusses open opportunities of this work.

Chapter 2

Tasks and Requirements

One important stage in software engineering encompasses the identification and detailed documentation of software requirements—known as *requirements engineering*. A well documented and detailed list of requirements is critical to the success of every software product as it serves as foundation for the features elaborated in the concept design phase. Too often, requirements are not fully gathered and the development team has to revisit and update the initial requirements specification when they were implemented already. While it is nearly impossible to identify all requirements for a software product initially, spending more resources to carefully elicit requirements lowers the risk of delivering wrong features with the final product.

In collaboration with the Refinery team and with the input from biomedical researchers, we carefully formalized a set of user tasks to understand the role of the user, their typical workflow and goals. The people involved in the discussion represent stakeholders with expert knowledge in domains such as biomedical research, bioinformatics and software engineering. From a visualization perspective, a well-established approach for conducting design studies is to first transform the domain specific problem, e.g., gathered from user stories, into abstract form which can then be solved through visualization [Mun14]. This process is referred to as *task abstraction*. Task abstraction aids multiple roles, e.g., domain experts and software developers, to have the same understanding of the problem. An abstract definition of these tasks may also be used to propose different solutions to the same problem [SMM12]. In the following, we describe each user task as well as their functional requirements beneath.

Task I: High-Level Overview

Users want to start with an aggregated version of the provenance graph that shows the overall structure and size of the graph. They are especially interested in the workflows that were run, how they were configured, and at which point in time the resulting analyses were created.

As the topological order of the analyses graph is predefined, analyses cannot be ordered by time within the graph. Regardless of the size of the graph, an overview of the graph needs to be presented within the limited drawing space. This requires the combination of novel aggregation and zooming techniques.

Task II: Attribute Encoding

Analyses must show date and time, in- and output files, and the creators. Workflow files and tools also have a title, a type, and a name. These attributes should be encoded onto the nodes using visual channels such as color, shape or size. Users want to choose from a set of attributes that they can map onto the nodes.

Every workflow file and tool must store its execution and linkage information, or derive it from its topological parent. Glyph encoding must be self-explanatory, distinct for all node types, and leave enough space for text labels.

Task III: Filter

Users want to emphasize certain aspects of the graph, e.g., a specific attributes, workflow types, or provenance data changes. In addition, users want to filter specific analyses or every analysis within an adjustable time frame.

Filtering should not manipulate the overall graph layout. As the graph's order is restricted to its topological structure, it cannot be reordered by time. Hence, a separate view is required to allow for time threshold-based filtering.

Task IV: Drill-Down on Demand

While the initial overview of the graph is reduced due to limited drawing space (see Task I), users want to inspect detailed information on demand. They want to manually drill-down into subgraphs of interest. While navigating in the graph, the surrounding context should be kept in a compact representation.

Users need interactive tools to navigate across multiple hierarchy levels. Transitions must reflect the direction and different hierarchy levels must change their visual representation accordingly.

Task V: Investigate Changes

Regularly, analyses of similar configuration (in- and output files, workflow, parametrization) recur. Users want to explore, track, and understand the changes over time. In particular, they want to compare similar analyses by meaningful change metrics. Some questions such as the following should be solved: Which workflow parameters have changed? Which in- (and output) files have changed? Which changes affect a particular file or set of files?

Linkage information to preceding and succeeding, parent, and child nodes are required for the definition and computation of change metrics.

Task VI: Investigate Causality

Users want to investigate the derivation of files and steps leading to or from a selected node of choice. It is important that the highlighted path is kept in context and does not change the layout of the provenance graph.

The provenance visualization must provide interactions for every node to allow the highlighting in both directions.

These tasks will be used and referred to throughout this thesis, in particular when we discuss related work as well as our own solution.

Chapter 3

Related Work

This chapter expands on the work's rationale and introductory background knowledge given in Chapter 1 and elaborates the status quo of existing work. First, we introduce background information and the principles of visualization in Section 3.1 followed by visual encoding of multiple attributes in Section 3.2. Data provenance is stored in an experiment graph, thus in Section 3.3, graph representations and their general visualization approaches are explained. Sections 3.3.2 and 3.3.3 cover novel aggregation techniques that address scalability and usability. Section 3.3.4 introduces the latest techniques on visualizing time-varying graph data. In Section 3.3.5, the concept and benefits of using a degree-of-interest function is explained. We conclude this chapter by discussing existing provenance visualizations in Section 3.4.

3.1 Visualization Principles

3.1.1 Information Visualization

Visualization emerged as a subdiscipline of *Human Computer Interaction* and historically was subdivided into two separate fields. The first, *Information visualization*, deals with *abstract* data such as trees or graphs, while the second, *Scientific visualization*, focuses on data with a *spatial* context, where the space coordinates of the datasets are mapped onto the screen (e.g., geographical data) [SB03]. Munzner also added that the spatial representation for information visualization can be chosen, while for scientific visualization it is given [Mun08]. However, the distinct categorization into these subdomains should not be taken too strictly. Weiskopf *et al.* discussed the historic division on information and scientific visualization at a panel at the annual *IEEE Visualization*[1] conference in 2006 [WMK06]. In information visualization literature, abstract data seems to be the main point. However, as datasets are becoming increasingly large and complex, information visualizations may as well need to convert discrete or abstract data into continuous or spatial data, like it is common with scientific visualizations. Members of both subdomains suggest to rethink the division of both fields as they can benefit from each others' advances.

[1] `http://ieeevis.org/`

Nevertheless, this section covers the principles of information visualization that might be applied to both domains.

Information visualization has been defined in many ways [CMS99, OL03, SB03, TC05, CEH+09, WGK10, Car12]. They often emphasize different aspects of the research field which in turn may cause ambiguities. One well-established definition is the one by Card *et al.*:

> *"The use of computer-supported, interactive, visual representations of abstract data to amplify cognition."* [CMS99]

3.1.2 Visual Analytics

As of the definition of information visualization, abstract data is conveyed into interactive visualizations to amplify cognition by the means of understanding the underlying dataset, detecting patterns, or discovering new insights [Car12]. These tasks directly relate to the research field of *Visual Analytics*. It is closely related to information visualization by the means of gaining knowledge from data through analytical reasoning [TC05]. The core aspect of visual analytics is the interactive back and forth between visualization and analytics [SB13]. Keim *et al.* defined it as the goal of understanding, reasoning, and decision making [KKEM10]. Figure 3.1 illustrates this visual analytics process that exploits the benefits from visualizations and models to derive knowledge.

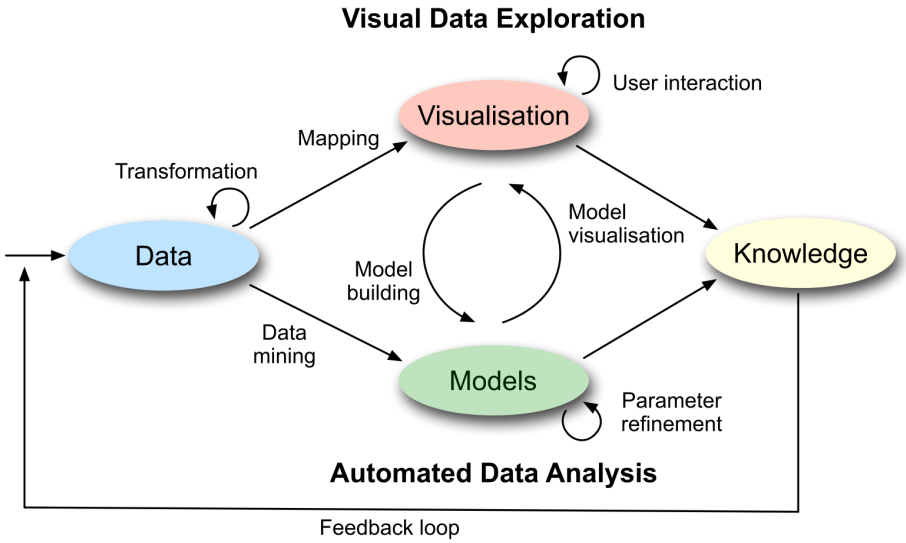


Figure 3.1: *The visual analytics process illustrates the tight coupling of visualization as well as model-based data analysis. (Figure from [KKEM10].)*

3.1.3 Data Types

Information visualization and visual analytics both process data, as seen at the left side in Figure 3.1. At the very beginning, raw data is recorded (e.g., measurements) in real world or generated artificially. An important characteristic of data is whether it has a value and can be ordered. Card *et al.* classified data types into the following three types [CM97]:

- *Nominal* data has no value and no order.
- *Ordered* data has a value which can be ordered.
- *Quantitative* data is numeric and allows for arithmetic operations.

Ward *et al.* later refined the definition of data types [WGK10]. They described ordered as *ordinal* data that can be binary, discrete or continuous. *Nominal* data was re-defined as categorical, ranked and arbitrary data. Further they suggested to add *scales* to cope with the size of raw data. Scales use a distance function to fit a range of data from data space into visualization space.

Another way to classify data is based on the dimensions and natural tasks applied to the type as proposed by [Shn96] as *type by task taxonomy*:

- *1-dimensional:* Linear data types.
- *2-dimensional:* Planar or map data.
- *3-dimensional:* Real-world objects.
- *Multi-dimensional:* Items with n attributes are mapped to n-dimensional space.
- *Trees:* Hierarchical data with single parent and arbitrary child relations.
- *Networks:* Arbitrary number of relations among items.
- *Temporal:* 1-dimensional and time-dependent data with a start and an end.

[Shn96] noted that the list does not cover all possible forms of data and suggested that every visualization rather uses a combination of them. In the context of this work, Refinery imports datasets that also contain meta-information extracted from the ISA-tab file format. This type describes the content, context, and structure of data [GS00]. Meta-data as well as images, videos, and text are missing in this classification hierarchies by Card *et al.* and Ward *et al.*. With the exception of meta-data, these types are of no particular interest to this thesis and therefore are not addressed any further. In Refinery, workflows and analyses also hold provenance information in the form of graph data. Obviously, these multitude of types can hardly be classified into a single type. They are rather represented by a mix of the following types: multi-attribute, directed acyclic graph, and temporal.

3.1.4 Interaction Techniques

Regardless of the classification of data, the major challenge of visualization is to map data into visual space. The goal is to provide a presentation of the data, including interaction, that enable users to carry out tasks to solve a certain problem. Usually, there is insufficient drawing space to visualize the whole dataset in its raw form. Thus, complex data is transformed first (e.g., dimensionality reduction) and presented in a high level overview (Task I) that reveals just enough information. For a summarizing overview, much of the data is hidden or just shown with less details. With interaction tools such as zoom and filter (Task III), users are able to adjust their viewport or filter out uninteresting items. In order to inspect details which are initially obscured by abstraction, users need to be able to further drill-down (Task IV) into regions they are interested in.

The above tasks resemble Shneiderman's visual design guidelines for visualization systems known as the *Visual Information Seeking Mantra* [Shn96]:

> *"Overview first, zoom and filter, then details-on-demand"*

The mantra describes the visual exploration approach from a top-down and task-based perspective. It begins with a complete but high-level overview first, then the user zooms in into regions of interest or filters out uninteresting items before finally selecting items that reveal details. The mantra originally was extended by three additional terms: *Relate* helps to discover relationships among different data items. When manipulating data in the course of visual exploration, a *History* allows to recover a previous state and *Extract* means to save discovered findings for later use. In the early history of visualization, the mantra started an ongoing discussion on visualization guidelines. In 2005, Craft and Cairns reviewed about 50 papers which have referenced the mantra frequently as a guideline, but not more. They argued that guidelines alone do not suffice and the mantra lacks methodological guidance for novices and experts [CC05].

A more recent classification approach was made by Pike *et al.* [PSCO09]. Their method underlines the point of task abstraction [Mun14, SMM12], which we briefly mentioned in Chapter 2. In Figure 3.2, high-level tasks such as browse, explore, and analyze depend on their underlying low-level processes such as retrieve value, filter, or sort. From the visualization viewpoint, representation and interaction intents also depend on low-level techniques. Select, query, pan and zoom, and others are the corresponding methods of the high-level intents such as select, encode, filter. Overall Pike *et al.* gave a comprehensive overview of tasks and interaction techniques commonly used in novel visualization systems. On top of Shneiderman's mantra and the review by Craft and Cairns [CC05], they describe the underlying low-level techniques and patterns required to achieve high-level user goals and tasks. In the following sections, we will discuss a collection of well known and contextually relevant interaction techniques.

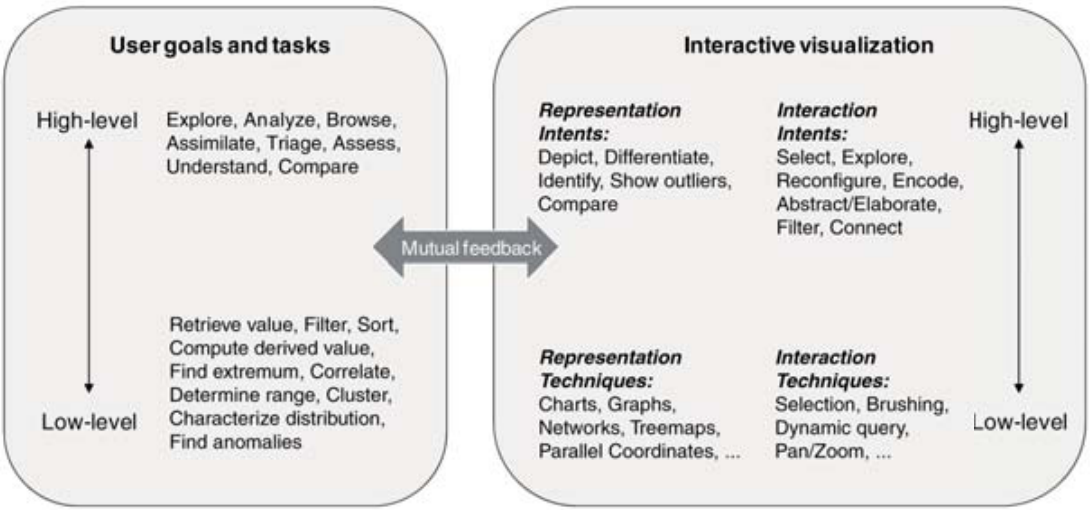


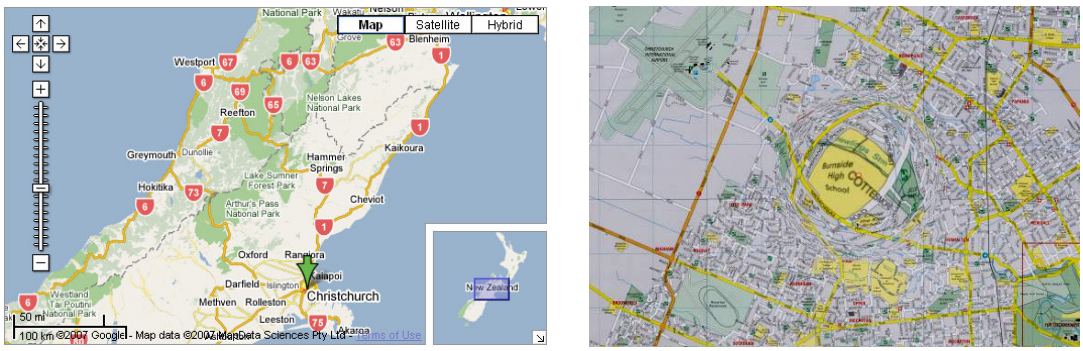
Figure 3.2: *Classification of user goals and tasks that relate to the interaction techniques in interactive visualization systems. (Figure from [PSCO09].)*

Focus+Context

Focus+context represents one of the most commonly followed principle in visualization. When datasets are large, an overview does not leave enough space to simultaneously show the details too. Naturally, one would zoom into a region of interest but in turn would lose the surrounding context due to limited visualization space. The trick here is to increase the level of detail within the focal area while simultaneously lowering the level of detail of the surrounding context [CMS99]. In contrast to overview+detail, both views are seamlessly integrated into a single view [CKB08].

Kosara and Hauser introduced four categories of focus+context visualizations [KHG03]:

Overview Methods In [Shn96], *overview first* means to present an initial overview of the dataset. The reason is that the overall structure may reveal patterns and themes which cannot be seen from other vantage points. A perspective onto the whole dataset helps to understand relationships between greater regions while obscuring details that otherwise would distract [CC05]. The overview window contains a movable field-of-view box which lets the user control an additional detail view—referred to as **Overview+Detail**. Acting like a camera, the inset allows for greater zoom levels projected onto the detail view. Both views are physically separated from each other but linked through the position of the inset. The user can interact with both views separately where the actions from one view manipulate the other simultaneously. The concept of both views, overview and detail, are shown in Figure 3.3a. Opposed to the following categories, this is the only one that is not seamless. More examples such as *lenses* or *z-separation* are reviewed in [CKB08].



(a) (b)

Figure 3.3: *(a) Overview+Detail example with a context inset in Google Maps[2]. (b) Focus+Context example in a map distorted by the fisheye mechanism. (Figures from [CKB08].)*

Distortion-oriented The visualization space is geometrically distorted by stretching the focal area while less but sufficient context remains. A classic example was described by Furnas *et al.* known as the fisheye view [Fur86]. It builds on the idea of assigning relevance to each data item via the application of a *degree-of-interest (DOI)* function (further discussed in Section 3.3.5). An example is shown in Figure 3.3b.

Filtering Filters can be used to selectively show or hide detail information for certain areas without distortion. Bier *et al.* proposed this technique in their work on magic lenses. They use a movable object on top of the visualization to display different information of data items that are currently within it [BSP+93].

In-Place Techniques Rather than changing the information, it is instead emphasized, for example by changing color, transparency, or size. Another technique known as *Semantic Depth of Field (SDOF)* indirectly creates a focus by blurring the surrounding context.

Filtering and in-place techniques are not identical to traditional filtering which is described in Section 3.1.4.

Panning and Zooming

Pan and zoom operations are not directly originating from the visualization domain. They change the viewpoint and therefore are equivalent to camera movement as known from basic photography [CMS99]. With panning, the user translates the position of the viewing window along two-dimensional coordinates. Zoom adds a third and spatial coordinate, letting users zoom into an area of interest or zoom out to get an overview. The zoom level

[2] `https://maps.google.com`

(or magnification) starts at 0 when zoomed out and might not be limited in the opposite direction. The object is geometrically resized by the current magnification.

Furnas and Bederson introduced a framework to help understand pan and zoom in multi-scale visualizations. They reduced the three-dimensional zoom and pan trajectory to two-dimensionsal *space-scale diagrams*: spatial (horizontal axis) and scale (vertical axis) [FB95]. The trajectories basically record the temporal and spatial position of the viewing window as seen in Figure 3.4.

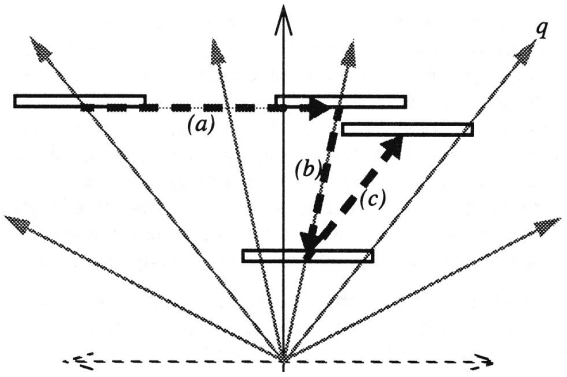


Figure 3.4: *First, (a) shows a pan trajectory, followed by zoom out action in (b), and ends with a pan and zoom movement in (c). (Figure from [FB95].)*

Semantic Zoom

Traditional zooming has a significant drawback as it is limited to geometric enlargement or reduction to the visualization space at the same time. No information except scale is gained from it. However, with the combination of semantic zoom, the level of detail of a visual object can be controlled by a predefined magnification. This method is commonly used to show less details at low magnification and enriched information vice versa. In the Google Maps example shown in Figure 3.3a, the inset at less magnification displays only a shape of New Zealand, whereas the detail view shows actual city names, streets and the topography. With this technique, objects are represented in different representations [Spe07, CKB08].

Filter and Details on Demand

The sections above introduced methods that handle multiple levels of detail well. But what if datasets are becoming increasingly large and complex so that even the said techniques cannot deal with them anymore? On the one hand, we need a solution to temporarily reduce the complexity of the dataset. And on the other hand, we need to present detail information for initially abstracted data. Shneiderman originally suggested filtering with widgets, sliders, or buttons to filter out unwanted items based on attributes. Filtering does indeed work well as the user can quickly trim the dataset to a new and partial overview

that again fits into the drawing space. The details (e.g., all attributes) of a data point can be shown by pop-up windows triggered upon selection [Shn96]. Craft and Cain specifically state quick access to details without changing the current view. In addition to conventional data point selection also mouse-over triggered tooltips would suffice [CC05].

A pioneering example for filtering and details on demand is the *Information Visualization and Exploration Environment (IVEE)* framework [AW95]. Based on *dynamic queries*, IVEE removes uninteresting data items and in turn reduces complexity in the current view. Another positive side-effect is that the query parameters (persistent in the query window) already carry a lot of detail information about the reduced view. From a data exploration perspective, users want to look up the attributes of some initial data items to get themselves an entry point in the narrowed view. Upon selection, details are shown in a pop-up window. Clicking on the displayed attributes again applies them to the query filter. This feedback loop significantly simplifies and enhances the data exploration experience. Not only is IVEE a good example for details on demand, it also illustrates the benefits gained from initial dataset reduction through attribute-based filtering.

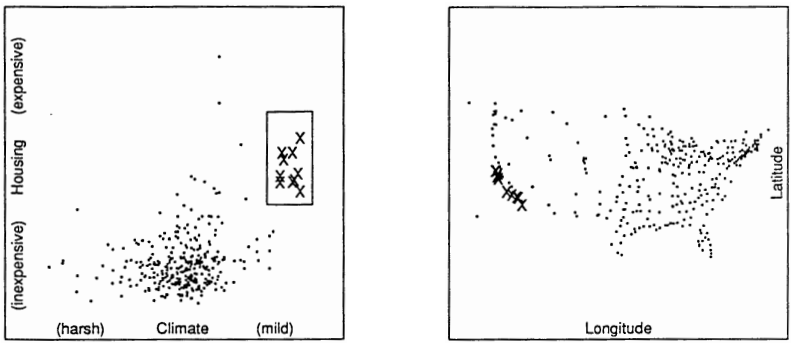
Multiple Coordinated Views

Introduced in the paragraph above, the IVEE main visualization view is connected to a tab that allows for query attribute adjustment and a pop-up window that shows details for a selected data item. All views are synchronized with each other. As a combined system they help to preserve context and provide more insights on the data [AW95]. The overview + detail example in Figure 3.3a combines the two graphical views focus and context. As of [BWK00], a multiple view system is a system that uses at least two distinct visual representations (graphical or textual) to support the investigation of the same data. Multiple views are also used to visualize multiple aspects of data that require different visualization approaches. For example, a graph is constrained to its topological relations and cannot be brought into temporal order easily. However, one could map the time of each vertex to position on a linear scale, resulting in a flat sequence of vertices. It sometimes is necessary to create different representations of the same data using multiple coordinated views. Baldonado *et al.* formalized eight rules for using multiple views in visualization. The rule of self-evidence suggests to use perceptual cues to amplify relationships among multiple views. This means that the selection of data points in one view is synchronized with others simultaneously [KHG03].

Brushing Brushing is an interaction technique which is done by selecting one or multiple data items. This selection can either be made by drawing a rectangle on top of the visualization or by executing a query [BC87]. Martin *et al.* describe n-dimensional brushing in data instead of drawing space. With their data-driven approach they can consider multiple dimensions, combine brushes with logical operators, and apply non-discrete brush boundary functions [MW95].

Linking Linking multiple views means to synchronize representation, interaction, or data manipulation among all views. For example, interactions such as selection (e.g., via brushing), rotation, zoom, or filter affect all views simultaneously [KHG03].

In general, the brushed points are highlighted through identical visual channels such as color or size, so they are consistent in every view [KHG03]. Figure 3.5 shows an early example of this technique [BMMS91]. In combination this process is known as linking and brushing.



(a) (b)

Figure 3.5: *The visualization depicts expensive housing at mild climate zones. Brushing is done via rectangle selection in (a). A linked map where the same data items can be located in the State of California, USA is shown in (b). (Figure from [BMMS91].)*

3.2 Visual Attribute Encoding

In today's research, it is common that complex datasets contain multi-dimensional data of various data types. Especially in biomedical research, samples are acquired through different techniques and are processed in custom pipelines, resulting in heterogeneous datasets and attributes.

In the literature, **multi-dimensional data** is often used as a synonym for table data. Typically, multi-dimensional data is organized in columns and rows, just like a relational database table, storing the data records, relations, and descriptive information such as meta-data [CMS99].

3.2.1 Multi-Attribute Encoding

Oliveira and Levkowitz surveyed visual mapping techniques for multi-dimensional data. An initial step that all visualizations share is the usage of data aggregation techniques to

reduce the complexity of raw data. Methods like *dimension reduction*, *subsetting*, *segmentation*, or *clustering* are the most prominent [OL03] among them. In our work, we consider a set of abstraction techniques shown in Section 3.3.2 and Section 3.3.3 to fold the provenance graph into multiple layers of abstractions that we call hierarchy levels. Besides the topological information that every level holds, analyses were created at a certain time and correspond to a single workflow template. Datasets are uploaded in the ISA-tab format that contains a series of generic attributes such as author, sample name, or cancer type. As of Task II, users want to observe these attributes from analyses, workflows, and files while they browse the provenance graph.

Oliveira and Levkowitz reviewed visual data mapping techniques for multiple attributes and assigned them to four main categories [OL03]:

Icon-based This approach maps multiple attributes to an icon (also referred to as symbol or **glyph**), whose visual representation varies depending on data values. In 1973, Chernoff did some precursory work on displaying multiple attributes with glyphs. He proposed the well known *Chernoff faces*, that map various attributes to parts of the human face (e.g., eyes, nose, mouth, etc.). The main reason was that different faces and their changes are easily recognized and memorized. The individual parts of a face are perceived with different weight, allowing to order attributes by their importance [Che73]. The disadvantage of glyphs however are that they only work for a small attribute count, they may be perceived subjectively, and training may be required as their graphical composition might not always be self-explanatory.

Geometric projections This visualization technique projects attribute values of multi-dimensional data into geometric space. *Parallel coordinates* [ID90] are a method that draw n parallel axis for n attributes of a data item, where each attribute value is mapped to the y position on the corresponding axis. Connecting these coordinates creates a polygonal line that represents all dimensions of a single data item. Probably the most widely recognized technique are *Scatterplots* [Cha77], which project pairs of attributes onto x and y position in a coordinate system. For dimensions greater than two, multiple scatterplots are necessary. Both visualization approaches are very effective in detecting outliers and correlations among multiple attributes. However, parallel coordinates are limited to a few thousand data items and matrices can get very sparse when outliers are present.

Pixel-based The visualization space is partitioned into multiple subwindows, one for each attribute (dimension). The attribute is drawn as colored pixel, using a scale that maps the attribute value to color. All data items can be arranged in an appropriate order, though their position within all subwindows should be identical. An advantage of pixel-based visualizations is that a fixed pixel size cannot cause visual clutter [Kei00].

Hierarchical Multi-dimensional data, that does not necessarily contain a hierarchy, is (recursively) divided into subspaces by hierarchical transformation (e.g., dimension stacking as of [LWW90]). Each attribute is treated differently and separately, resulting in multiple views.

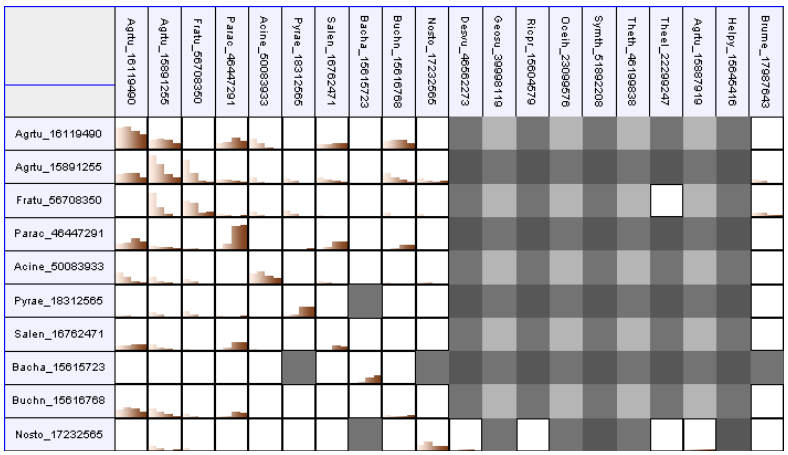
3.2.2 Visual Variables

Originally, glyphs were invented for cartographic purposes. Bertin followed the idea of arranging visual symbols in particular ways to convey information [Ber83]. He introduced the term *visual variables*, which sometimes are also called visual *attributes* or *channels*. They describe the methods for modifying *marks*, the primitives of graphical objects, by *position*, *size*, *shape*, *value*, *color*, *orientation*, and *texture*. Later, Carpendale added *motion* for the use of animations in computer graphics visualizations [Car03]. Each variable has its characteristics, hence it is important to choose the appropriate variables when mapping data to marks. The characteristics are (1) selective: the distinction of marks, (2) associative: similar marks are perceived as group, (3) quantitative: the numerical difference between marks, (4) order: the representing mark conveys order, and (5) length: the range of different representation. Note, the color spectrum is theoretically infinite, but only a limited amount, e.g, seven colors, are suggested in practice. [Ber83, Car03]

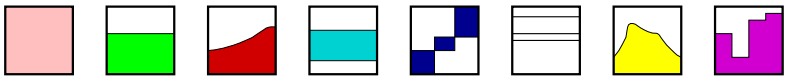
The effectiveness of how multiple attributes convey information heavily relies on the right choice, use, and combination of visual variables as different encoding may influence human perception. Before going into the details of graph visualization in the next section, we will have a look at two different approaches on visual glyph design for graph-based data.

ZAME [EDG+08]

Zoomable Adjacency Matrix Explorer (ZAME) is a visual exploration tool for very large graphs. In order to deal with large datasets, millions of data items and the relations among them are summarized into visual aggregates in an adjacency matrix. Matrix columns and rows are reordered by the means of minimal distance measures to related data items or aggregates. The matrix tile glyphs feature multiple levels of detail, with the level zero showing the most detail in an adjacency matrix. A set of tile glyphs is shown in Figure 3.6b. Various metrics such as minimum, average, or maximum amount of edges (the relations) are precomputed. These metrics are then encoded by the means of a set of visual variables (position, size, shape, color) and different visualization techniques (e.g., histogram in Figure 3.6a) [EDG+08].



(a)



(b)

Figure 3.6: *ZAME glyph design: (a) A subset of a matrix visualization using histogram glyphs. (b) Aggregated edges in a graph are represented in one out of eight differently textured glyphs. (Figure from [EDG+08].)*

Taxonomy-Based Glyph Design [MRSS+12]

Maguire *et al.* illustrate how novel visual encodings can deal with multiple attributes in workflow visualizations. With their approach, attributes are encoded into meaningful glyphs, accelerating visual search tasks. Still, the creation of glyphs is handled spontaneously during development or developers not have sufficient background knowledge in the applied domain so that glyphs turn out very subjective, are too abstract, or lack domain-specific metaphors.

Maguire *et al.* addressed this issues for biomedical workflows. First, they preprocess data to build a taxonomy and classify attributes such as in- and outputs, material, or data and many more for higher levels of detail. The resulting taxonomy reflects a hierarchical abstraction of attributes where the level of detail decreases in depth. Second, they review and order visual variables based on their perceptual strength. Third, domain-specific glyphs are created through abstraction and metaphors. In a final step, each attribute class is mapped to the corresponding glyph representation. Figure 3.7 shows glyph components for the attribute class at the highest level of detail. Their results show aesthetically pleasing glyphs that even can convey domain-specific information. However, this enriched information comes from a precomputed taxonomy that requires a time-consuming process of evaluation and a possibly large user study within the biomedical domain first.

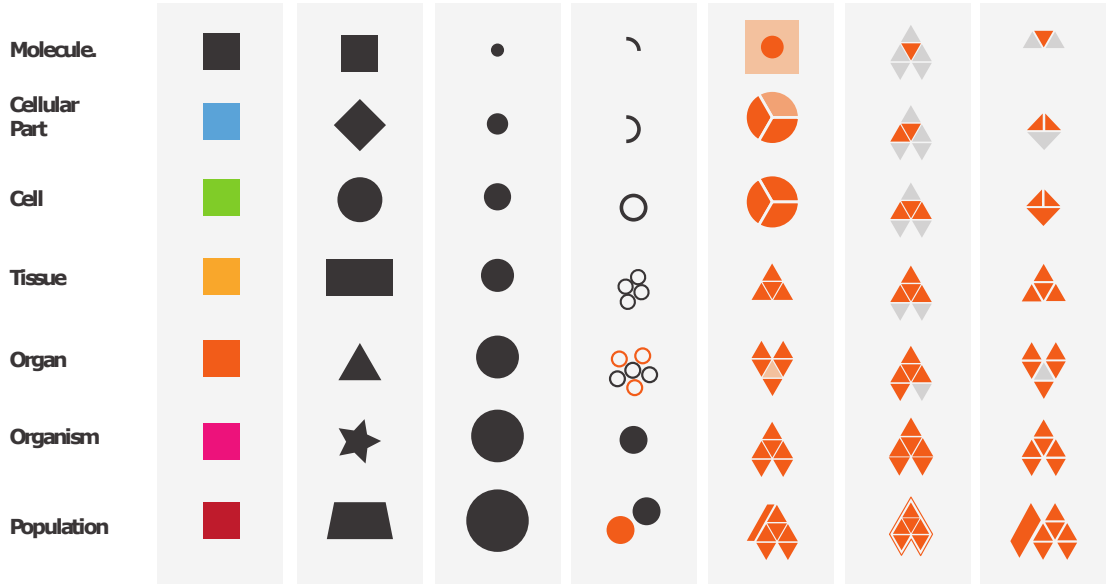


Figure 3.7: *This palette of icons and glyphs shows the taxonomy class with the highest level of detail (a leaf node in the taxonomy tree). Each row corresponds to a different attribute within the class. The columns represent seven design choices, where exactly one is picked for the visual mapping. Visual variables from left to right are: color, shape, size, and value. Domain-specific metaphors that vary in orientation, position, and texture add more choices (column five to seven). (Figure from [MRSS+12].)*

3.3 Graph Visualization

Whenever data items contain inherent relationships, a graph is used to depict the underlying structure. Graphs are used in a wide variety of applications such as circuit diagrams, social networks, or biological pathways. Within the said fields, graphs typically are very large, so humans rely on proper models or visualizations to carry out exploration and navigation tasks. In this section, the fundamentals of graph visualization are explained. Starting with a formal terminology below, representation approaches and graph drawing are the topics thereafter. The main part of this section addresses related work on novel graph aggregation techniques.

An *undirected graph* $G = (V, E)$ comprises a set $V = \{v_1, v_2, ...\}$ of *vertices* or *nodes* and a set $E = \{e_1, e_2, ...\}$ of unordered pairs (v_i, v_j) of distinct elements of G, the *edges* or *links*. If E is ordered, the graph is also known as *directed*. For every vertex v_i, if there is no sequence of ordered pairs e_k to reach v_i again, the graph has no cycles and therefore is called *Directed Acyclic Graph (DAG)*. In this thesis, whenever the term graph is used, we simultaneously refer to a DAG, as the provenance graph in Refinery conforms to this rules too. A formal encyclopedia for over more than two hundred classes of graphs is given in *Graph Classes: A Survey* [BLS99]. For further readings in graph theory, we suggest one of

the comprehensive books such as the *Graph Theory with Applications to Engineering and Computer Science* [Deo74].

Graph-based Tasks

While both nodes and links separately contain attribute-like information, the relations among them have significant value when formalizing tasks that are carried out on graph-based datasets. Users not only want to know the length, value or name of a data item, they are interested in hierarchical (e.g., number of child nodes) and topological (e.g., shortest path from node to node) information [Shn96]. A detailed list of typical user tasks in graphs can be found in [SA06]. Lee *et al.* defined a list of tasks for graph visualization, categorized into four groups [LPP+06]. We assigned our tasks (presented in Chapter 2) to this four high-level categories: *topology-based* (Task V and Task VI), *attribute-based* (Task II, Task III, and Task V), *browsing* (Task IV), and *overview* (Task I and Task V) [LPP+06]. Our elicited tasks are well balanced within the graph-based task taxonomy by Lee *et al.*. We also suggest that all tasks are equally important and they rather reflect a priority list similar to the visual information seeking mantra.

In the next section, we will discuss the advantages and disadvantages of various graph visualization approaches.

3.3.1 Representation Taxonomy

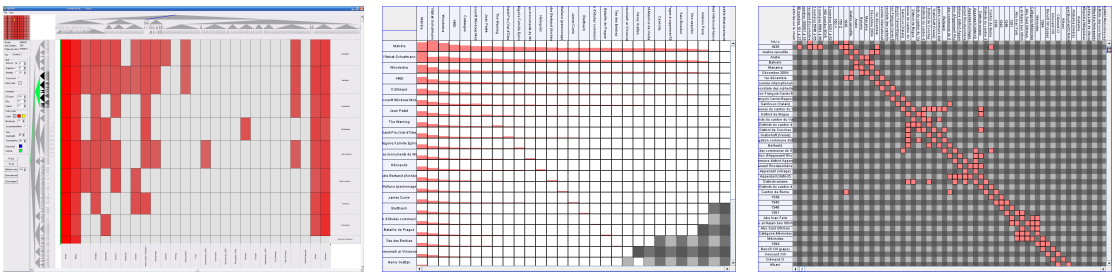
As of Schulz and Schumann, graphs are mostly visualized as one out of the three types: *matrix*, *explicit* (also known as node-link diagrams), and *implicit* representations. Other specific approaches try to combine these techniques and go by term of *hybrid representations* [SS06, Sch10]. In order to differentiate node-link diagrams from other representations, we rather use the formal terms, vertex and edge, instead of node and link.

Matrix Representations

When a graph is be written down in tabular form, we refer to an *adjacency matrix*. In this square matrix, the vertices V are associated to rows and columns, resulting in VxV matrix tiles. These inner tiles represent all possible edges E in the underlying graph, and if an edge between two vertices exist, the corresponding tile is marked. In the context of our work, a DAG has two important attributes. First, the graph is *directed*, so an edge is read in the direction from row to column or vice versa. Second, the graph is *acyclic*, meaning that cycles are prohibited. Furthermore, diagonal tiles are formally speaking *zero*, as not even self-loops are allowed for this type of graphs. From a visualization perspective, this representation follows an edge-centric approach, devoting most of its drawing space to edges. Edge tiles are usually encoded based on a color scale mapped to their weights [Sch10]. More complex approaches such as mentioned in Section 3.2.2 even use comprehensive

glyphs to encode multiple attributes (see Figure 3.8b and Figure 3.8c) [EDG+08]. A matrix representation greatly depends on its row and column order, and with appropriate adjustment, it simplifies the visualization overall and reveals clusters. Typical ordering algorithms include simple sorting, attribute similarities, or distance measures such as Euclidean distance [Sch10]. For large and sparse graphs, screen space becomes an issue. In this case, vertices and edges that share similarities can be hierarchically aggregated into subgraphs (see Section 3.3.2) and visualized as abstract tiles. As shown in Figure 3.8a, for example, alpha values represent the edge density in a subgraph [AvH04].

Due to the emphasis on edges, matrix representations are well suited for attribute-based tasks performed on edges. Counting them, or finding adjacent neighbor vertices in dense clusters can be achieved very fast. However, path traversal prove to be more difficult. When following a short path, the user usually loses context after a few steps already. Node-link diagrams, which we will discuss next, excel in topology-based tasks [Sch10].



(a) (b) (c)

Figure 3.8: *Matrix representation examples: (a) a screenshot showing the Matrix Zoom visualization on colon cancer cases in the counties of the State of Connecticut. The horizontal axis represents human racials, and the vertical axis counties. (b) shows an overview with aggregated tiles in contrast to a detailed subset in (c). (Figures from [AvH04, EDG+08].)*

Explicit Representations

The fabrication of electronic devices is a practical example where board space is explicitly limited to optimize production costs. A space-efficient wiring diagram of electrical components and connections is required just like in today's visualization approaches. The main idea of this representation is to map vertices to coordinates, and edges to arcs or curves [Sch10]. The graphical representation of vertices ranges from textual labels to composite glyphs or a combination of both. Edges are drawn as simple line segments, and in case the graph is a DAG, the orientation can also be encoded by the diagram's overall orientation (e.g., top to bottom, left to right). However, a common way to encode the directionality of edges is to add arrowheads to one endpoint of an edge. Edge weights are sometimes added as textual labels or are implicitly encoded in the width of an edge.

For medium-sized graphs, node-link diagrams are certainly more intuitive than matrix representations. They display vertices and edges in a balanced way, so that even users without visualization expertise are able to comprehend the visualization. In Section 1.1.5, a classic node-link diagram was presented in Figure 1.2. The dataflow goes from left to right. Vertices are encoded as node glyphs, having in- and output files attached to the left and the right side. Because vertices and edges are tightly coupled, node-link diagrams are well suited for topology-based as well as path-finding user tasks [Sch10] such as Task VI.

Figure 3.9 shows a more complex visualization example. Vertices and edges are overlapping as the graph is more dense, and on first sight, vertex placement seems arbitrary. It becomes apparent, that node-link diagrams do not scale to a large number of vertices and edges. As in any visualization, hierarchical aggregation and appropriate visual encoding are good countermeasures against visual clutter and other resulting issues. Yet the hardest challenge lies in the computation of the node-link layout, which will be addressed in the context of graph drawing below [Sch10].

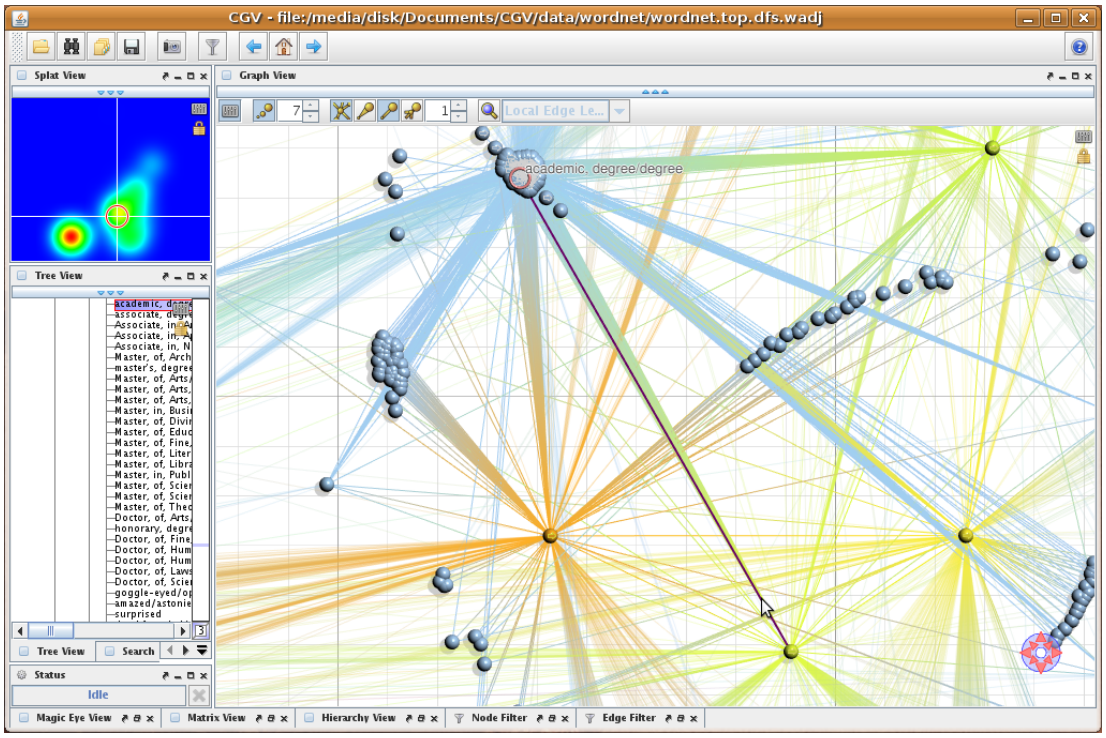


Figure 3.9: *The Coordinated Graph Visualization (CGV) is a novel and general purpose graph visualization implemented in the node-link diagram style. It features a wide variety of state-of-the-art interaction techniques such as multiple views, filtering, lenses, and many more. (Figure from [TAS09].)*

Graph Drawing

This research field builds the foundation for every graph visualization. For node-link representations, layout computation is a key aspect as it helps to improve readability and furthermore enhances the user experience. An effective layout depends on the properties known as graph drawing aesthetics. They are concerned with the following problems: *edge crossing reduction*, *edge bend minimization*, *uniform edge length*, *graph symmetry*, and many more. A list of them can be found in the classic book on graph drawing by Battista *et al.* [BETT98]. To compute a layout that conforms to most of them is considered an NP-hard problem [DPS02]. Precise methods, such as the exact determination of edge crossings, are not always feasible because they often result in polynomial runtime complexity. Hence, heuristic approaches which follow a practicable approach are far more realistic [Sch10]. On top of that, postprocessing techniques such as *edge bundling* [Hol06, HVW09] can further reduce visual complexity and clutter.

When designing graph visualizations, it is advantageous to review the tasks the users seek to accomplish. How well a task performs is often closely related to the aesthetic appearance of a visualization. Hence, tasks and requirements might help to narrow down the choices for the right layout. There exist a wide variety of layout strategies [HBO10]. As of Schulz, node-link diagrams can be classified into three layout types by their degree of freedom as follows [SS06, Sch10]:

Free Vertex placement does not conform to any rules and coordinates are assigned arbitrarily. However, it is used to point out interesting patterns such as clusters or sparse regions in the graph structure itself. They are the first layout choice for displaying social networks. Common implementations are *force-directed* or *spring embedder algorithm* approaches. They compute attractive forces of adjacent and repulsive forces on all vertices [Ead84]. Fruchterman and Reingold later enhanced this method. They aimed for uniformly distributed vertices and edge lengths [FR91]. The downside of the original force models is that a uniform distribution distorts natural clusters in the graph. In [vHvW04], a modified force model better preserves and amplifies these natural cluster structures in the graph.

Fixed Vertex placement is associated with predetermined coordinates in the drawing space (e.g., geographical map data). Fixed layouts only permit edge adjustment, known by the term *edge-routing*. With fixed vertices, a heavy focus is laid on how edges are routed around other unrelated vertices to avoid overlapping [DK08].

Styled Vertex placement is less restricted and neither is it free. When vertices are constrained to a scheme, e.g. they are placed onto geometrical shapes such as predefined circles or grids, an average layout size can be computed reasonably well. In addition, certain interaction techniques may work best with a distinct layout style. In contrast to free layouts,

a style also preserves recurring patterns that are laid out deterministically. Among the most well-established types are circular and grid-based techniques. With a *circular* layout style [GK06], vertices are positioned on a perimeter. Adjacent edges are bundled and either routed in- or outside of the perimeter to reduce density and minimize crossings. *Arc diagrams* [Wat02], formerly known as *linear* layouts [Cim06], are used to spatially separate vertices from edges. *Grid-based* layouts place vertices onto a *rectilinear grid* with the edges routed orthogonally. In [RHR+10] they use this particular method for metabolic pathway networks. The pioneering work on styled layouts was done by Sugiyama *et al.* in [STT81] when they introduced the technique of *layered graph drawing* for DAGs. Their method follows a three-step approach: (1) *cycle removal and layer assignment*, (2) *crossing reduction*, and (3) *horizontal coordinate assignment*. These steps will be explained in more detail below.

Layered Graph Drawing [STT81] The general idea is to arrange vertices into a sequence of horizontal layers (also known as ranks or sometimes referred to as columns). In the next step, edge crossings are minimized by pair-wise layer iterations. The remaining step aims at symmetry aesthetics by horizontally balancing vertex placement along each layer. The method focuses on aesthetics of a maximum layer width, evenly distributed vertices over the drawing space, straight edges, and edge bend minimization.

1. *Cycle Removal and Layer Assignment:* First, pair-wise relations are extracted to form a proper data structure such as a matrix. In case the graph contains cycles, a minimum set of edges is removed by fast heuristics or exact methods, resulting in a single or multi-level acyclic digraph, a DAG. Second, the graph is divided into a sequence of horizontal layers, so that a vertex's predecessors are exactly one layer above and the successors one below. For edges that span over 2 or more layers, vertices are added to empty layers and the edge is split into multiple edges with span 1. All of the resulting layers now only contains edges with span 1. *Coffman-Graham* layering [CG72] is now applied to minimize the graph width.
2. *Crossing Reduction:* The number of crossings between two layers is computed through the combinatorial order from both. In contrast to exact methods, heuristics such as *median* or *barycenter* methods are fast and deliver satisfying results. The barycenter method is among the most common technique that is typically used. The properly layered graph is iterated in pair-wise layers and based on the degree of neighbor vertices, the determined barycenter coordinate is used to reorder the current layer (see Figure 3.10).
3. *Horizontal Coordinate Assignment:* This step focuses on symmetry, maximizing edge lengths, and minimizing edge bends that occur at the dummy vertices. Horizontally shifting vertices reduces bends, however it simultaneously may effect the maximum graph width (see Figure 3.10). Brandes and Köpf enhanced this step by recomputing the layout four times with varied alignment initializations: top-left, top-right, bottom-left, bottom-right. The vertex placement is then determined by the mean

of all four layouts [BK02]. In a last step, the dummy vertices are removed and the original edges are restored.

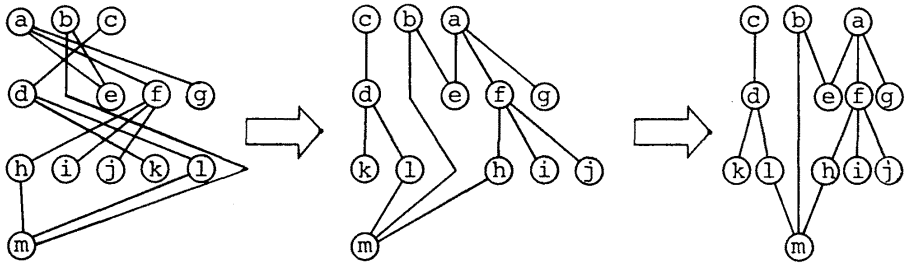


Figure 3.10: *Example drawing for step 2 (left arrow) and step 3 (right arrow) of the Sugiyama method. (Figure from [STT81].)*

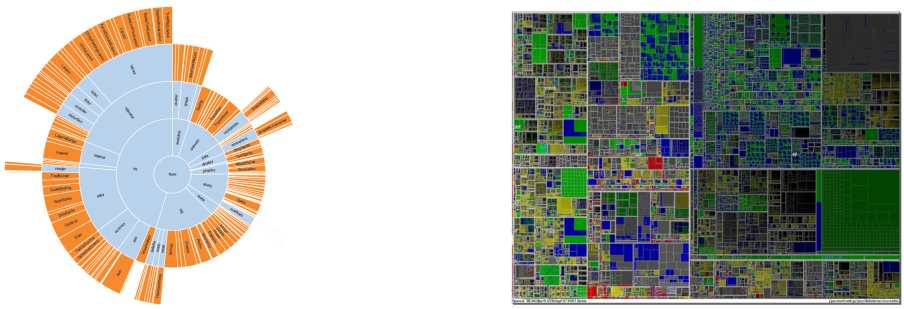
The open source graph visualization software *Graphviz*[3] is a well-known application that features a layered graph drawing algorithm known as *dot* [GKNV93]. Gansner *et al.* developed a four-pass algorithm similar to that of Sugiyama *et al.*. For the first step, they make use of the *network simplex* algorithm to assign ranks.

Implicit Representations

Implicit visualizations are characterized by their underlying hierarchical data and the absence of edges. Relations are encoded by the relative position of the vertices through *inclusion*, *adjacency*, and *overlap*. Two implicit approaches we discuss next are depicted in Figure 3.11. Due to their hierarchical top-down approach, implicit representations are often called *space-filling* techniques [Sch10]. The most commonly used is known as the *treemap* layout and has been invented by Johnson and Shneiderman. The main idea is to recursively divide the drawing space into nested rectangles. The rectangle's depth and spatial proportions directly encode its hierarchy level and in turn makes parent-child relations easily recognizable. Additionally, rectangles are often color-coded to display vertex attributes. In contrast to classic node-link diagrams, this approach makes use of the whole drawing space. Layout algorithms deal with user-controlled rectangle size and aspect ratios, thus larger hierarchies can be displayed [JS91]. Modern implementations have shown that this technique scales well for more than a million vertices [FP02]. Another implicit representation, the *sunburst* layout, follows a radial growing approach where deeper hierarchy levels are drawn further from the center [SZ00].

Implicit representations make efficient use of the drawing space and further enhance hierarchical perception. Though they are constraint to tree-like data only. Some work has been done to overcome this limitation by superimposing (overlaying) edges into treemaps. In order to reduce visual clutter, the edges can be bundled hierarchically [Hol06]. This

[3] `http://www.graphviz.org/`



(a) (b)

Figure 3.11: *Implicit representation examples: (a) sunburst and (b) treemap. (Figures from [HBO10, FP02].)*

approach also falls into the hybrid representations category as it combines features from both treemaps and node-link diagram.

Discussion

Each representation comes with a series of distinct advantages or disadvantages.

Ghoniem *et al.* evaluated the readability of matrix and node-link representations according to the answer time that subjects needed to accomplish each of the seven graph-related tasks [GFC04]. They suggest that matrix representations outperform node-link diagrams at a size of twenty vertices. From a time complexity perspective, matrix representations have a few advantages over explicit and implicit representations. No complex layout computation nor edge crossing reduction is required [Sch10]. They scale well to millions of vertices while occlusion or clutter is of no concern [EDG+08]. Columns and rows can be reordered appropriately to show clusters or adjacent vertices quickly. Another advantage which is often forgotten is that they can highlight absent connections. Especially for dense graphs, where node-link diagrams struggle due to edge crossings, a missing edge cannot be recognized easily [vHSD09]. Nevertheless, the adjacency matrix takes up quadratic drawing space, its tiles are fixed, and the absent of drawn edges make it very hard to follow paths. Although vertices can be reordered and edge weight is color-coded, path following remains as one of the hardest task to accomplish in matrix visualizations.

Node-link diagrams are effective for moderate sized and sparse graphs. They provide a balanced overview for both vertices and edges. They are well suited for topology-based tasks, especially for path-finding tasks [Sch10]. Directed graphs such as scientific workflows, automatically imply a general flowchart that is far more intuitive and recognizable as matrices ore treemaps possibly could achieve. However, for larger graphs the performance of node-link diagrams deteriorates quickly. Although many heuristic approaches considerably reduce the computational effort, they are still slow in comparison to implicit or matrix representations. Computing optimal node-link layouts is considered an NP-hard problem.

Within dense regions that contain a high amount of edges (e.g., social network data), they tend to suffer from occlusion and clutter, resulting in chaotic *hairballs* [Sch10].

Implicit techniques are preferred when dealing with hierarchical data. Due to space-efficient mapping, millions of vertices pose no problem. Though just like matrix representations, edges are omitted which makes it hard to follow paths [Sch10]. A user study about usability characteristics for explicit and implicit representations can be found in [BN01]. The results show that implicit visualizations are well suited for cases where size encodes vertex attributes. However, they perform poorly on edge attribute-based tasks.

Hybrid Representations

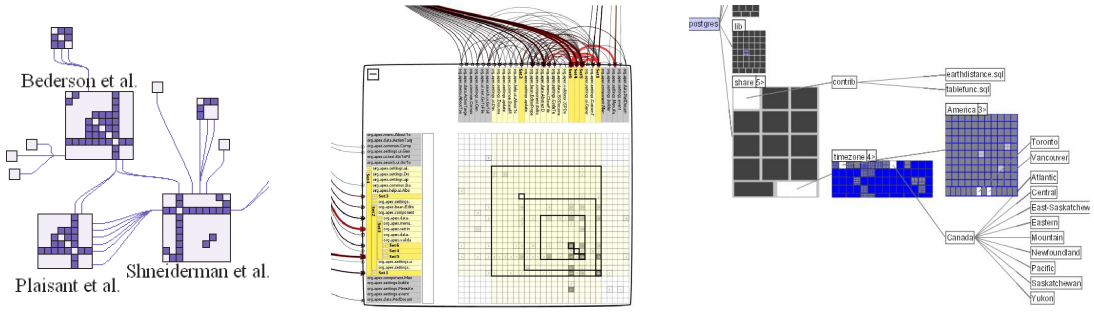
Recently, more and more *hybrid* approaches mix representations to compensate the limitations and emphasize the strength of individual layouts. When datasets feature a mix of data types, it is not ideal to stick to a single layout. In the following, we discuss the three hybrid approaches shown in Figure 3.12.

NodeTrix [HFM07] *NodeTrix* computes a node-link layout for its overall structure and uses matrix representations for dense regions. This hybrid approach is motivated by identifying communities in a social network. The relations among the people in a group are less important than the presence of a community itself. A community counts the members that are simultaneously related to others, resulting in many edges and potential crossings. Henry *et al.* propose three options to handle the inter-connections between matrices and nodes: (1) display aggregated links, (2) display every link, and (3) display attribute-based edge-bundles. The third option significantly improves readability and points out node attributes at the same time.

TreeMatrix [RMF12] Analogous to NodeTrix, *TreeMatrix* renders its global structure based on a node-link approach. Due to the hierarchical nature of their dataset, which represents the tree structure of source code files, the upper levels of the tree is depicted in nested bounding boxes. These compound subgraphs are subdivided into adjacency matrices that again show hierarchies, but this time within the matrix header. Additionally, the intra-connections inside matrices are displayed with arc diagrams attached outside to the matrix header labels. However, they overlap with inter-node connections which affects readability. Overall, this hybrid approach features multiple hierarchy levels and keeps the overview in context while enabling details on demand in a single view.

Elastic Hierarchies [ZMC05] *Elastic hierarchies* are the means by which node-link diagrams and treemaps are dynamically combined in interleaving fashion. Zhao *et al.* implemented a side-by-side overview+detail visualization where the user can manually switch from treemap to node-link representation and vice versa. This technique can be applied

on the overview graph as well to particular subgraphs of interest. For example, drilling-down multiple hierarchy levels, uninterested context can be collapsed into a space-efficient treemap while subgraph of particular interest are shown as node-link diagrams.



(a) (b) (c)

Figure 3.12: *Hybrid representation examples: (a) NodeTrix, (b) TreeMatrix, (c) Elastic Hierarchies. (Figure from [HFM07, RMF12, ZMC05].)*

3.3.2 Hierarchical Aggregation

Driven by technological advances, datasets become larger and more complex continuously. Especially the visual space is limited, and if it were not, humans nonetheless are restricted to their field of view as well as perception capabilities. In order to provide interactive visualizations that are capable of handling large datasets within a limited visual space, either the data or the visual complexity must be reduced [EF10]. In Section 3.2, visual mapping techniques that deal with multiple attribute on a per data item basis were already described [OL03].

This section however introduces hierarchical aggregation as a data abstraction technique to reduce the size of the raw dataset by grouping related data into so-called *aggregates*. This process can be applied recursively, resulting into multi-scale structures built upon a tree of aggregate items with parent-child relations (see Figure 3.13) [EF10]. In Refinery, the provenance graph is divided into predefined hierarchy levels (analysis, subanalysis and workflow). If this would not be the case, there exist well-established clustering algorithms that are suited for adding such hierarchical abstraction levels (e.g., *k-means* clustering).

Each aggregate from the data space is translated to its corresponding *visual aggregate* in the visual space. These aggregates hold metrics about their aggregated child nodes (or edges) such as average, minima, or maxima. The hierarchy levels are rendered in breadth-first fashion by traversing the aggregate tree in top to bottom order. Higher-level aggregates are occluded by lower-level aggregates which show a more detailed representation. Figure 3.13 shows four different types of rendering traversals. Given the appropriate use case, it could be beneficial to perform an unbalanced traversal, e.g., expanding detail information in one branch while collapsing them in the other [EF10].

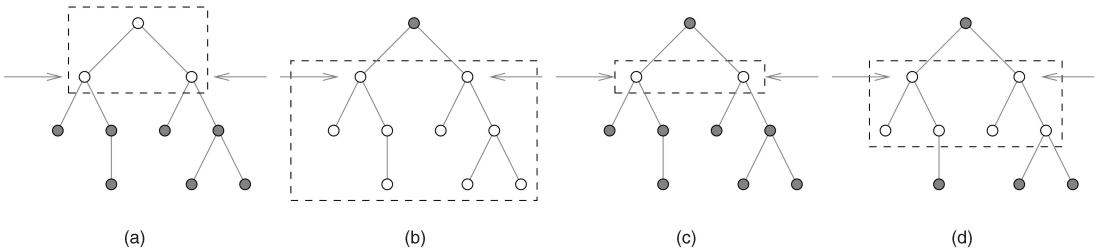


Figure 3.13: *Hierarchy levels are rendered according to the shown arrows: (a) abstract nodes above, (b) detailed nodes below, (c) single hierarchy level, and (d) a range of hierarchy levels. (Figure from [EF10].)*

A significant challenge when rendering aggregated data lies in the visual continuity and transition of hierarchy levels. Interactive navigation techniques such as iteratively collapse and expand, known as drill-down (Task IV) and roll-up (Task I), within the hierarchy levels must be provided to maintain the mental map of the overview data [EDG+08].

Besides Elmqvist and Fekete's overview on hierarchical aggregation [EF10], other work has been done. In [EDG+08], more than one million data items are aggregated into a matrix-based overview+detail visualization. Noel and Jajodia apply hierarchical aggregation based on protection domain categorization in network attack graphs. Their visualization builds on a classic node-link diagram and supports novel interaction techniques [NJ04]. Another node-link diagram that uses clustering is the *ASK-GraphView* which is capable of dealing with over more than 16 million edges. In contrast to the former examples, [AvHK06] generate an arbitrary hierarchy through recursive clustering.

3.3.3 Network Motif Discovery

Every network has its unique structural characteristics, both globally and locally. In natural sciences (e.g., biochemistry, neurobiology, ecology, and engineering) there is a particular interest on how networks are formed over time [MSOI+02]. In the context of this thesis, workflows often conform to a distinct sequence of preprocessing or analysis steps that are more frequent than others. We argue that scientific workflows, as well as the evolving provenance graph on top of it, evolve based on those recurring substructures. From a visualization perspective, motifs can be exploited to further reduce visual complexity by aggregating these recurrent patterns.

Milo *et al.* implemented a network motif detection algorithm to reveal *motifs* to classify networks based on motif occurrences. They defined the term *motif* as a recurring and significant pattern of relations among a set of vertices. In other words, motifs describe the basic building blocks of a network that recur more frequently in real networks than in randomly generated networks [MSOI+02]. Yet, motif discovery does not particular well in terms of runtime complexity. As of Aittokallio and Schwikowski [AS06], the basic steps of motif discovery are (1) determine motif frequency, (2) assign motifs to groups and remove

isomorphs, and (3) order motif groups by their appearance frequency. Exact computations in step 1 can be very time-consuming. In practice, no motif discovery algorithm considers motifs of a size greater than ten. A detailed survey on graph-driven methods that incorporate motif search or network clustering was conducted in [AS06].

Visual Compression with Macro Motifs [MRSS+13]

Recent work on motif discovery for the use in scientific workflow visualization has been done by Maguire *eta al.*. Their motif discovery is based on the popular *pattern-growth* algorithm. Applied in a state transition manner, a motif can start at any vertex in the graph. While a set of transition rules apply (e.g., branching), a motif can only grow forward and in breadth-first fashion until it is terminated by a rule again. In contrast to generic methods, their detection distincts between vertex and edge types (differently colored vertices in Figure 3.14c). This allows them to include semantic context that restricts motif frequency to a manageable size, which in turn reduces processing time. In a final step, most frequent motifs are determined and substituted with an appropriate graphical representation, the *macro*. A macro glyph is designed automatically under the consideration of metrics about the underlying aggregated topology (e.g, breadth, depth, vertex type, structure).

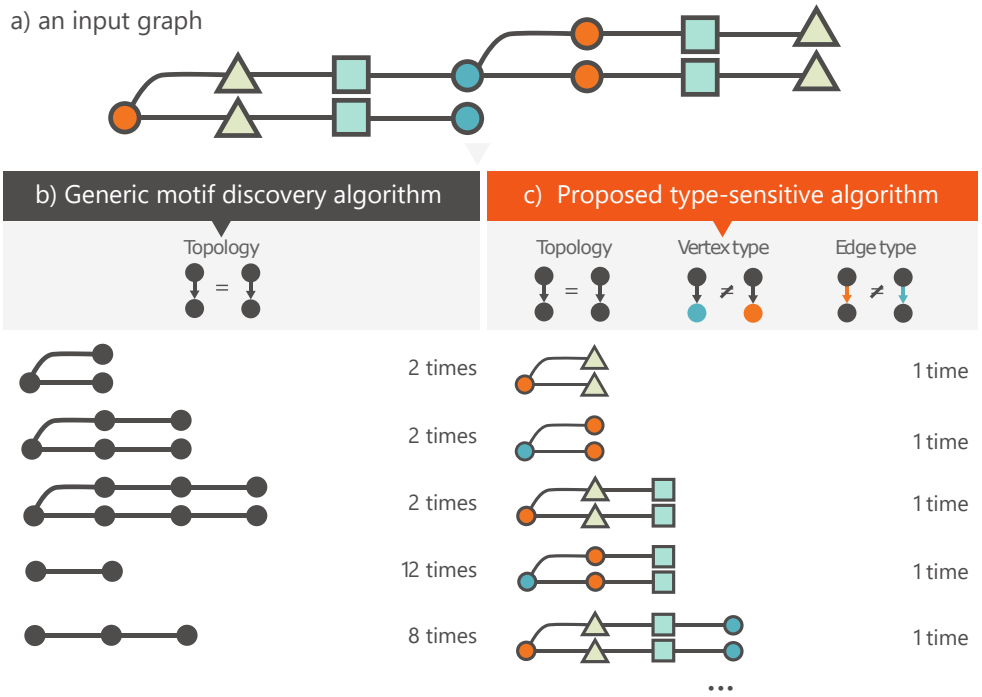


Figure 3.14: *Type-Sensitive Motif Discovery: (a) an example graph with varying vertex types, (b) generic motif discovery, and (c) motif discovery constrained to vertex and edge types. (Figure from [MRSS+13].)*

Yet, with their approach, identical motifs that are aligned in parallel are repeatedly

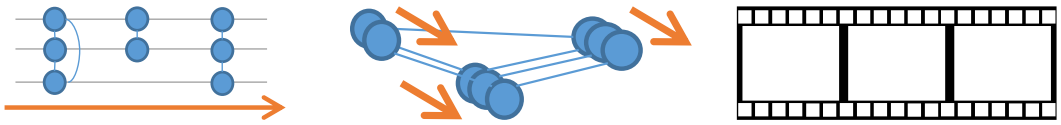
rendered again. It seems obvious to us that additional motif-based aggregation could be used, resulting in an even higher compression.

3.3.4 Dynamic and Temporal Graphs

In real world scenarios, graph-based data is rarely static and instead evolves over time. The dynamic aspect of a graph relates to the changes made to vertices, edges or their attributes. In Refinery, the provenance graph is subject to topology and attribute changes alike (Task V). Both topological and temporal information pose new challenges to the visualization of node-link diagrams. Static approaches consider each modification as an independent problem, compute a new layout, and redraw the graph for each step. This strategy comes with the following downsides. First, a slight change may not require a complete redraw instead minor adjustments to the layout could save some processing time. Second, the layout is recomputed which requires the user to re-familiarize with the updated graph. The latter suggests to prevent greater changes in dynamic graphs so that the user's *mental map* [ELMS91, MELS95] is maintained. General countermeasures that are widely applied in dynamic graph drawing are: (1) minimize layout change, and (2) support the user with smooth transitions [Bra01] to make inevitable changes more apparent. The use of animations is still considered controversial as well as their usage requires careful thought. On the one hand, they attract the user's attention to the changes, but on the other hand, they may distract and misguide. However, for dynamic content they have proven to be beneficial [GEY12]. An empirical user study on how transitions affect human perception is presented by Ghani *et al.*. Their results show that slow transition speed and a maximum distance between affected entities greatly helps to recognize and follow them [GEY12].

The topic of dynamic graph drawing has become very popular over the recent years. For example, three surveys have been published since 2014 [BBDW14, KKC14, HSS15]. All of them reviewed novel dynamic graph visualizations which deal with temporal data. A recurrent classification scheme is based on the spatial or temporal mapping technique that either map time to time (*animation*) or time to position (*juxtaposition* and *superimposition*). Figure 3.15 depicts these three approaches. While the animation approach is not capable to present changes over multiple time steps and does not scale to a large number of changes, juxtaposition uses multiple adjacent views at different time steps. Superimposition seems to be the most intuitive approach, because time is integrated seamlessly into the view. Time-varying nodes and links are stacked on top of each other with an optional vertical shift that represents time [BBDW14]. Over time, static parts of the graph will grow in stack height, while one-time changes are represented by single node placements distinguishably. However, this kind of technique may be inefficient and unintuitive for graphs that are subject to frequent changes.

In [BC03], Brandes and Corman present a three-dimensional approach that shows the incremental evolution of a graph over time. Each time step is mapped onto an individual and transparent layer on top of each other. The work by Erten *et al.* simultaneously



(a) (b) (c)

Figure 3.15: *Dynamic node-link diagram visualization techniques for time-varying graphs: (a) juxtaposition, (b) superimposition, and (c) animation. (Figure from [BBDW14].)*

draws two related graphs that share a subset of nodes into a single view. Additionally, an overlaid intersection shows the shared node- and edge-set [EKLN04]. Similar work on comparing two graphs has been done by Archambault. With hierarchical aggregation and path-preserving coarsening that group adjacent nodes and edges into a supergraph, a difference map for both underlying graphs that depict changes through spatial position and color is drawn [Arc09].

3.3.5 Degree-Of-Interest Function

The first visual representation that is presented to the user, showing the overall structure of the data (Task I), must provide an intuitive starting point for task-driven visual data exploration. State-of-the-art visualizations commonly address this with preprocessing and novel data abstraction techniques that reduce the raw data to an aggregated model before rendering. Usually only a high-level abstraction of the underlying content remains. Likely, users get confused or are clueless where to start and what to reason about. Users are in the need of a guided entry point that is based on their interest or task they pursue. In addition, they need interactive methods to deal with this multitude of hierarchy levels manually as well as automatically (Task IV). Furnas *et al.* proposed the idea of a geometrically distorting drawing space based on the level of interest the user currently has. We briefly introduced the fisheye view as a focus+context technique in Section 3.1 [Fur86]. In this section, however, we take a closer look at how user interest can be incorporated to help them dynamically control various levels of semantic aggregation.

The *degree-of-interest (DOI)* is a normalized number that represents the user's current interest about a visual element or region. This value is mapped to an interest scale, the DOI function, where a lower DOI means that the user is less interested in a particular element. The element with the highest DOI is also referred to as focal point, or foci for multiple points. Each element's value is computed from multiple attributes and metrics that reflect interest. The value decreases with distance to the closest focal point. The original DOI function by Furnas *et al.* comprises two components, the *a priori* importance and distance:

$$DOI(x|y) = API(x) - D(x, y)$$

For a focal point y, the DOI of any point x consists of the global a priori importance $API(x)$, and the distance $D(x, y)$. Hence, the level of interest increases with a priori importance and decreases with distance [Fur86].

The positive effects that a DOI function can have, are showcased by Heer and Card in their work on DOI trees [HC04]. They extended their original implementation with an optimized and scalable approach to DOI calculation and refined the tree layout to include multiple focal points. Results show that this interaction techniques scale well to hundreds of thousands of nodes, which is more than three times of the original size they could handle [HC04]. In [vHP09], Furnas' DOI function is adapted to work with graphs including specific metrics. They further augment the function with interest derived from user actions during the search process. Abello *et al.* [AHSS14] introduce a modular DOI specification for dynamic graphs that incorporates neighborhood structure, vertex/edge attributes, and temporal change metrics. The DOI function combines the a priori part, interest derived from user actions (e.g., node selection), and distance to multi-focal points. Figure 3.16 shows a sample graph visualization with the DOI function applied. Nodes of high interest (green colored circles) and their connecting edges (black lines) are emphasized according to their DOI.

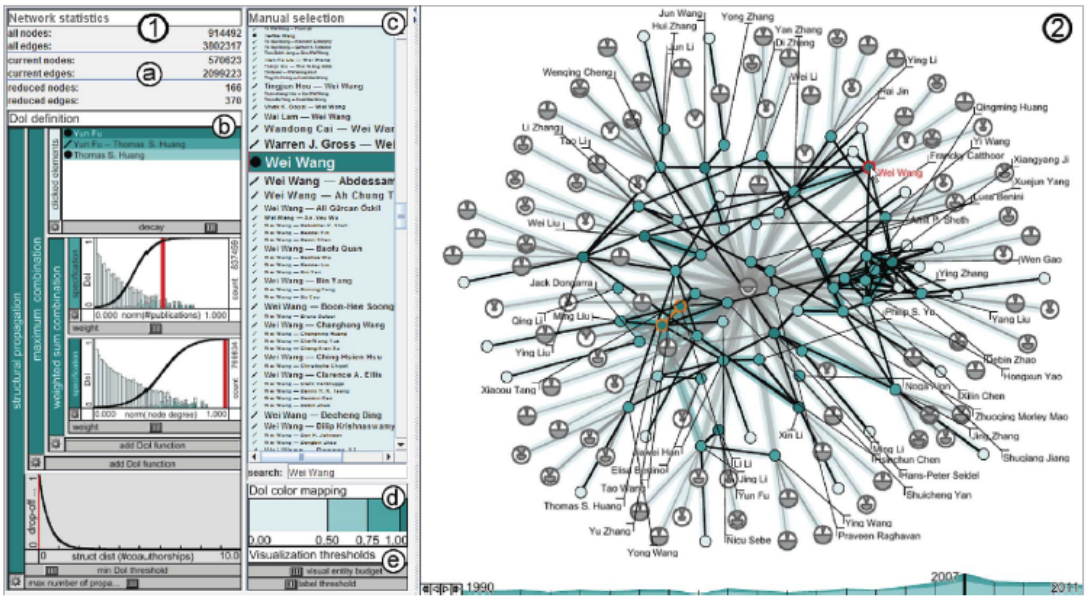


Figure 3.16: *The visualization shows two windows: (1) DOI view. (2) Network view. Single nodes (circles) and glyph that represent subgraphs with their node (top) and edge (bottom) count encoded. The DOI view is separated into five panels: (a) Graph and aggregation metrics. (b) DOI function variations. (c) Detailed list of graph elements. (d) DOI mapping color scale. (e) The labels of the DOI components shown in panel (b). (Figure from [AHSS14].)*

3.4 State-of-the-Art Provenance Visualizations

Data provenance, as described in Section 1.1.4 and [RESC15], captures the history of any changes through subsetting, merging, formatting, transformation, or execution of data. In the context of this work, we deal with workflow executions that cause changes at multiple levels (analysis, subanalysis, workflow) in a biomedical study. According to Ragan *et al.* [RESC15], there are four additional provenance types which are not addressed by this thesis. However, they are subject to future work and thus briefly mentioned as part of this introduction:

- *Provenance of Visualization* captures the history of views and visualization states by recording the images or settings needed to recreate any state of the visualization.
- *Provenance of Interaction* captures the history of user actions and commands such as button pushes, view manipulations, or queries.
- *Provenance of Insight* includes information obtained from the analysis process such as the history of hypothesis, insights, and other forms of analytic findings.
- *Provenance of Rationale* is concerned with the history of user intentions and reasoning behind insights and interactions, outlining the user's thought process and analytic strategy.

This work focuses on data provenance, that helps users to *recall* the analysis history, *replicate* intermediate and the final result, and *communicate* the development of the whole study over time.

Only a few frameworks deal with the collection as well as the visualization of data provenance. Specifically for biomedical research, novel and interesting approaches are rare, though since 2010 there seems to be a slight upwards trend. In the following, we will introduce and discuss the feature set of the most comprehensive systems and approaches the literature has to offer:

VisTrails [BCC+05]

The most prominent provenance visualization framework that has been around and actively developed since 2005 is VisTrails [BCC+05, FSC+06, CFS+06, SFC07, SKS+08, FS12][4]. It is an open-source scientific workflow and provenance management system that offers simulation, data analysis and visualization tools. Multiple views show information about workflow provenance, the derived data products, and their executions over time. Raw documents of study results can be added to workflows to support reproducible research. Redundant operations are cached to save significant processing power, for example when rerunning them for multi-view comparisons.

The main view provides a static node-link tree visualization of a workflow, with each

[4] http://www.vistrails.org/

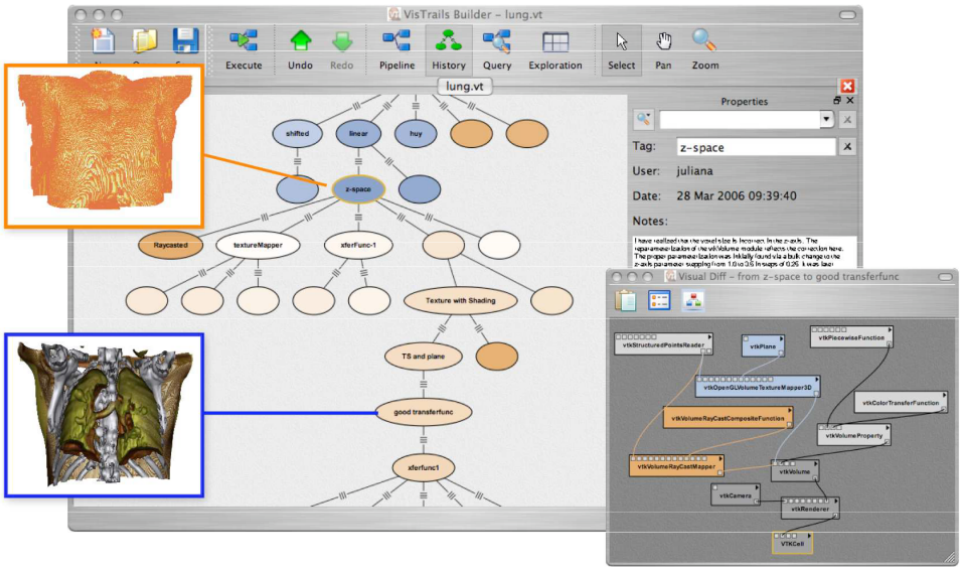


Figure 3.17: *A screenshot showing a workflow and multiple views in VisTrails. (Figure from [DCBE+07].)*

node representing a unique image manipulation step. However, the system lacks novel interaction techniques, node glyphs do not utilize the full bandwidth of visual variables, and aggregation techniques, which could make the visualization more scalable, are missing.

Kepler [ABJ+04]

Kepler [ABJ+04, ABJF06, LAB+06][5] is a domain-independent scientific workflow management system that provides accessible workflow creation tools, a workflow execution pipeline, and monitoring support. Kepler also supports access to distributed resources over the web through integrated web services. Its generic provenance collection module features a smart rerun manager that gives users the ability to execute parts of a workflow again. Redundant processes are detected and not executed again, instead they are replaced by intermediate results from the cache. Kepler's integrated visualization approach does not meet the expectations to a modern system. It lacks visual encoding and methods to handle large node-link diagrams. However, the focus is laid on technical features such as actionable provenance and the provision of an accessible API that allows third-party frameworks to integrate their comprehensive provenance collection module.

[5] `https://kepler-project.org/`

Taverna [WHF+13]

Taverna[6] is a general purpose, open source, and domain independent *Java*-based workflow management system for designing and executing scientific workflows. Its field of application is not limited to the biomedical research domain. For example, it is used in astronomy too. Complex workflows are created from resources of *Web Services* or locally installed tools. Taverna captures provenance information of evolving workflows via versioning and execution properties including in- and output files. A provenance visualization module is not included in Taverna, instead provenance is exported in the *Open Provenance Model (OPM)* format[7] and the *W3C PROV* model[8], both are accessible via the API, which third-party applications can make use of to visualize provenance.

GenePattern [RLG+06]

GenePattern[9] is a web-based analysis framework to manage genomic data. The extensible architecture allows collaborators to expand its core with data analysis and visualization modules. *GenePattern-Word RRS* is the result of early efforts to bring reproducible research to the masses, especially because it is integrated into *Microsoft Word*. Users are able to create analyses and keep track of their multi-step pipeline [Mes10]. During a user session it captures every process including, workflow parameter configuration, inputs, results, and tool versions. This comprehensive provenance information enables users to investigate and reproduce every step done in a study. However, information is displayed in tables and its dependency to Word makes it rather constricting.

Synapse [OEY+13]

The TCGA Pan-Cancer project required over 250 researchers around the world to work on the same set of data. To do collaborative science in distributed teams, a standardized environment is required. Hence, *Sage Bionetworks*, a nonprofit organization that promotes open science built Synapse[10] to manage data, allow groups to share projects, and track provenance to ensure reproducibility. The provenance visualization in Synapse is based on a graph that is rendered as a node-link diagram. Their simplistic approach clearly lacks novel visualization techniques. They do not use any form of semantic aggregation and the overly use of text labels combined with the lack of visual glyph encoding does not scale to graphs of moderate size and above.

[6] http://www.taverna.org.uk/
[7] http://openprovenance.org/
[8] http://www.w3.org/2011/prov
[9] http://www.genepattern.org/
[10] https://www.synapse.org/

InProv [BYB+13]

InProv is an interactive visualization for file system provenance. It uses a radial-based tree layout to encode parent-child relations among multiple hierarchy levels (see Figure 3.18). With hierarchical and temporal aggregation, they can handle more than 60 thousand nodes. Time-based hierarchical node grouping uses the idea of aggregating multiple timestamps into a single time frame. An overview timeline at the bottom in Figure 3.18 points out changes over time and additionally allows users to adjust the range of a time frame that is displayed in the main view, while intermediate steps are added to the right. Their encoding approach to show the relations between hierarchy levels requires training to comprehend. Relations among hierarchy levels are encoded with different colors than the sectors they connect to. This is potentially confusing and users may require additional time to familiarize themselves.

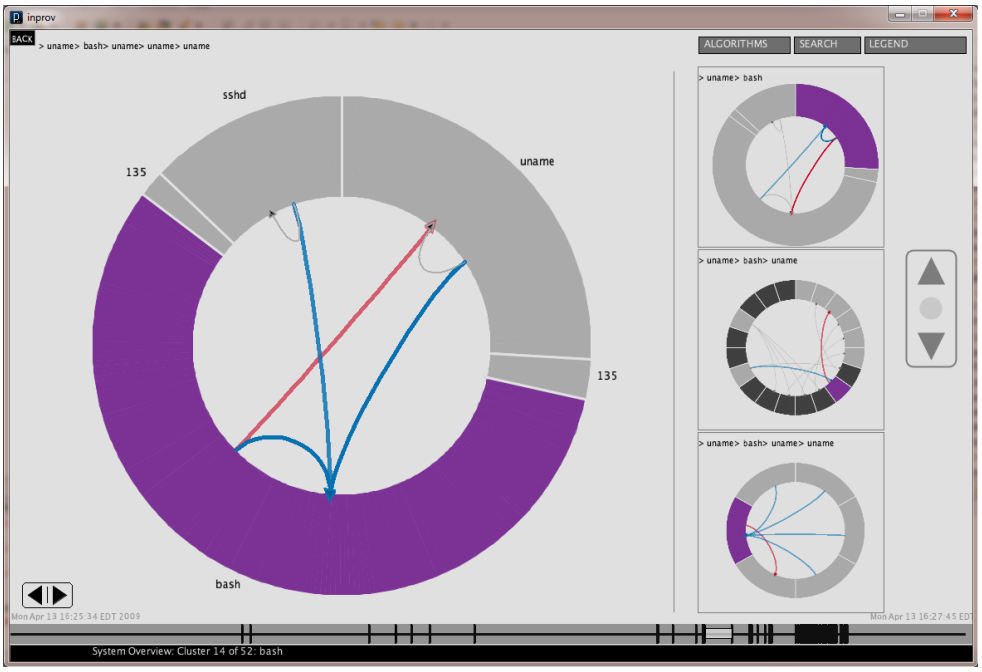


Figure 3.18: *A screenshot showing a time frame of the file system provenance graph in InProv. (Figure from [BYB+13].)*

Provenance browser [ABL10]

This approach presents a multiple view visualization for workflow provenance acquired with the Kepler provenance collection plug-in. The provenance model builds upon workflow traces that are stored in the *XML* format. In Figure 3.19, the XML structure is displayed in the top left, allowing users to navigate data items. In the main view, the execution trace is rendered as a node-link diagram of data items (circles) and processes (squares). In the

bottom left, details of the currently selected data item are listed. Their solution cannot display graphs with hundreds of nodes initially, though users are able to narrow their search by making use of a formal query language. Query results generate new subgraphs in new windows, reducing visual complexity of the graph. The downside of writing queries is that they break interaction-based navigation and moreover, they are not trivial to formalize.

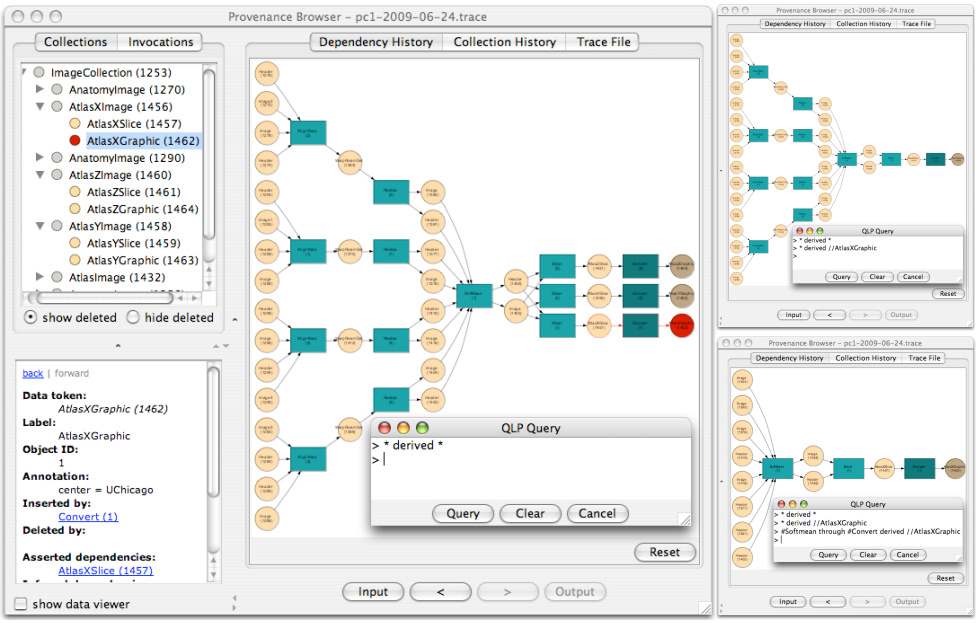


Figure 3.19: *A screenshot showing the XML structure (top left), details on demand (bottom left), and the provenance graph (middle) as well as two different query-resulting subgraphs (right) in Provenance Browser. (Figure from [ABL10].)*

Provenance Map Orbiter [SM11]

The *Provenance Map Orbiter* is a graph visualization that accepts OPM data, and from it, generates a node-link diagram that offers interactions to explore the provenance graph in a single view. With the use of graph aggregation techniques, a graph is initially compressed to a high-level overview. At certain magnification, semantic zoom adjusts the representation of atomic nodes and aggregates (see Figure 3.20). Additional reduction of visual complexity is achieved through attribute-based search and filter operations. Users can drill-down to get details on demand, while non-relevant context is collapsed into aggregates automatically. The rich and interactive feature set that this approach offers makes it stand out among most existing provenance visualizations. However, visual encoding is suboptimal, and edge crossing as well as routing pose problems that leave a lot of room for improvements.

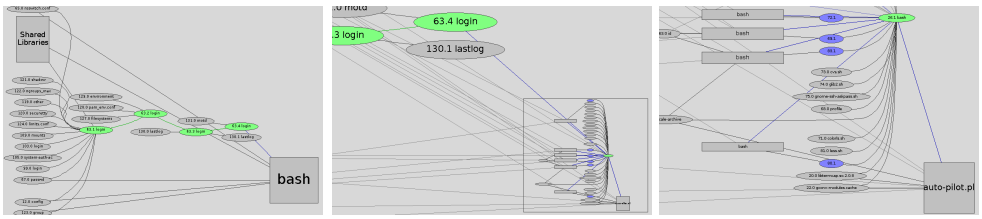


Figure 3.20: *Three views show an increasing level of semantic zoom on the provenance graph in Provenance Map Orbiter. (Figure from [SM11].)*

Network Provenance [CPC+12]

The work of Chen *et al.* describes a visualization approach that promises to handle large-scale data provenance. Their visualization is implemented as a *Cytoscape*[11] plugin, a popular JavaScript library for analysis and visualization. Their provenance model is again based on the popular OPM. Due to incremental loading and neighbor clustering, they make sure that they can handle large graphs. Figure 3.21 shows the node-link representation which however suffers from overlapping text labels and suboptimal visual encoding. Note, the graph contains redundant branches that are drawn repetitively. Semantic aggregation such as motif-based compression would let them summarize redundant information that eventually leads to a cleaner overall graph.

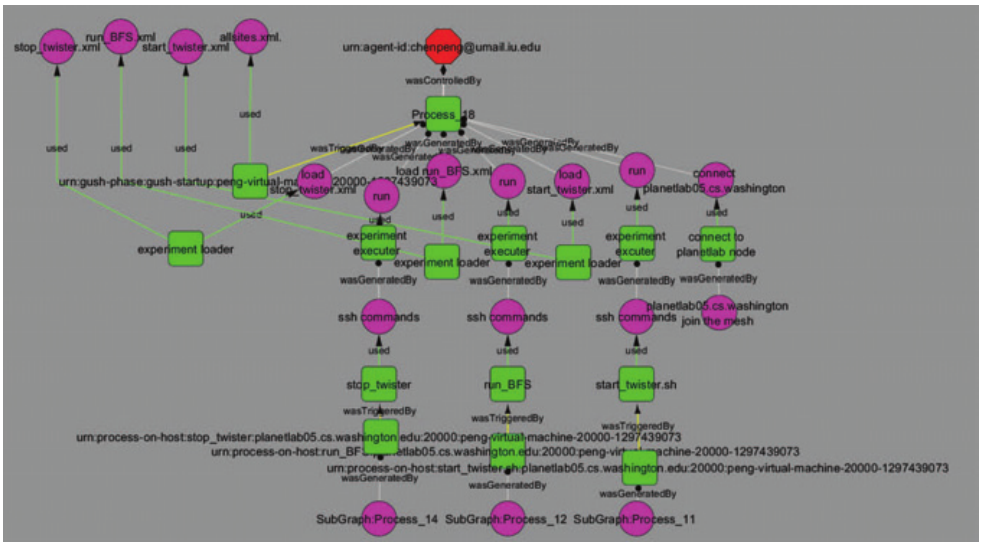


Figure 3.21: *A screenshot showing a provenance graph that contains data products (magenta circles) and processes (green squares) in the Network Provenance Visualization. (Figure from [CPC+12].)*

[11] http://www.cytoscape.org/

3.4.1 Discussion

Other approaches [HC07, RS07, PHG+13, SYH+13] that have been published are even more simplistic when compared to the visualizations above. They do not bring significant added value, but rather share the same limitations. The reviewed visualizations are static, lack modern visualization techniques, and cannot deal with moderate to large-sized graphs. Visual encoding is often simplistic and an overly use of text labels that lead to visual overhead and clutter can be observed. From the background on biomedical workflow provenance, the elicited users tasks, and related work, we can derive two major challenges that are crucial to the design of provenance visualizations: (1) provenance graphs are large and grow very quickly, and (2) they contain time-dependent information. Both are central to our user task-driven visualization concept which we will introduce in the next chapter.

Chapter 4

Concept

The main objective of this thesis is to implement a provenance visualization for Refinery that can deal with (1) quickly growing and (2) time-evolving provenance graphs. These two major challenges are drawn from the many issues that current state-of-the-art approaches do not address appropriately (see Section 3.4). In this chapter, we present a high-level concept for each method that is used in the implementation to overcome this challenges. The design choices were made with concern to the tasks described in Chapter 2.

First, in Section 4.1, we choose an appropriate graph representation and design glyphs that convey information about node attributes, changes, and causality. The core concept comprises a combined aggregation approach to abstract data and in turn reduces the visual complexity. In order to control the level of semantic aggregation of each node, the graph layout is dynamically adjusted by the means of an attribute and action-driven DOI function.

Second, in Section 4.2, an overall multiple view concept and interaction techniques that enable users to accomplish their tasks are presented. The provenance graph must provide basic navigation, interaction tools and a filter interface. Furthermore, the graph must simultaneously be shown as an overview to convey temporal analysis order. The DOI components that control the different levels of granularity in the graph require a view to adjust user interest. Other supporting views include a tabular-styled node info and an interface to switch between color schemes.

4.1 A Node-Link Diagram Approach

Users need an initial overview (Task I) that displays a summary of the whole provenance graph which evolves over time. This view has to point out topological and attribute changes of workflow runs. In the previous chapter, we compared different graph representations concerning topology-based tasks. Node-link approaches are ideal for path-related tasks as the relations in matrix and implicit representations are not tightly coupled to their data points (see Section 3.3.1). Yet, node-link diagrams struggle to handle large graphs as they require non-trivial layout computations such as minimizing edge crossings and coordinate

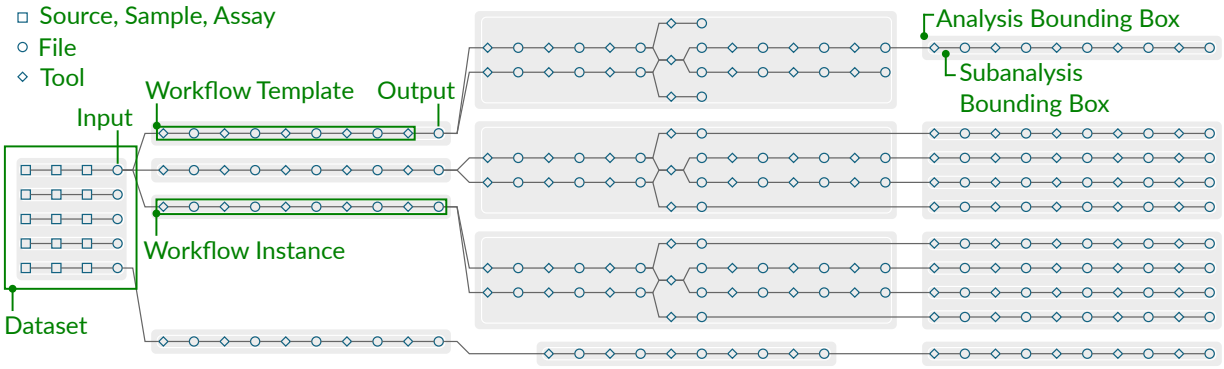


Figure 4.1: *A provenance graph is laid out in a column-based layout at the workflow level. As of Refinery's workflow execution process, a workflow template takes inputs from a preceding analysis results or the raw dataset. At workflow execution, a workflow instance that contains the produced outputs is created—also referred to as analysis. Workflows are indicated by the files and tools within their transparent and gray colored bounding boxes. Note, each workflow is enclosed by two bounding boxes, the subanalysis bounding box encloses the workflow itself and the analysis bounding box groups one or multiple subanalyses, here laid out as workflows, into an analysis.*

assignment. However, due to the hierarchical nature of Refinery's provenance graph, we can reduce the complexity by applying aggregation techniques which we will explain in the next section. For the layout, we choose a column-based approach which separates each depth level from each other and shows the overall flow in the graph more understandable (see Section 3.3.1). To further emphasize readability, the graph's orientation is specified from left to right.

Due to the chosen node-link diagram approach, we refer to vertices as nodes and to edges as links from here on. Figure 4.1 shows a conceptual drawing of a small provenance graph that will be continuously refined as this chapter progresses. At this stage, the dataset and the provenance graph is shown at its finest granularity—the workflow level. Obviously, the graph needs to be reduced to fit the resolution of modern devices and that node details can be shown at reasonable size. The means to reduce the complexity of the provenance graph are therefore discussed next.

4.1.1 Hierarchical Aggregation

As described in Section 1.1.5 and Section 3.3.2, Refinery's provenance data model distinguishes between three hierarchy levels (analysis, subanalysis, and workflow). Workflow files and tools represent the basic building blocks in a Galaxy workflow template. Tools are basically a synonym to *Data Transformation*, whereas files can be further differentiated into *Derived data*, *Source*, *Sample*, or *Assay* results. The former are the products of

direct workflow executions in Galaxy (see the introductory Section 1.1.5 about Refinery in Figure 1.1). The latter three are extracted from the dataset itself, the ISA-tab archive. Figure 4.1 shows these files and tools isolated in workflows that are indicated by surrounding bounding boxes. Yet, this level of granularity cannot be maintained for large graphs. Thus, we have to exploit the hierarchical nature of the Refinery provenance graph to apply a bottom-up semantic aggregation as conceptualized in Figure 4.2.

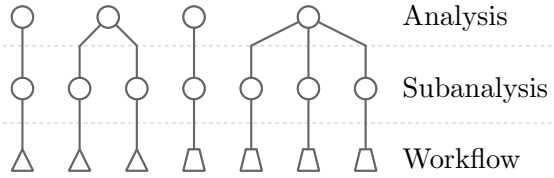


Figure 4.2: *The hierarchical aggregation of four analyses at every hierarchy level. The workflow level shows two different workflow templates either as single triangle or trapezoid shapes—referred to as workflow instance. From bottom to top: a workflow instance is aggregated (summarized) into a subanalysis, and multiple subanalyses are aggregated into an analysis.*

Eventually, the graph is compressed to manageable size (Task I). For example, only rendering the subanalysis level (see Figure 4.3 and Figure 4.4a) while hiding their workflows significantly reduces the visual complexity of the graph.

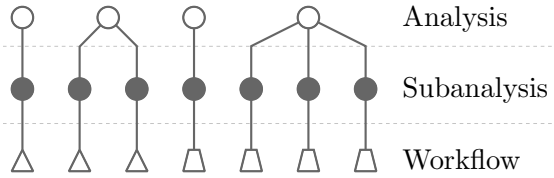
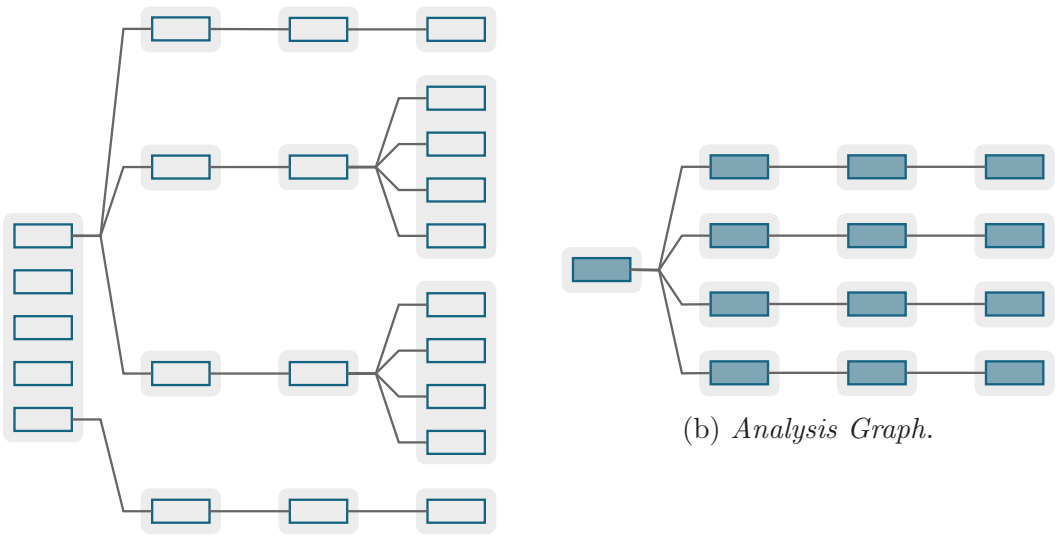


Figure 4.3: *This concept shows the rendering of a specific level, here subanalyses, while others are hidden. The rendered and thus visible nodes are indicated as filled nodes. The graph would then be laid out as in Figure 4.4a below.*

Respectively, we could also choose to explicitly render the analysis level as shown in Figure 4.4b.

As demonstrated above, hierarchical aggregation greatly reduces visual complexity through the introduction of separate hierarchy levels of varying levels of detail. Though, users need to be able to navigate within the resulting hierarchy levels. Such interaction techniques are commonly known as *expand* (Task IV) and *collapse* (Task I). Expand or collapse can either be controlled manually by the user, or automatically by a degree-of-interest function, which we will discuss later. As pointed out in Section 3.3.2, when navigating between these hierarchy levels it is necessary to aid the user with visual continuity and transitions to make them easier to follow. Neglecting this may cause the user to lose the mental map of the overall layout. We address this issue later in Section 4.1.5.



(a) *Subanalysis Graph.*

(b) *Analysis Graph.*

Figure 4.4: *A provenance graph shows both subanalysis and analysis graphs: Subanalyses (a) in a free layout are shifted vertically and all of them that belong to the same analysis are enclosed by a gray bounding box. Subanalyses are further aggregated into analyses (b). Both graphs show the single level rendering approach as explained in Figure 4.3.*

4.1.2 Motif-based Layering Approach

Traditional network motif discovery algorithms search for all permutations for a fixed amount of nodes, excluding isomorphs, which have the same structure but different appearances. Such motifs are commonly used to compress (sub)graphs while keeping the overall structure (see Section 3.3.3).

In the provenance graph, a motif is derived from the analysis with the earliest execution time in every layout column. The motif is characterized by the workflow template, subanalysis count, and the incoming or outgoing links. In simple terms, a motif reflects the basic structure of an analysis where multiple analyses share the same workflow template. Based on this definition, similar recurrent and parallel motifs can be discovered in a single breadth-first traversal during run-time.

In a subsequent step, and within every layout column of the graph, analyses that share the same workflow template are aggregated into a combined layer node. This aggregation approach creates an additional hierarchy level on top of analyses as shown in Figure 4.5—called *layer*. Note, the layering is constrained to workflow templates only as we do not want to restrict layering to a fixed size of subanalyses—called *soft layering*. In contrast, *hard layering* would constrain the aggregation to the full list of motif properties. To point out the differences in soft layering, we introduce numeric change metrics, a similarity measure for the deviating properties, in reference to the earliest created analysis in this layer.

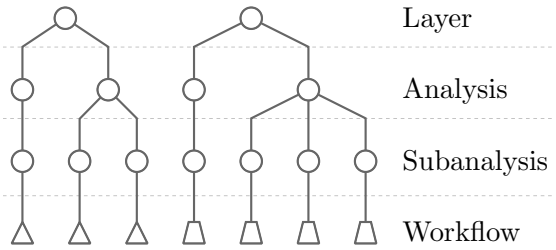
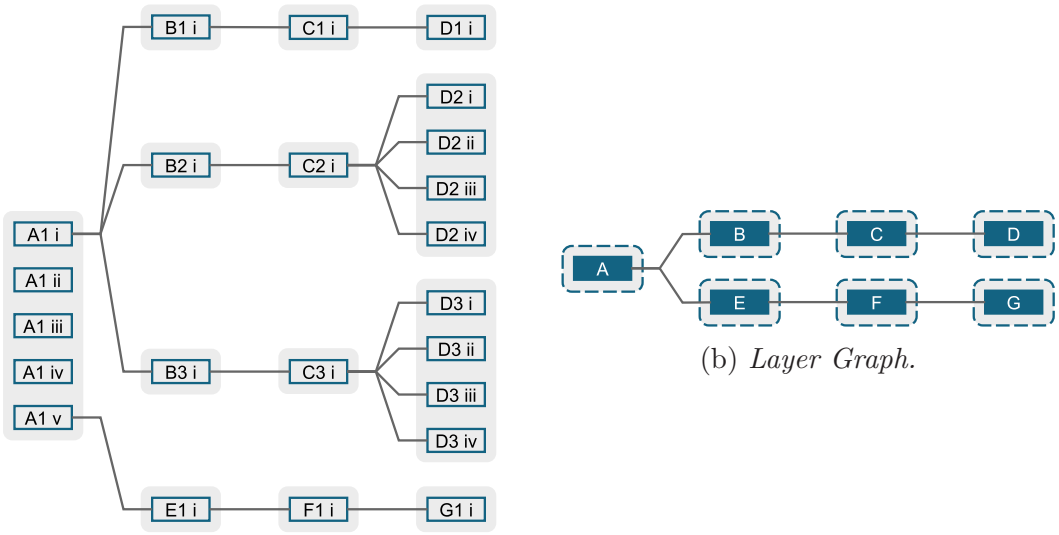


Figure 4.5: *Similar analyses are aggregated into combined layer nodes, adding the new layer level on top of the analysis level. As noted in Figure 4.2 before, there are two different workflow templates which are either shown as triangle or trapezoid shaped nodes. Here, two layers are created based on the workflow templates their analysis children are constrained to. The layers also group analyses although they do not share exactly the same subanalysis count:* 1 *and* 2 *in the left tree as opposed to* 1 *and* 3 *in the right.*



(a) *Subanalysis Graph.*

(b) *Layer Graph.*

Figure 4.6: *Soft layering does not consider the subanalysis count as for example* `D1` *and* `D2` *are layered although they are different at the subanalysis level (a). Each subanalysis is annotated with an uppercase letter that represents the corresponding layer, followed by an uppercase number for the analysis, and a lowercase roman numeral for the subanalysis itself. The provenance graph is shown at the layer level (b) where each layer node is annotated by an uppercase letter that corresponds with the subanalyses in (a).*

The application of our layering algorithm can be demonstrated best at the subanalysis level as shown in Figure 4.6a. For instance, the three subanalyses `B1 i` to `B1 iii` share the same workflow template and inputs, they are layered into layer node `B` (see Figure 4.6b).

Although `E1 i` was executed upon the same workflow, its input derived from `A1 v` differs and therefore a new layer `E` is created. Analyses `D1`, `D2`, and `D3` do not share the same inputs and their subanalysis count is varying too. However, the three subanalyses actually belong to the same motif sequence that originated from input `A1 i` and because their varying subanalysis count does not break layering, `D1`, `D2`, and `D3` are layered into `D`.

Analyses such as `D2` and `D3` have there changes calculated and encoded in reference to `D1`, in this case their subanalysis count is responsible for their change value of +3. With the separation of layers and the use of meaningful change metrics, users are now able to comprehend the development including time-dependent changes of a study (Task V).

4.1.3 Node Glyph Design

Every file in the provenance graph carries a set of attributes such as time or type. Regardless of type, each of the four levels (layer, analysis, subanalysis, workflow) also contains topological information about interconnections to predecessor, successor, parent, and child nodes. We follow the guidelines of creating multi-attribute glyphs as described in Section 3.2. Both attributes and relations are mapped to visual variables in a level-specific node glyph (Task II). To avoid visual clutter, the use of text labels is kept to a minimal degree and the glyph's visual space is limited to a constant cell size in the column-based graph layout.

A node glyph concept for each hierarchy level and type is illustrated in Figure 4.7. For the conceptual glyph design, we make use of visual variables, text labels, and numerals:

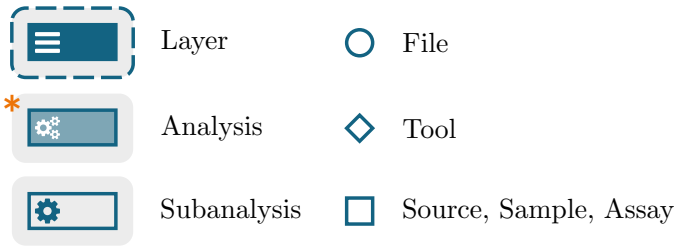


Figure 4.7: *The visual design of every glyph at each hierarchy level. Note that text labels and numerals are not shown here.*

1. *Shape:* A glyph uses its shape to convey the purpose of the node, either representing semantic aggregation, a data-manipulating process, or a file that holds data. We chose a diamond to represent the underlying processes in a tool, a square for preprocessed data, and a circle for derived data. Aggregates are indicated by slightly bigger rectangles.
2. *Icon:* To improve the distinctness among aggregation types, we append different symbols to the rectangles.

3. *Bounding Box:* In order to emphasize the boundaries of aggregated nodes (layers, analyses, and subanalyses), a semi-transparent bounding box is placed underneath. For better distinction between layers and analyses, we add a dashed outline to the layer bounding box that indicates the compound aggregation process. When a node is expanded into its child nodes, the box grows in size to cover all nodes belonging to their parent.
4. *Brightness:* As the graph's global topological order is predefined, an alternative way to convey temporal order is required. The relative order, and in turn the execution date and time of analyses including all sublevels, is encoded by mapping time to brightness. The brightness ranges from black to white, where the former represents the most recent and the latter the earliest analysis execution time. As layers contain analyses that were created at different time steps, a black-to-white gradient is used to represent multiple analyses within a time frame.
5. *Text Labels:* Every aggregated node such as layer, analysis, or subanalysis has its underlying workflow name added to the bounding box. As files and tools mostly carry generic node attributes, a meaningful encoding is hard to find. Hence, the default name attribute, which all files or tools have, is chosen by default. However, users should be able to switch between all attributes available, addressing Task II.
6. *Numerals:* We encode the number of incoming and outgoing edges as well as the number of child nodes for aggregated nodes by showing the corresponding count.
7. *Change Indicator:* Layered analyses that have different incoming and outgoing links or subanalysis count are marked with an asterisk, addressing Task V.

4.1.4 Filtering

We propose two filter interfaces to address Task III. First, we integrate Refinery's faceted-browsing interface to filter nodes by attributes such as tissue, drug, or cell type. Second, we introduce an interactive timeline view to filter analyses based on their creation time in Section 4.2.2.

As described in Section 3.1.4, the complexity of the graph can drastically be reduced through filtering. This results in more available drawing space needed for nodes that need to show more information than others. Nodes that are affected by a filter are either hidden or blended. Hiding nodes is equal to trimming the graph into a subset. Because this process might result in a sparse layout, we choose to recompute the layout from scratch. However, we also provide an option to blend (decreasing their opacity) to keep the graph's structure, and moreover, the mental map.

4.1.5 Dynamic Graph Layout

We choose the layered graph drawing method (see Section 3.3.1) to compute the graph layout, because it delivers satisfying aesthetics and fits our dynamic approach that requires recomputations during runtime. Because of our motif-based layering approach, the layout should align nodes at equal depth into the same column. Every node is mapped to one cell, while compound structures such as workflows occupy multiple cells at once.

Operations like filtering or navigating between hierarchy levels affect node placement and in turn the user's mental map. As explained in Section 3.3.4, there are two countermeasures that aim to minimize layout changes on the one side, and make use of transitions to point out actual layout changes on the other side.

In the provenance graph, workflows tend to recur frequently. It seems reasonable to render them in the same layout to emphasize on the recognition value of workflow templates. At the subanalysis level, nodes are reordered with a barycentric heuristic to minimize edge crossings. In graph theory, barycentric coordinates take the position of the preceding nodes into account. This means that a node's placement is computed as the mean of its predecessors—generally speaking, at the center of mass. This order is persistent and even the user should not be able to change it. In contrast, analyses and layers are not constrained to their position in the layout. This is due to the various combinations layers and analyses may appear through motif-based layering or filtering. They may be visible or hidden while they are combined or separated. The uncertainty that comes from user interactions may affect the aesthetics of the graph such as edge crossings. In order to deal with that, the layout for layers and analyses is recomputed at major user actions. However, to facilitate these updates, continuous and smooth transitions are used.

4.1.6 Modular Degree-Of-Interest Function

The combined use of aggregation techniques helps to reduce visual complexity on the one hand, but might also hide interesting information on the other hand. With high priority, users want to observe the factors that drive the development of a study, in particular workflow runs, reruns, and changes (Task VI). Insights might not be gained from the top level aggregate and one has to drill-down into deeper levels to reveal them. Based on graph attributes or a given user task, the overview graph must dynamically adapt to varying levels of detail.

We have reviewed the literature on DOI functions in Section 3.3.5. The DOI reflects the current level of interest to the node and controls its degree of semantic aggregation. Our modular DOI function incorporates both general interest components *time* and *change* as well as user actions such as *filter*, *highlight* and *selection* on a per node basis. These components carry a DOI value for each node at every level in the graph. Analysis creation time and change metrics, introduced in Section 4.1.2, are mapped to a continuous and normalized scale ranging from 0 to 1. Filtered, highlighted, or selected nodes have their

corresponding DOI value set to 1 and 0. In addition, users are able to adjust the weights of each individual component in a dedicated DOI components view which we explain in Section 4.2.3. The sum of the weights from all five components must be 1. The DOI for every node n is then computed as the weighted sum of each component's value v_i and weight w_i as follows:

$$DOI(n) = \sum_{i=1}^{n} w_i \times v_i \mid \sum_{i=1}^{n} w_i = 1.$$

As in Figure 4.8, the DOI of every automatically trigger compression and expansion among various hierarchy levels, addressing Task I, Task IV, Task V, and Task VI. The higher the DOI, the higher the level of detail that will be rendered.

Extraction vs. Expansion Figure 4.8 shows two unbalanced hierarchy level trees where the visibility of every node is controlled by the DOI. While the top layer `L1` is visible, `L2` is hidden in the right tree. Analysis `A1` has a low DOI so it remains hidden. However, `A2` has both their descendant subanalysis and workflow rendered. Note, as at least one analysis has a low DOI so that no expansion is triggered, its parent layer node will be rendered while all other descendants with a high DOI, are *extracted*. Another example of this specific case is also demonstrated in Figure 4.9. The extraction technique allows to hide uninteresting analysis paths (`A1`), while rendering relevant ones (`A2`) that are still attached to the layer. As for `L2`, both analysis branches exceed a minimum DOI threshold, resulting in a hidden layer `L2` that *expands* into separate analyses (or their descendants).

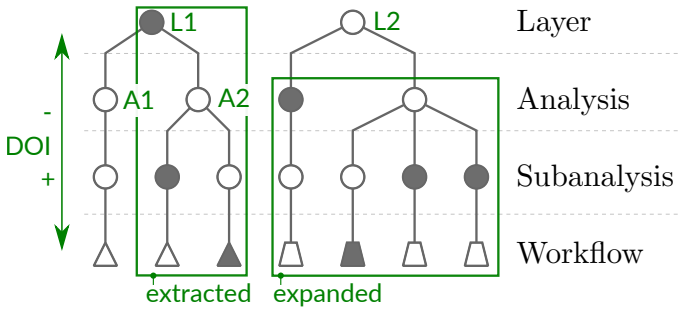


Figure 4.8: *The unbalanced rendering of two layer hierarchies shows visible (filled circles) as well as hidden nodes that are both controlled by the DOI function. Eventually, the graph would be rendered with varying hierarchy levels as shown in Figure 4.9. Note, the node labels do not correspond to the graph labels in Figure 4.6.*

This approach of unbalanced hierarchy level rendering resembles the idea of the focus+context technique originally introduced by Furnas *et al.* in their fisheye view example [Fur86].

4.2 Multiple View Visualization

In the second part of the concept, we describe graph interaction techniques and multiple support views that are linked to the main graph visualization. Several other related views such as a timeline, a bar chart, a details tab, and a facet browser provide interactive tools which are required to complement the visualization as a whole (see brushing and linking in Section 3.1.4).

4.2.1 Graph Interaction Techniques

The graph view offers a set of novel interaction techniques to navigate hierarchy levels and manipulate the appearance of the graph (see Section 3.1.4). We illustrate these techniques with respect to the user tasks and in an order that resembles the top-down approach of the popular visualization seeking mantra. Besides that, user interactions are applied either onto the whole graph or single nodes. The links are rendered as Bézier curves or straight lines that do not offer any form of interaction.

Overview First Initially, the graph is scaled to fit the drawing space by the application of a minimal zoom factor. Thus, readability to all visual elements in the graph is guaranteed.

Zoom and Filter Basic pan and zoom operations allow for basic viewport adjustments. For large graphs that do not fit into the drawing space, semantic zoom dynamically adjusts the level of detail (e.g., hiding glyphs details, bounding boxes, or text labels). Filtering is not applied on node interaction, instead an existing Refinery faceted-browsing interface is integrated and an analysis timeline view is implemented to support time threshold-based filtering (see Section 4.2.2).

Details on Demand The user can select nodes to list all attributes and details on demand in a separate view. Another technique that is typically applied to nodes is hovering to show tooltips which can be used to show node-specific attributes as well. To be able to navigate through the provenance graph, every node glyph implements both expand and collapse.

The methods described above are from a node-centric perspective. Yet, we have to address Task VI that enables users to investigate the derivation of files and transformation leading to a selected file of choice. With path highlighting, every node and link alongside this path has their visual representation highlighted by color. Nodes are not locked at their computed position, so users are also allowed to freely drag and drop them anywhere within the current view. These operations will be lost on layout recomputation.

A concept of the graph that shows our general approaches and the interaction techniques applied is depicted in Figure 4.9.

4.2.2 Analysis Timeline

As the graph's topology is not suited to convey temporal analysis order, this view creates a new perspective of analyses ordered by time (see Figure 4.10). The creation date and time of each analysis is mapped to position, the x-axis, while the y-axis shows the subanalysis count. We redundantly encode time to brightness. The timeline background is mapped to a white-to-black gradient that spans from left to right. Because the timeline view is rather small, it would struggle to display dozens of analyses on the x-scale distinctly and without overlapping. Hence we add semantic zoom (see Section 3.1.4) that enables the user to zoom into dense regions or when analyses are overlapping in the initial view. The timeline's second purpose is to act as a time-based filter. With two sliders, the start and end time can be adjusted, resulting in a time span that trims all analyses beyond it. When hovering over an analysis, the corresponding node is highlighted in the provenance graph.

Figure 4.10: *A timeline visualization that shows analyses ordered by date and time. The x-axis shows the creation time of analyses (orange vertical bars) that are mapped to a white-to-black gradient. The y-axis encodes the subanalysis count an analysis contains. Each of the four time-varying runs count a sequence of three analyses, synchronized to the example graph depicted in Figure 4.9.*

4.2.3 Degree-of-Interest Components

While every node carries a combined value of all five components, introduced in Section 4.1.6, we let the user adjust the weights of every component in a *Stacked Bar Chart* individually (see Figure 4.11). Each component's weight can be adjusted interactively, and moreover, every component can be turned on or off completely. While a components weight changes, the others adapt proportionally.

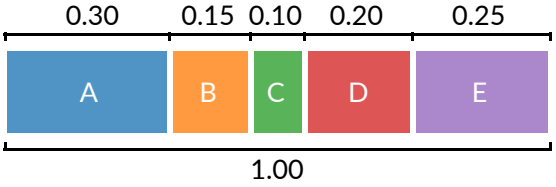


Figure 4.11: *A stacked bar chart illustrates five differently weighted DOI components.*

4.2.4 Node Info

In addition to tooltips (see Section 4.2.1), details on demand for each hierarchy level or node type are provided in an own tabular view, the *Node Info* tab, addressing Task II. For example, a selected node shows details about attributes, aggregation count, change metrics, and a file download.

4.2.5 Color Scheme

To further support perception of node types or attributes (Task II) such as time or workflow template, the *Colors* tab lets users apply custom color schemes chosen from a universal color picker. While links can be colored freely, node glyphs allow color coding by analysis execution time, workflow template, or node type.

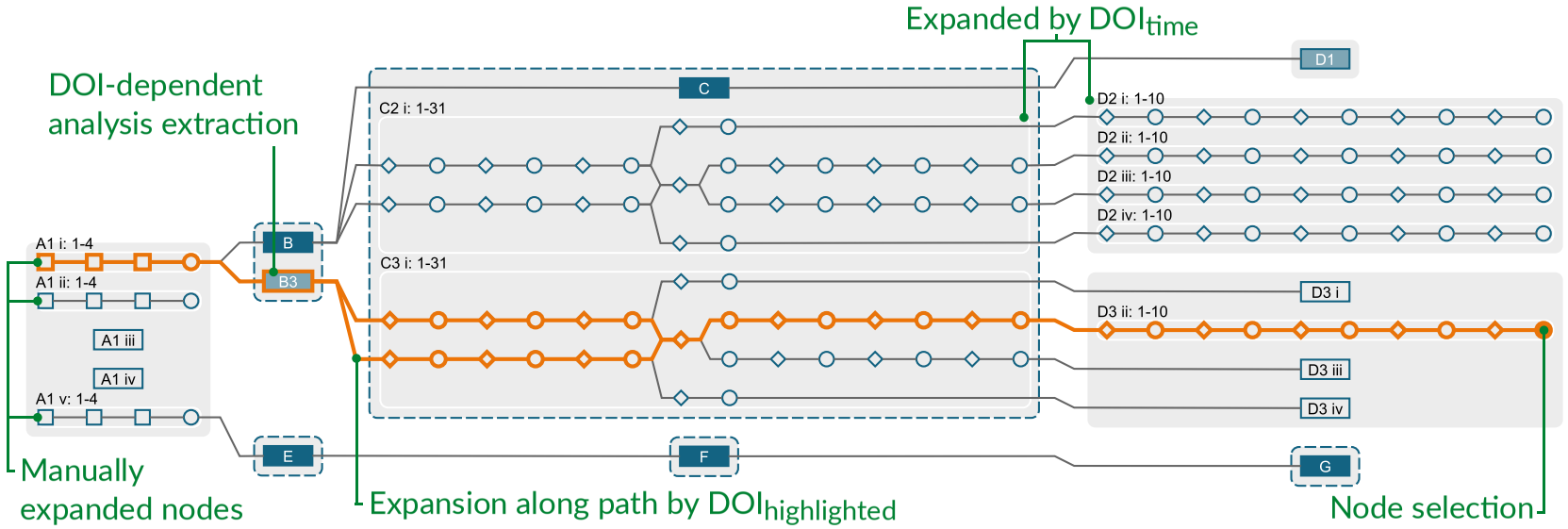


Figure 4.9: *A concept illustration of the provenance graph view shows the node-link approach, various hierarchy levels that are dynamically adjusted by the DOI. For example, the highlighted (orange colored) path triggers node expansions to the highest level of detail, while surrounding nodes are collapsed.*

Chapter 5

Implementation

This chapter describes the implementation of the conceptual design presented previously. As the provenance visualization is integrated into the Refinery Platform, we reintroduce Refinery from a technical viewpoint first, cover its system architecture and the technologies it is built on top of. In the second section, we describe the actual implementation of the provenance visualization, explain the modular design approach, the data model, and the graphical user interface.

5.1 Refinery Platform Architecture

Figure 5.1 shows an overview of the core components and the connections between them. Before going into technical details, we briefly summarize a typical work cycle in Refinery.

Refinery's work cycle starts at the file store, where private or public datasets are uploaded and stored. These repositories are saved in the ISA-tab file format. ISA is an abbreviation for *Investigation* (overall goals and means in an experiment), *Study* (a unit of research including characteristics and treatments), and *Assay* (analytical data of measurement and methods). This standardized format already contains information about the processes in a study. With the Galaxy workflow execution engine, analyses are created that extend these studies within Refinery. For every analysis created, workflow provenance is collected and added to the provenance graph of a study. The work cycle above includes a wide variety of internal and external components (e.g., Galaxy) that facilitate individual tasks brought together in a coherent system.

5.1.1 RESTful Style

Modern web applications like Refinery commonly build upon a *Representational State Transfer (REST)* [Fie00] architectural style that decouples implementations from the services they provide through an uniform *Application Programming Interface (API)*. Clients communicate over this interface to access and manipulate data resources that are located on the server typically over *Hypertext Transfer Protocol (HTTP)* methods (e.g., *GET*,

POST, *PUT*, *DELETE*) and a *Uniform Resource Identifier (URI)* that points to the data resource.

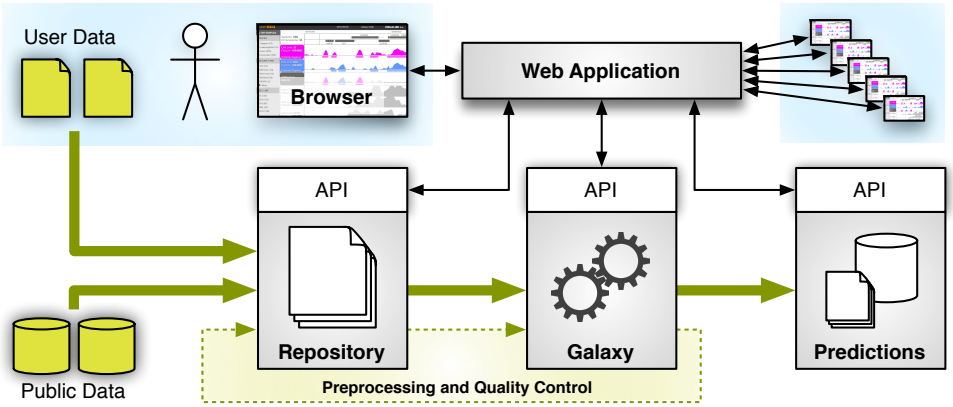


Figure 5.1: *Refinery is a web-based data management, analysis and visualization system that imports genomic data repositories and stores them on a local file server. It offers functionality to execute workflows via Galaxy and visual exploration tools to support reproducible research based on workflow provenance. Clients are separated from the resource and service providing server through a modern RESTful API design. (Figure courtesy of Nils Gehlenborg.)*

From a high-level perspective, Refinery's architecture follows the separation of concerns design principle by the means of separating the functionality into independent modules with each of them addressing a separate concern. The key aspect of such a modular approach is to encapsulate functionality or information into modules while offering a stable interface. Modules are independent from each other and therefore can be developed concurrently without worrying about side effects. Encapsulation protects the module implementation from modifications due to potential design changes, supports reuse, and maintains information hiding.

Another important constraint to REST is to partition modules into logical layers, most commonly known as presentation, business and data access layer. Besides the logical aspect, these layers may be physically separated what is called a multi-tier architecture. A basic client-server architecture features two tiers, the client and the server. However three tiers are often used to centralize the business logic and tighten security: client, middle, and data tier.

Refinery's core is built around the renowned *Model-View-Controller (MVC)* pattern that separates presentation (view), data access (model) and business logic (controller).

5.1.2 Django Model-Template-View Framework

Refinery's core is developed on top of *Django*[1], a popular free and open source web framework implemented in *Python*[2]. Django's *Model-Template-View (MTV) framework* resembles the MVC pattern by relying on the following core components:

- *Model*: The data access layer of Django describes data-centric tasks such as access, validation, and the relations among data. Basically, every model represents, or is mapped to, a single database table.
- *Template*: The presentation layer handles how the data is displayed to the end user.
- *View*: The business layer takes care of the data, or more precisely, which data gets presented.

The terminology of Django's MTV framework can get a bit confusing, compared to MVC, the MTV template represents the view and the MTV view the controller. One key advantage of Django is that it follows a self-contained app creation concept that follows the separation of concerns principle introduced above. An app provides distinct functionality to the project and is only accessible via API so it is decoupled from the rest of the project. A lot of the features that Refinery uses are already included in Django, e.g., object-relational mapping, an admin interface, or the robust templating system. Furthermore, Django takes care of secure user authentication, content administration, and caching.

5.1.3 Technologies

Besides Django, Refinery uses a set of web technologies to handle various tasks within the framework. All of them are open source. For the three-layer architecture, the most important ones are introduced in the following paragraphs. Figure 5.2 shows an overview of the most prominent libraries and technologies used.

Data Access Layer Refinery's back end uses the open source and object-relational *PostgreSQL*[3] database and manages uploaded files such as repositories in the ISA-tab file format in a local file store.

Business Layer *Solr*[4] queries are used to search and filter repositories with full-text matching capabilities.

Django offers a back end database API with which you can create, read, update, or delete data. As this API will only be used by local administrators (e.g., Django admin

[1] `https://www.djangoproject.com/`
[2] `https://www.python.org/`
[3] `http://www.postgresql.org/`
[4] `http://lucene.apache.org/solr/`

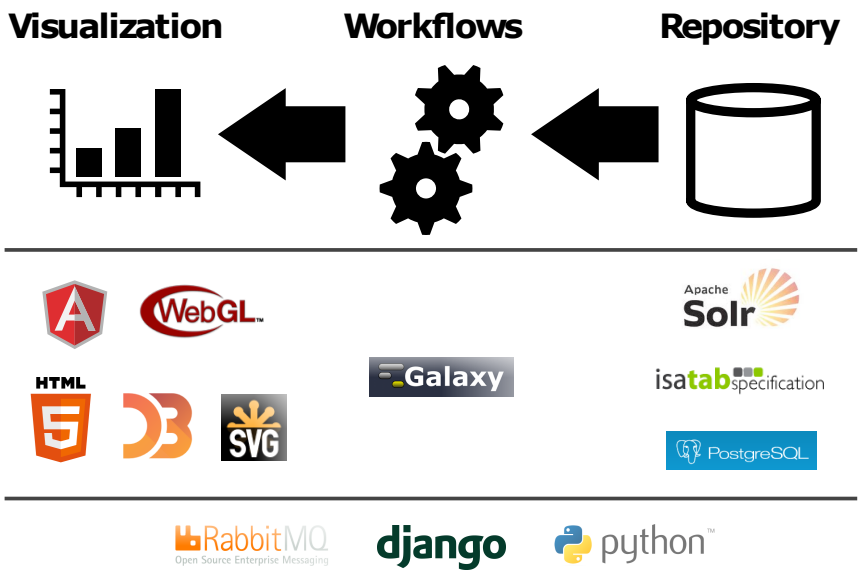


Figure 5.2: *Refinery is separated into three logical components: Refinery Visualization, Galaxy Workflows and Data Repositories. (Figure courtesy of Nils Gehlenborg.)*

interface), Refinery integrates *Tastypie*[5] to provide a REST-style API to the front end that forces user authentication. All Django models and views, most notably the provenance experiment graph are (de)serialized in *JavaScript Object Notation (JSON)*[6] format. Via HTTP methods (e.g., GET) and the given resource URI, front and back end modules communicate data resources to each other.

The Galaxy workflow execution engine is accessed via a Django model that holds a URL and an API key. The object returned represents the Galaxy instance which is used to interact with Galaxy workflows and their histories.

Presentation Layer Like modern web pages, Refinery's front end makes use of the following classic web technologies. *Hypertext Markup Language(HTML)*[7] is the standard markup language to write and render web pages. During rendering, the browser creates a *Document Object Model (DOM)*[8] of the HTML content that serves as API to access and manipulate the HTML structure. While HTML describes the structure and semantic content of a web page, *Cascading Style Sheets (CSS)*[9] describe how content is rendered based on numerous style properties. *JavaScript (JS)*[10] rounds up the classic web development standard for

[5] `http://tastypieapi.org/`
[6] `http://www.json.org/`
[7] `https://www.w3.org/html/`
[8] `https://www.w3.org/DOM/`
[9] `https://www.w3.org/Style/CSS/`
[10] `https://developer.mozilla.org/en-US/docs/Web/JavaScript`

which today's web browsers offer native support. JavaScript can manipulate the DOM, handle client-server communication, interactions, animations, or validations.

Furthermore, *jQuery*[11], a popular JavaScript library, is used to simplify DOM traversal and manipulation as well as event handling and animations. *Angular*[12] is a powerful Javascript framework used for the front end. It dynamically extends HTML with Angular directives and expressions. It comes with basic features such as data-binding, templating, form validation, page routing, reuse and dependency injection. *Bootstrap*[13] again is a JavaScript library that enhances responsive user interface experience.

Development Environment

The source code of Refinery is hosted in a *GitHub*[14] repository[15]. GitHub hosts repositories on the web and it is based on the popular version control system *Git*[16]. Refinery handles its versions in a main development branch, release tags, and feature branches such as the provenance visualization.

The implementation of Refinery is encapsulated into its own development environment. *Vagrant*[17] ensures that Refinery is set up identically in a virtual machine regardless of machine or operating system (e.g., *VirtualBox*[18]). Unfortunately, Galaxy is only available to Linux and Mac OS operating systems, that is why Refinery is limited to them.

Front end development, which we will discuss in in the next section, makes use of *Grunt*[19], an automatic task runner that handles minification, optional compilation, and automatic testing of the source code.

5.2 Provenance Visualization

The main functionality of the provenance visualization is written in JavaScript and jQuery. Its user interface, the provenance graph, and support views are integrated into the dataset front end of Refinery. The dataset view, analogous to all front end components in Refinery, is based on a Django template. The template basically defines the structure of the HTML content and embeds data resources requested from the Refinery back end. On top of that, JavaScript as well as Angular is used to add interactive user interface elements. The actual rendering and parts of event handling is implemented in the JavaScript library

[11] `https://jquery.com/`
[12] `https://angularjs.org/`
[13] `https://getbootstrap.com/`
[14] `https://github.com/`
[15] `https://github.com/parklab/refinery-platform`
[16] `https://git-scm.com/`
[17] `https://www.vagrantup.com/`
[18] `https://www.virtualbox.org/`
[19] `http://gruntjs.com/`

Data-Driven Documents (D3) [BOH11][20] that allows binding data to the DOM. D3 is built around the creation or manipulation of HTML, CSS, and *Scalable Vector Graphics (SVG)*[21] to implement sophisticated and fast visualizations on the web.

Note that this section only provides a rough overview of the implementation as it would go beyond the scope of this thesis to explain the programming languages, libraries, or code snippets used in detail.

5.2.1 Modules

The provenance visualization view is first loaded when the user selects the provenance view tab. At this point, the main module, which is named `Provvis`, triggers a five-step module pipeline that streamlines initialization, layout computation, motif-based layering, and rendering (see Figure 5.3). Each of these modules, including `Provvis`, is implemented as a self-executing anonymous function that acts as a closure. A closure stores a first-class function together with its lexical scope that can be accessed even when the function is used outside the scope. With this principle, we can keep the internal module state private while publishing a minimal API to communicate with other modules at the global scope.

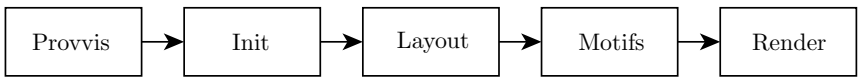


Figure 5.3: *The provenance visualization is decoupled into five modules. The modular design follows the principle of a separation of concerns design through the JavaScript module pattern. This design pattern guarantees an internal state and information hiding. The modules communicate over a public but minimalistic API.*

Obviously, each module operates on different data and implements individual functionality. As a result, inconsistent states or unforeseeable errors have to be dealt with individually. For example, the graph might not be acyclic or misses some nodes or links. That is why we added individual exception handling to each module to address specific failure states.

Before the actual module pipeline is initiated, Angular templates (note they are different to Django templates) are created at the global dataset scope. These templates define the HTML structure of the user interface (side- and toolbar) and canvas (main graph view) container. With Bootstrap, we add complex interface components such as tabs, an accordion, and drop-down menus to the side- and toolbar.

In the following sections, the responsibilities and important aspects of each module as well as specific algorithms will be explained. In addition, we continuously refer the data

[20] `http://d3js.org/`
[21] `http://www.w3.org/Graphics/SVG/`

model illustrated in Figure 5.4, which shows first-class JavaScript function objects and their relations among them.

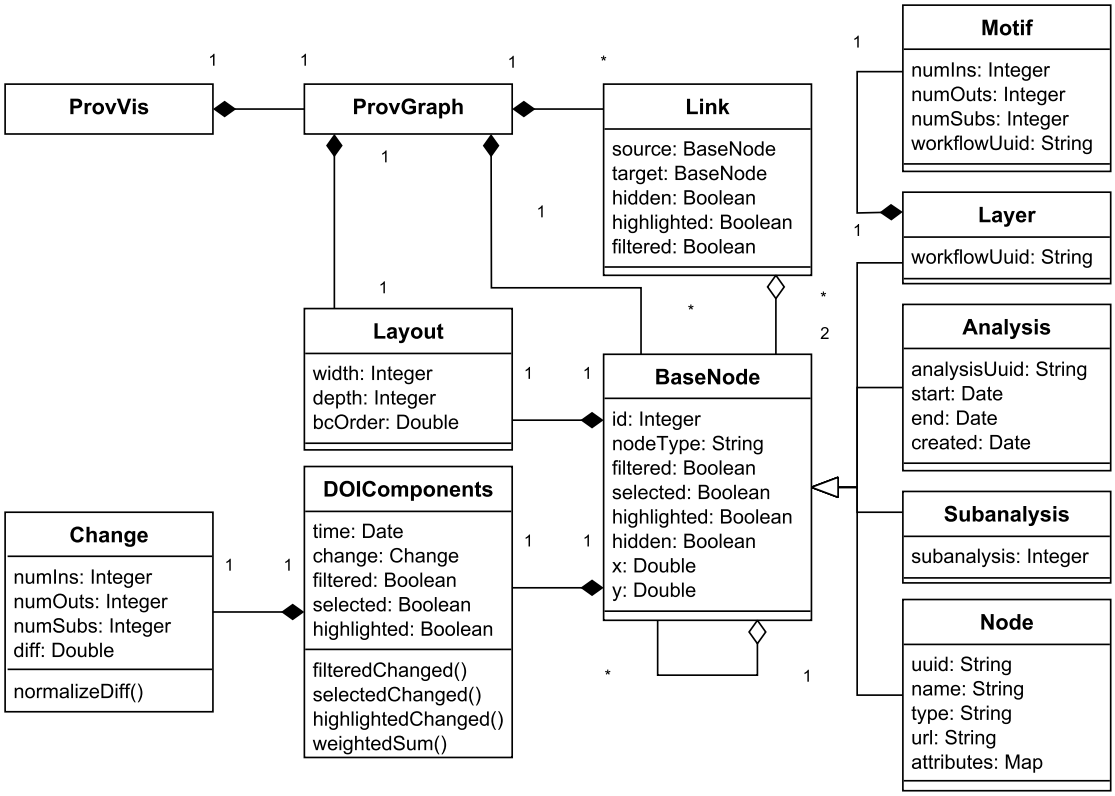


Figure 5.4: *The data model of the provenance visualization* `ProvVis` *shows nodes of each hierarchy level* (`Layer`, `Analysis`, `Subanalysis`, `Node`) *that all inherit from the generic node type* `BaseNode`. *Each node may belong to a single parent, contain multiple child nodes and* `Link` *objects. Every node stores topological information about their children's layout (e.g. size) in a specific* `Layout` *object. Every node has a implementation of the modular DOI function that controls its visibility. A* `Layer` *also influences the DOI by the* `Change` *metrics of a layered* `Analysis`.

Provvis

`Provvis` is the main module that is hooked into Refinery's dataset page. We implement a singleton design pattern to prevent multiple instances. Switching back and forth between the study's dataset and provenance view tab will not reload or change the provenance visualization.

As a first step, Provvis requests the provenance graph data via HTTP GET over the REST API that returns a JSON object. The actual request is processed by D3's json() callback function that asynchronously loads the data with the resource URI provided as parameter. As part of D3's visualization canvas creation, the function declaration redraw(), which handles basic geometric pan and zoom, is added to the visualization root element <svg>.

In a second step, the remaining modules are executed in the order Figure 5.3 shows. All modules, Init, Layout, Motifs, and Render are invoked by their public run() function. There is one exception to the fixed order of the module pipeline. When the user switches between layering methods (soft or hard), layers are recreated by re-executing Motifs.run() and Render.run().

Init

The public run() function of the Init module takes three arguments:

1. data: A JSON object of the ISA-tab dataset, that contains all files and tools, is requested over the Refinery REST API.
2. analysesData: The array of analyses is accessible at the global scope. It associates analyses with their workflow uuid.
3. solrResponse: The result set of filtered dataset and analysis results, including their list of generic attributes, is obtained by the execution of a Solr query.

This is the raw information that is needed to assemble all hierarchy levels, associate workflows, and extract attributes. Yet the given data structure is limited. For example, relations among files and tools only exist in backwards direction. The hierarchy levels analyses, subanalyses and workflows are not directly connected with parent-child relations, instead they are mapped to analysis identifiers. Workflows are not assigned to any node type except analyses. All these limitations result in a rather suboptimal data model of the provenance graph. The purpose of this module is to preprocess Refinery's stored provenance information and translate it into an appropriate graph data structure. First, a new Node object is created for each basic file or tool from data. Their attributes are assigned to them based on the solrResponse. We create actual Link objects that reference to the connecting pre- and succeeding nodes. Second, Analysis nodes and workflow templates are extracted from analysesData and their corresponding Subanalysis nodes are assigned. Third, the missing relation between workflow files and tools as well as their subanalysis parent is restored. From these object instantiations, a ProvGraph object is created that is returned and passed over to the remaining steps in the module pipeline.

Layout

In this module the layouts for both analyses and workflows are computed. We use the JavaScript library *Dagre* [pet15][22] that implements a layered graph drawing approach as described in Section 3.3.1. Dagre can compute the layout of medium sized graphs very quickly. The computation takes place at the client-side, meaning it makes a compromise between speed and layout aesthetics. Dagre initially transforms the graph into a directed acyclic graph (DAG) [GKNV93]. Analogous to layered graph drawing, nodes are assigned into ranks with the application of a network simplex algorithm [GKNV93]. Note, ranks are also referred to as layers or columns in the graph drawing domain. These ranks are then ordered based on two-layer crossing minimization [JM97, BJM04]. In a last step, the horizontal node coordinates are computed based on [BK02]. In our implementation, we add the nodes and links to a Dagre graph object and execute the layout computation by calling `layout()`. This function returns a map of identifiers with x and y coordinates. Both `x` and `y` coordinates are then added to `Analysis` and `Node` objects of the `ProvGraph`. Whereas the analysis layout may change due to user interactions, workflows are static to emphasize on their recurring templates. Subanalysis do not rely on Dagre, instead they are arranged in reference to their analysis parent node position and shifted vertically downwards. However, we reorder subanalyses based on the barycenter crossing heuristic to minimize edge crossings. The ordering of subanalysis nodes is stored in their `Layout` object.

Motifs

Next, we have a closer look on motif discovery and aggregate their corresponding analyses into layer nodes.

As already described in Section 4.1.2, there are two different methods: hard and soft layering. Hard layering constrains a motif to three conditions: workflow template, subanalysis count, and incoming and outgoing links, whereas only the workflow template is considered for soft layering. As a result, we introduce numeric change metrics that point out the differences of the two unconsidered properties. Basically, our implementation is divided into four steps: (0) preprocessing, (1) discover motifs, (2) apply layering, and (3) compute numeric change metrics. Based on Algorithm 1, we now illustrate these three steps:

In a preprocessing step, the analysis graph is brought into topological order, layered into column vectors. Note, in the graph drawing domain this step is commonly called layer assignment that groups nodes into a column (layer or rank) (see in Section 3.3.1). Our layering technique is based on the first created analysis within each column vector. Thus, each column is ordered by analysis creation time.

In breadth-first fashion (line 4), the algorithm traverses over the analysis graph. The

[22] `https://github.com/cpettitt/dagre`

first analysis automatically creates the first motif that holds information about workflow template, incoming and outgoing links, and subanalysis count. Whether an analysis conforms a motif is determined by the conditions in line 9 and 10. Note, we wanted to fit this algorithm to a single site, so this version exclusively checks for soft layering. Each analysis either creates a new `Motif` object (line 17) or is assigned to an existing one (line 15).

As every analysis has their motif assigned, the algorithm iterates over the column vectors a second time. The motif of every analysis, including their preceding motifs (determined via analysis connecting links) forms a sequential motif pattern we call `motifSeq + a.motif`. An analysis is then only assigned to an existing `Layer` object not only when its current motif, but also if the preceding motif sequence matches (line 23). Otherwise, a new layer is created in line 25. With this additional aggregation constraint, we avoid to solely layer analyses based on their workflow without looking back on their provenance of preceding layers.

In the third and last step, we calculate the numeric change metrics for every `Layer` descendant. The difference amount of incoming or outgoing links and subanalyses is computed and stored in the `Change` object of the current analysis.

Render

The rendering of the provenance graph, analysis timeline, and DOI components use D3 to build a SVG tree in the DOM. D3 follows a general update pattern, where data is bound to the DOM and manipulated through `enter`, `update`, and `exit` selections. Enter represents the creation of SVG elements, while `update` enables to bind or unbind the data from `enter` again, also referred to as data-join. `Exit` removes the elements from the DOM again.

All node glyphs are created by adding primitive SVG shapes, paths, and *FontAwesome*[23] icons to SVG group elements. Links are implemented as quadratic bezier curves that are defined by their control points.

All node glyphs, bounding boxes, and links are rendered up front, regardless if some of them are hidden in the initial overview. By default, nodes have their CSS `display` property set to `inline` when visible, and to `none` vice versa. With CSS, we are able to define a visual styles such as `fill`, `stroke`, or `opacity` to any SVG element. Multiple CSS class references have their visual properties defined in a separate CSS file. Analogous to the hierarchy levels, node types are added to the DOM in a top-down approach. Thus, geometric transformations that are applied to root nodes such as layers or analyses simultaneously affect their descendant child nodes (subanalyses and workflows) at no additional cost.

[23] `https://fortawesome.github.io/Font-Awesome/`

5.2.2 Interface and Interaction Design

In this section we describe the general user interface and interaction techniques for all views.

Toolbar A toolbar gives the user global control over the graph as a whole. We implemented hierarchy level shortcuts to quickly show the graph at a single hierarchy level (as conceptualized in Figure 4.4). A drop-down menu can be used to choose the file attribute that is mapped to each file node. Additional toggles let the user switch between filter methods, activate the sidebar, turn on automatic graph scaling, or view a help dialog that explains how to navigate through the graph.

Graph Interaction Design All user actions on graph elements are triggered by mouse events from single or double clicks on the left button, the scroll wheel, or dragging.

- *Node selection*: Click on the node glyph to select the current node.
- *Path highlighting*: Click on either of the two attached anchors. For example, the left anchor highlights all preceding links and nodes that lead to the current node. A click on the visualization background removes both, selection and highlighting. The highlighting changes both the color and stroke width of links and nodes.
- *Expand node*: Double click on the node glyph to hide the current node and reveal all child nodes.
- *Collapse node*: Click on the node's bounding box to navigate one level upwards. Double click to immediately go back to the layer level.
- *Drag node*: Hold down the left mouse button, move the node to a new position, and release it again. Dragging behavior is exclusive to layer and analysis nodes.
- *Pan*: Hold down the left mouse button on the visualization background to move the whole graph.
- *Geometric and semantic zoom*: Use the scroll wheel to zoom in or out of the graph. The higher the magnification the more node glyph details are revealed.
- *Scale graph to drawing space*: Double click on the visualization background.

Both collapse and expand initiate a recomputation of the layout that adjusts to the local changes triggered by the user. Because nodes are repositioned slightly and we want the user to keep the mental map, the changing nodes and links are animated in a one second long transition.

Filter and Update Both filters (see Section 4.1.4) trigger a Solr query to the relational database of Refinery. The returned result set, which contains the workflow files, is propagated upwards the hierarchy. When the filter method is set to *blend*, affected nodes are

alpha-blended by decreasing their value of the CSS property `opacity`. In case the chosen filter method is *hide*, the layout for this particular result set is recomputed. Analogous to collapse and expand, an animated transition is added to the translated nodes and links.

Analysis Timeline The implementation of the analysis timeline maps time to position on a D3 time scale. The position of the adjustable sliders is inversely mapped to data and time to trim analyses beyond these thresholds. Analogous to the graph visualization, we added pan and semantic zoom. Hovering over an analysis changes the color of the corresponding analysis bounding box in the graph view.

Modular DOI View The DOI view is realized as a vertical stacked bar chart that maps each component's weight to height (implemented as a D3 linear scale) as conceptualized in Section 4.2.3. Each bar provides an attached HTML input field that allows for individually weight adjustment.

At the start of the `Render` module, general interest DOI values of *time* and *change* are normalized on a linear scale and stored in the `DOIComponents` object. User actions such as *filtering*, *highlighting*, and *selection* are of Boolean type and thus set to value 1 or 0. The actual user interest, which incorporates both the node value and the components weight, is computed in the `DOIComponents.weightedSum()` function according to the formula in Section 4.1.6.

When one of the above mentioned user actions is triggered, the weighted sum for this particular node is recomputed and the updated DOI propagated up- and downwards to all hierarchy levels. This means that highlighting, filtering, and node selection not only affects the actual node, but also its lower and upper hierarchy levels alike. All affected nodes have their `DOIComponents` updated analogous to an observer design pattern. Finally, the every node in the graph is adjusted to represent the new level of interest. For this purpose, a state machine that maps the four hierarchy levels (states) to numeric DOI ranges (condition) is defined: layer: $0 - 1/4$, analysis: $1/4 - 1/2$, subanalysis: $1/2 - 3/4$, and workflow: $3/4 - 1$ (see Section 4.1.6). A single user action often affects multiple nodes, hence collapse/expand (transitions) are only triggered after all DOI values in the graph were updated and recomputed.

Node Info and Color Coding The Node Info tab provides detail on demand information for any selected node, this includes a file download link, attributes, and numeric change metrics. The Colors tab let users define and switch between custom color schemes. Colors may also be adjusted for nodes, links, highlighting, workflow templates, and node types.

Algorithm 1: Motif-based Layering

Data: Node-set A of analyses a
Result: Node-set L of layers l

```
1  A ← TopSort(A);                                  /* Topology sort analyses. */
2  C ← LayerAnalyses(A);          /* Group analyses into column vectors. */
3  L ← ∅;                                     /* Initialize layer node-set. */
   /* Breadth-first traversal of analysis node-set.                          */
4  foreach (c, i) in C do
5      SortByTime(c);           /* Sort analyses in column by start time. */
6      foreach a in c do
7          curMotif ← null; foundMotif ← false;
           /* Check if analysis conforms to an existing motif.            */
8          foreach m in M do
9              if (m.workflow = a.workflow) then          /* Soft Layering */
10                 if (Pred(a)= dataset and Preds(m)=Preds(a)) or
                   Pred(a)≠ dataset then
11                     curMotif ← m; foundMotif ← true;
12                 end
13             end
14         end
15         if foundMotif then                    /* Assign existing motif. */
16             a.motif ← curMotif;
17         else                                        /* Create new motif. */
18             newMotif ← CreateMotif(a);
19             a.motif ← newMotif; M.push(newMotif);
20         end
21     end
       /* Layering: Compare current and preceding motif sequences.     */
22     foreach a in c do
23         if PredLayers(a).get(motifSeq) + a.motif ∈ L[i] then    /* Assign */
24             L[i].get(motifSeq).push(a);
25         else                                  /* ... or create new layer. */
26             L[i].push(CreateLayer(a));
               L[i].get(motifSeq).motifSeq+ = a.motif;
27         end
           /* Compute change metrics (numSubs, numIns, numOuts) ...      */
28     end
29     return L
30 end
```

Chapter 6

Results

In this chapter, we elaborate the results and discuss the implementation of the provenance visualization. All datasets used were obtained from the Stem Cell Commons database as stated in Section 1.1.5.

6.1 Visualization Features

First of all, we demonstrate the features that let users accomplish their tasks as described in Chapter 2. Throughout this section, all implemented features of the provenance visualization are first introduced separately. Figure 6.1 showcases the combined use of them in a complete visualization example.

Graph Aggregation Strategies

The effectiveness of our combined aggregation approach is showcased in Figure 6.2 on real data do address scalability. With both hierarchical aggregation and motif-based layering, we reduce the raw graph to an overview that renders about 50 times less nodes than the graph fully expanded at the workflow level would, addressing Task I.

The different compression level achieved from soft and hard layering becomes apparent when we compare Figure 6.2d to the method of hard layering in Figure 6.3. Hard layering constrains motif-based aggregation to all three properties of a motif, workflow template, subanalysis count, and incoming or outgoing links. In this example, it requires about one and a half times of the drawing space of a soft layering layout. It is expected that analysts prefer soft over hard layering as it maximizes the compression solely based on the workflow template. The surrounding context of a particular area of interest must not always show every neighboring analysis. It gives enough context, when irrelevant nodes are abstracted to aggregated layers.

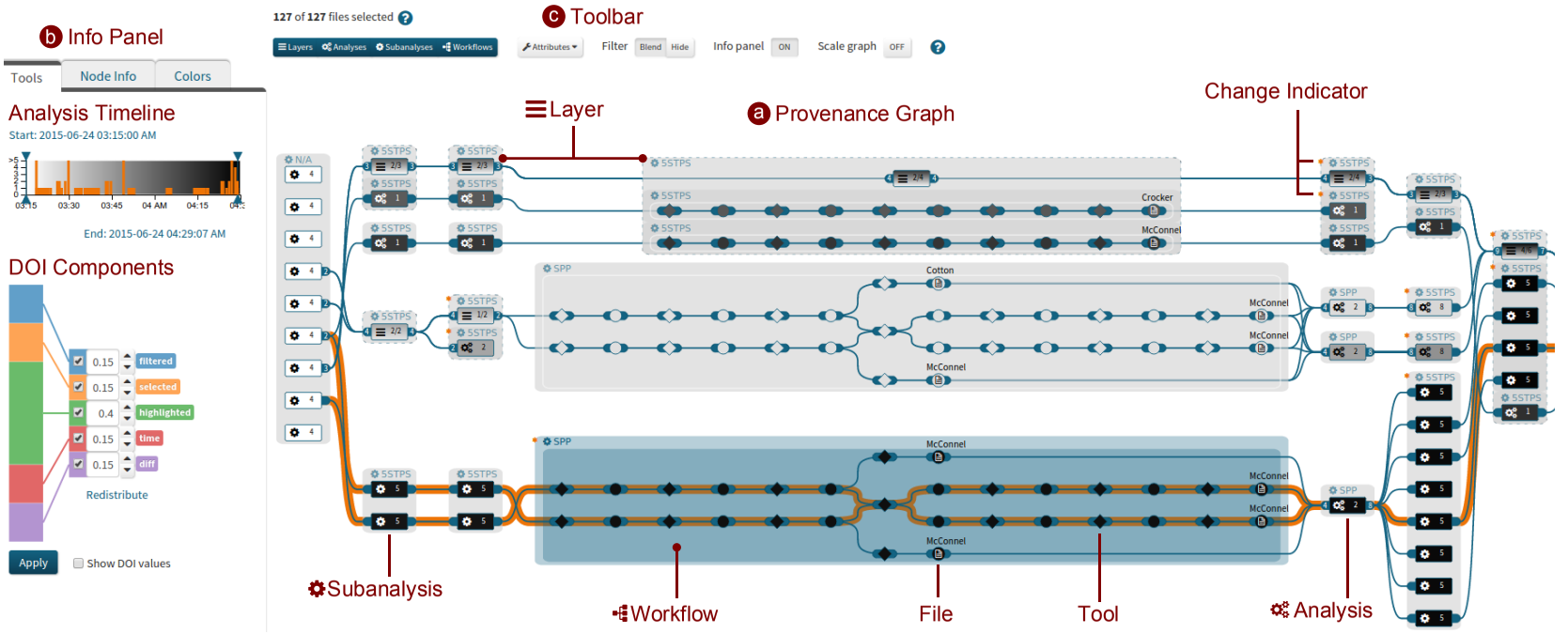


Figure 6.1: *A screenshot showing the provenance graph in (a) that is aggregated and filtered based on the selected workflow execution time. The weighted sum of all DOI components (b) dynamically adjust the level of semantic aggregation of every node. For example, in the top center of the graph (a), two horizontally aligned workflows show a combined layer node, where the top node represents the layer itself while two workflows are extracted based on their specific DOI. The toolbar (c) provides node type specific views and attribute mapping to nodes.*

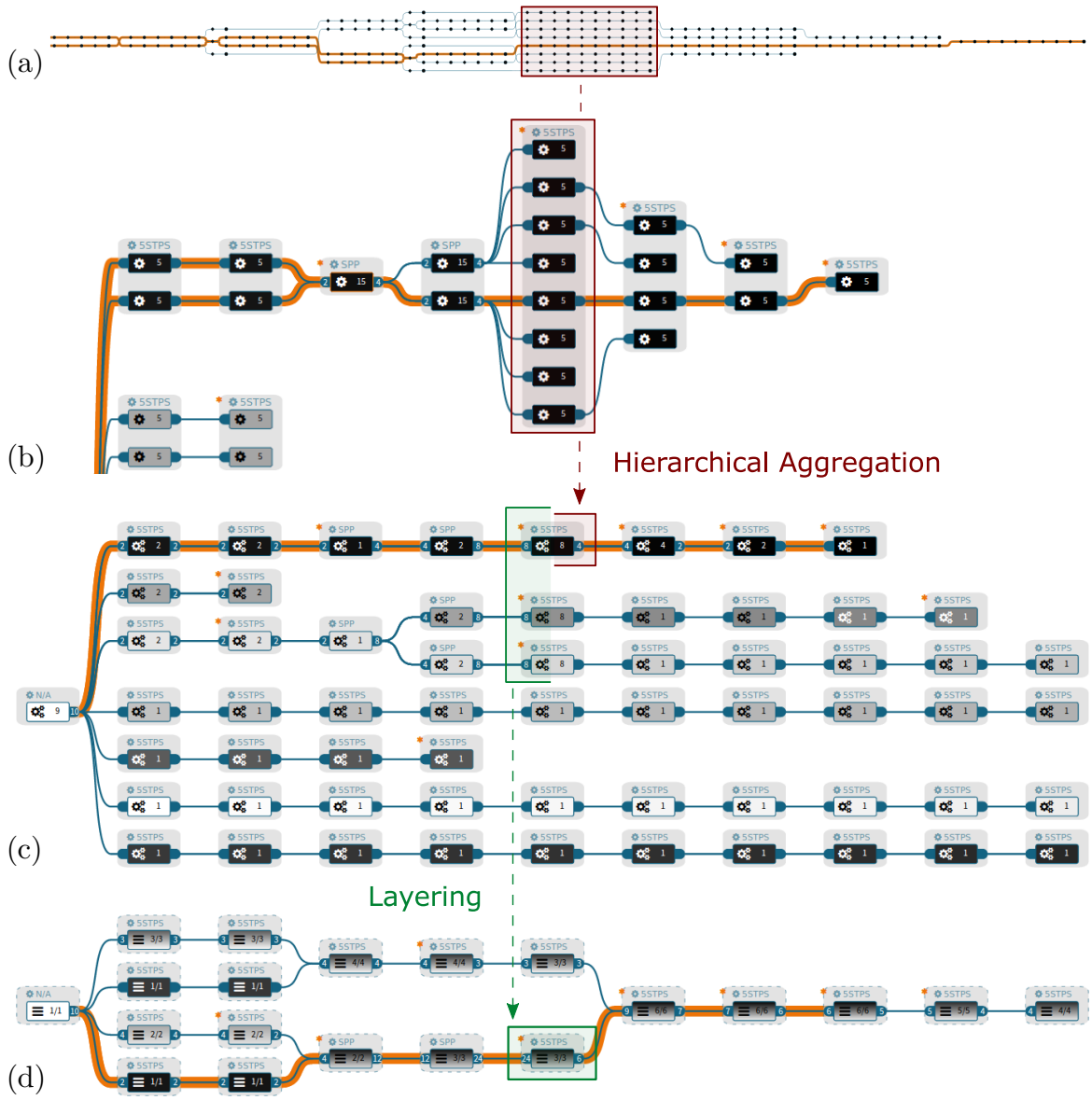


Figure 6.2: *The provenance graph is rendered in each of the four hierarchy levels: (a) workflows, (b) subanalyses, (c) analyses, and (d) layers. Both (a) and (b) only show a subset of the graph, mainly the context around the highlighted path. The example graph contains 1100 files and tools. Through hierarchical aggregation the visible node count is reduced to 100 subanalyses and 60 analyses. Motif-based layering further compresses the graph into 20 layers.*

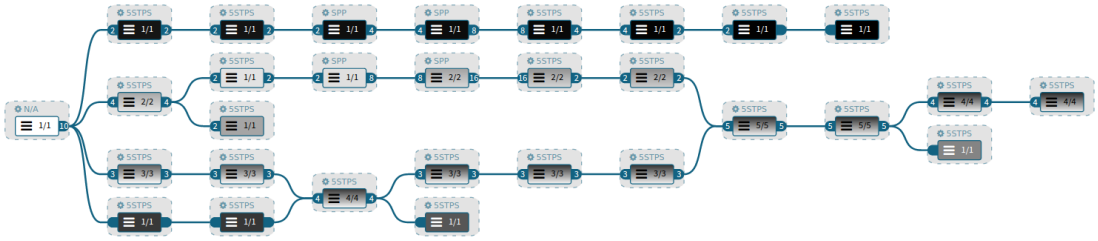
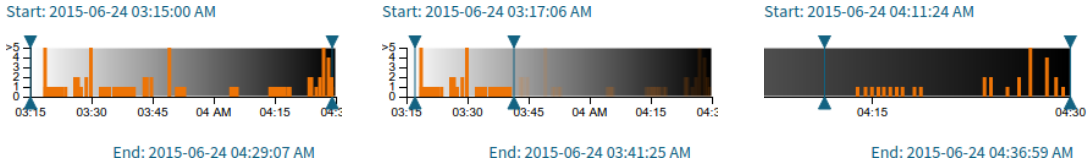


Figure 6.3: *The provenance graph, rendered with hard layering results in 29 layer nodes.*

Analysis Timeline

Users can determine the temporal development of the provenance graph in the analysis timeline view, which addresses Task I by providing an overview perspective of analyses ordered by creation time. In addition to this temporal overview, two sliders can be used to apply a time threshold-based filter. Figure 6.4 depicts three distinct filter configurations in which the complete, an earlier, and a later time frame is selected.



(a) (b) (c)

Figure 6.4: *Three screenshots are showing the analysis timeline filter in three different configurations, (a) the initial and complete overview, (b) the first 25 minutes, and (c) the last 15 minutes combined with semantic zoom. The top left and the bottom right text labels correspond to the lower and upper threshold respectively.*

Path Highlighting

With the functionality to highlight preceding or succeeding paths, users are now able to investigate the cause or the derivation of particular files and tools in the graph (Task VI). Every node and link alongside a path has their color changed to orange with an increased stroke width (see Figure 6.1).

Filter

We implemented an analysis timeline filter and integrated Refinery's attribute filter to address Task III. The filter trims uninteresting parts of the graph and can be applied in two ways, blending or hiding. The results of both methods are shown in Figure 6.5.

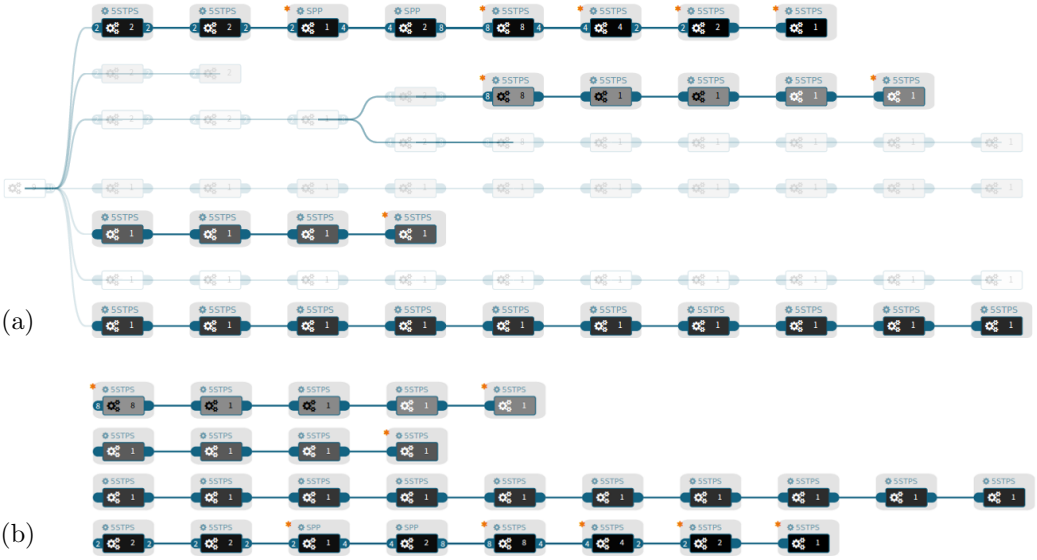
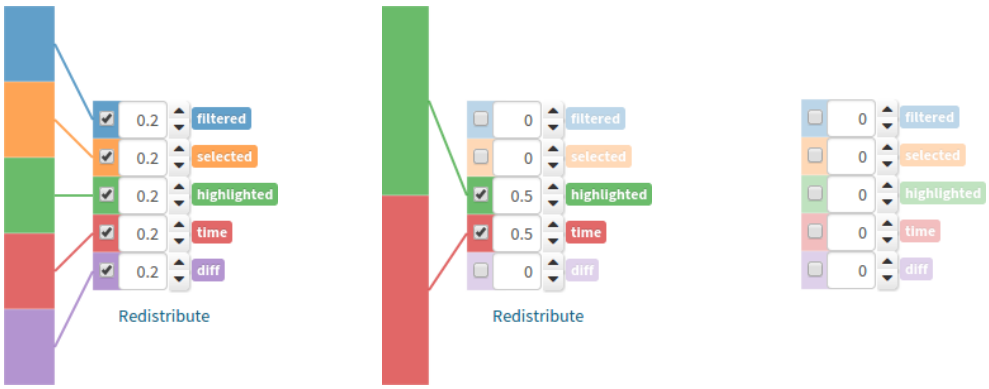


Figure 6.5: *The filter configuration for both graphs is derived from Figure 6.4c. While (a) uses blending, (b) hides affected nodes and recomputes the layout.*

Degree-Of-Interest Components

Users want to start out at a high-level overview as illustrated in Figure 6.2d. Though they are able to manually navigate through the graph, addressing Task IV, they are often uncertain where to start. To address this, we provide a flexible DOI function which lets them incorporate their interest into the representation of the graph. As illustrated in Figure 6.6, users can choose from five distinct aspects and weight them.



(a) (b) (c)

Figure 6.6: *Three screenshots are showing five DOI components that are weighted as follows: (a) balanced, (b) individually, and (c) turned off. When a configuration has been found, every node in the graph dynamically adapts to its new interest level.*

Node Attributes

Attributes and detail information about nodes can be acquired in multiple ways (see Figure 6.7). Semantic zoom incrementally reveals visual and textual details of all node glyphs (Task II), primary text labels can be chosen from a drop-down menu, and tooltips allow quick access to the most important file attributes. Furthermore, upon node selection, the user is able to list all attributes, changes (Task V) and graph-specific properties as tables in the sidebar tab.

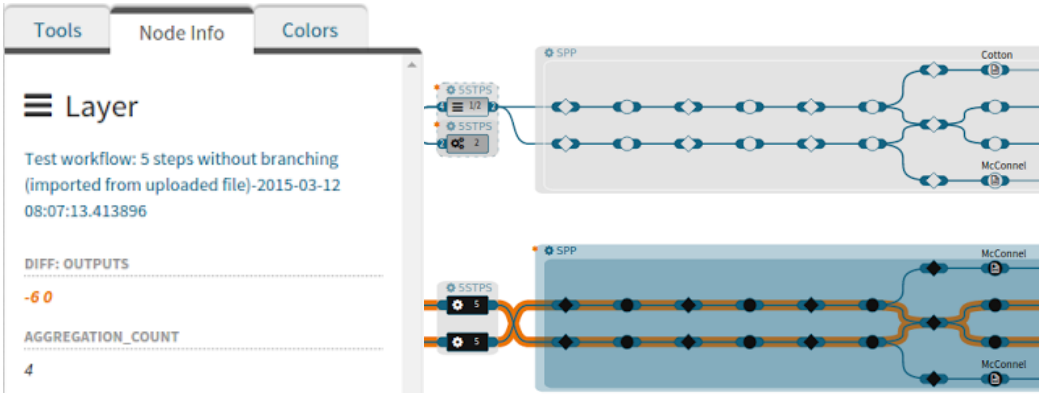


Figure 6.7: *A screenshot showing the Node Info tab that lists details about a selected layer node, including a link to the actual workflow template, its change metrics and aggregation count. At the right, various glyphs of all four hierarchy levels are shown within a subset of the provenance graph. The increased zoom level reveals all details such as icons, text labels and numerals.*

The color scheme interface is depicted in Figure 6.8.

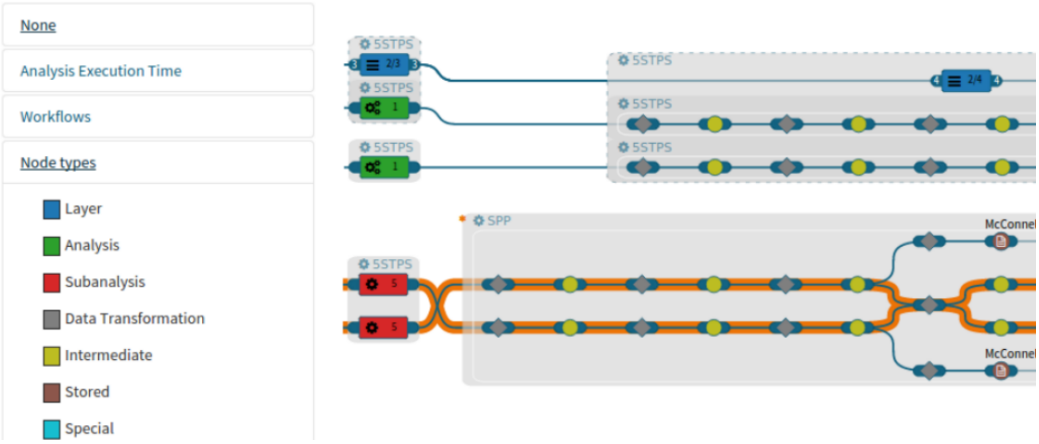


Figure 6.8: *A screenshot showing node type-based color coding. The user can also switch to color schemes that encode workflow template or analysis creation time.*

6.2 Use Cases

We demonstrate the value of our visualization by using the above feature set to accomplish distinct tasks in three different use cases.

Use Case 1: Find a Derived Path

The user wants to find all derived transformations and results for a recently created analysis (see Figure 6.9).

First, the user zooms into the timeline to get a detailed perspective on recently created analyses (1). After switching the filter method to *hide*, the user adjusts the lower threshold to trim all previous analyses (2). The user then chooses a particular analyses from the remaining graph and highlights its outgoing path with a click on the right node anchor (3). At this point, the user could either manually drill-down into each analysis that is highlighted or shift 100 percent weight to the DOI component *highlighted* (4). The user decides in favor of the latter strategy which automatically adjusts the layout by expanding all nodes within the highlighted path.

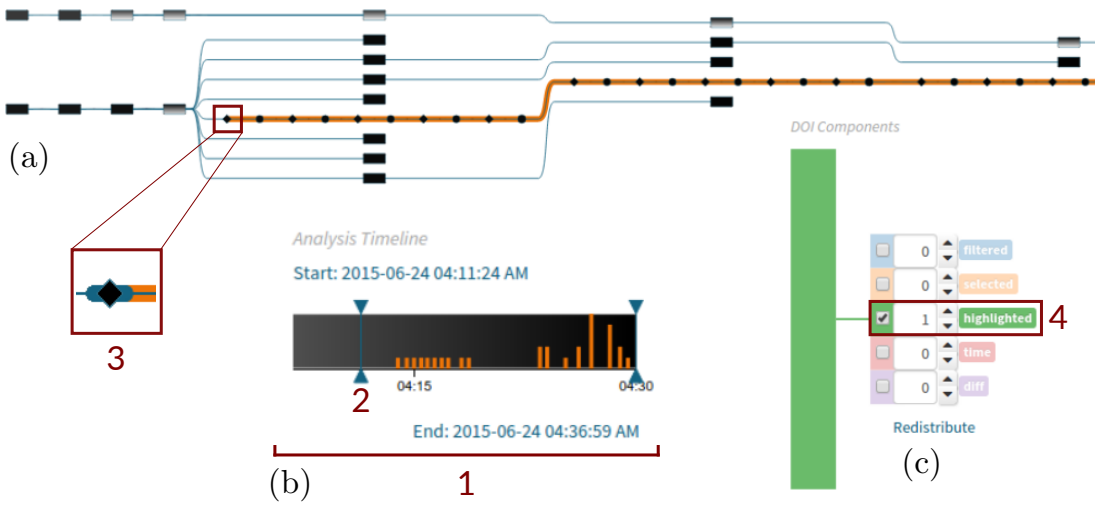


Figure 6.9: *A screenshot showing every step and the resulting overview graph (a). Files and tools alongside the highlighted path are expanded while its surrounding context is compressed to the layer level. The analysis timeline is shown in (b) and the DOI configuration in (c).*

Use Case 2: Investigate Change

The user wants to find out why a recently added analysis is not aligned to a former analysis (see Figure 6.10). Note, this use case requires deeper knowledge about motif-based layering rules that may be difficult to understand for new users.

By using the same analysis timeline window (1) as in Use Case 1, the user now instead manually expands the analysis of interest by double clicking on the node glyph, drilling-down to the workflow details (2). The surrounding bounding box encloses eight instances of the same workflow template. Compared to the layer (3) above, the user finds out that the workflow templates for both layers (2 and 3) are the same—called *5STPS*. And although the analysis change indicator of the analysis in context (2) points out dissimilarities, it is not relevant as it refers to another layer than (3)—hidden through filtering. Consequently, there must be another reason that causes the separation of these two layers. Thus, the user investigates both preceding layers (4 and 5) and finds out that their workflow templates differ—*SPP* at (4), and *5STPS* at (5). As a result, the investigated analysis (2) is separated from its top layer (3), because their predecessors originate from two very different layer sequences (4 and 5) where even their preceding analyses conform to different templates.

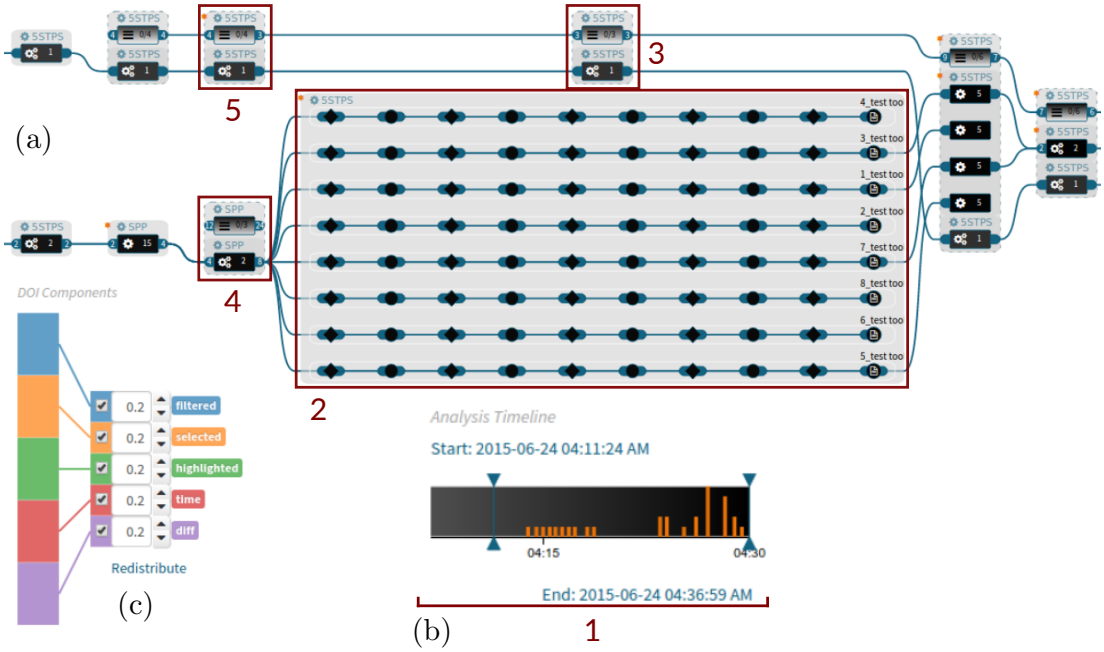


Figure 6.10: *A screenshot of the provenance graph view (a) showing the layers of interest (2 and 3) as well as their preceding layers (4 and 5) that cause the separation. The analysis timeline is shown in (b) and the DOI configuration in (c).*

Use Case 3: Review Reproduced Study Result

The user wants to investigate the files and transformations leading to a correctly reproduced result (see Figure 6.11).

The user shifts the focus on recent analyses by blending out the first third of the time frame (1). To validate a particular result of an analysis, the user manually drills-down into the workflow level (2), and selects the left anchor of the file (3) to highlight its preceding

path. In addition, the user configures the DOI components to greatly emphasize interest on this selected node and the highlighted path (4). This results in a partial expansion of the preceding layer (5), where one analysis is hidden (due to no change and low DOI) and their three siblings are extracted (6). Not a single analysis alongside the highlighted path differs, hence, the user can be sure that this workflow run has correctly reproduced the original study result.

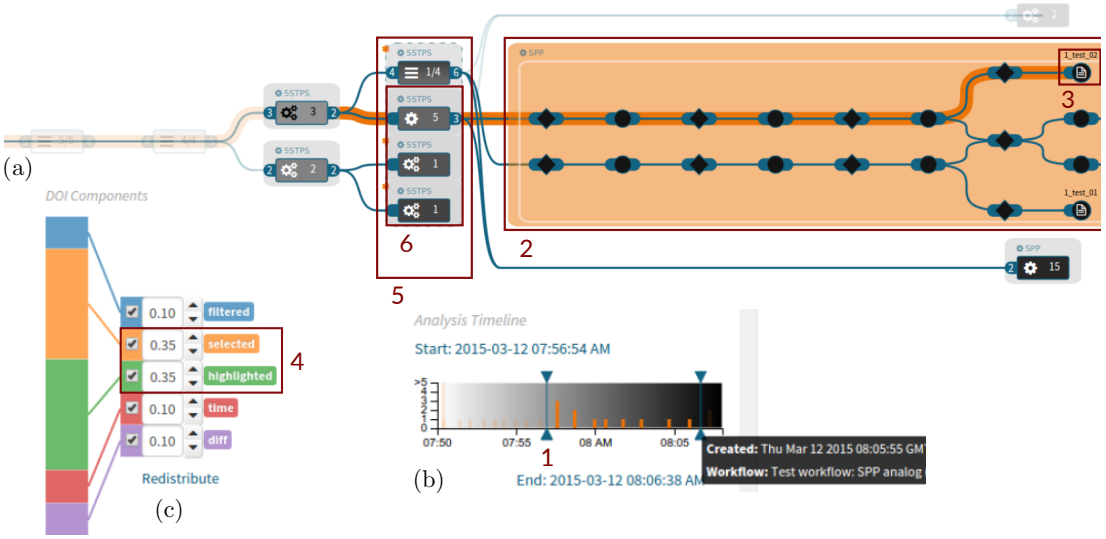


Figure 6.11: *A screenshot of the provenance graph (a) showing a correctly reproduced result as not a single analysis alongside the highlighted path indicates a change. The analysis timeline is shown in (b) and the unbalanced DOI configuration in (c).*

6.3 Limitations

We collected feedback and limitations from users, the Refinery team, and visualization experts at the *InfoVis'15* conference. Users, who had tested the visualization, were able to carry out tasks as presented in Section 6.2. They preferred use the button shortcuts to navigate in-between the hierarchy levels. Provided by a quick introduction to layering and DOI, they felt comfortable investigating changes based on layering, and in particular favored utilizing the DOI in combination with path highlighting. Based on their valuable feedback, we derived a series of limitations that we discuss next.

Time Encoding and Exploration In our approach the user explores the temporal development of the provenance graph using the analysis timeline view. It is also possible to read the temporal development directly from the node, as time is mapped to brightness. However, this method is cumbersome and inaccurate, and it might be better to pick multiple snapshots of the graph and compare them side by side. A recent approach

uses an aggregated timeline view with snapshots from which the user can pick some and show the difference in a separate view [vdEHBvW16]. In contrast, the flip-book metaphor shows the structural changes from different snapshots as slightly shifted layers in the main graph [SGL+14].

Graph Layout A crucial point to the perception of graphs is the chosen layout. We use a column-based approach as described in Section 4.1. However, the drawback is that the largest node (e.g., an expanded analysis at workflow level) defines, similar to a spreadsheet, the width for the remaining cells in the same column and therefore results in long links to adjacent column cells. Other layout approaches might create a more compact layout [YDG+16], but we had to make a compromise between aesthetics and the real-time needs that are required for fluent user interaction between different hierarchy levels.

Navigation among Hierarchy Levels The user can switch between different hierarchy levels interactively or dynamically adjust the graph using the modular DOI. For the layer and analysis level the layout is recomputed to minimize edge crossings (as discussed above). Using jump cuts between the different layouts might cause a loss of the mental map to the user [MELS95]. Instead, we use transitions of the visual space and animate the nodes' position and size. However, the change from subanalysis to the expanded workflow is discrete and intermediate transitions would improve the user experience.

Node Glyph Design Our node glyph encodes a fixed set of attributes that is limited to general information. Users might be interested to add further node attributes. However, the number of attributes that can be encoded in a glyph, without decreasing readability, is limited. To overcome this issue, we let the user choose one out of multiple attributes as well as select color schemes that distinctively encode time, node type, or workflow template in a more salient fashion.

Technical Considerations Besides the visual scalability of the technique, we should also briefly discuss the technical constraints comprising the performance of the provenance visualization. On the server side, Refinery stores the provenance graph structure in a relational database, which makes querying and updating the graph expensive and slow. A solution would be a switch to a dedicated graph database, such as *Neo4j*[1]. On the client-side the visualization handles a large number of graph nodes. Currently, the visualization generates and adds all graph nodes to the DOM tree at initialization time. Importing nodes on-demand would distribute load and result in a faster initialization of the visualization.

[1] `http://neo4j.com/`

Chapter 7

Conclusion

Reproducibility is becoming an issue of emerging importance as scientific datasets such in genomic studies increase in complexity and size. Existing provenance visualization try to address scalability but struggle to provide an interactive user experience for the exploration of time-evolving graphs at moderate scale.

In this thesis, we presented a provenance graph visualization for biomedical workflow provenance in Refinery. Facilitated through a large set of visualization techniques and the combination of multiple views, researchers are now able to interactively review and investigate genomic studies over time. The dynamic node-link approach fits both attribute and topology-based tasks well, and makes visual exploration of paths simpler. Our solution emphasizes on the perception of recurring workflows as their layout is deterministic. From a data model perspective, we incorporated the hierarchical graph structure of the provenance model in Refinery that allows us to display an initial overview of the graph. We further compress data through motif-based layering and encode essential attributes such as workflow template, change, or time in node glyphs. We also provide classic visualization techniques such as pan, (semantic) zoom, and filter to hide irrelevant information. In addition to these view-manipulation techniques, users are able to navigate over all hierarchy levels, incrementally revealing details of the underlying workflows. We provide interest-driven and guided navigation by the application of a modular DOI function. Our graph visualization is accompanied by multiple support views that are directly linked to user actions such as path highlighting, node selection or filtering.

Users are empowered by this wide variety of features, and with a little training, they are able to successfully accomplish their tasks as demonstrated in Section 6.2. Currently, the development team is transitioning Refinery to a permanent web hosting environment. Refinery is free of charge and an open source licenced project. As of our contributing effort, users will be allowed to upload their repositories and track the development of provenance graphs with our visualization in real-world scenarios.

7.1 Future Work

When users interact and navigate through the graph, they might make interesting observations. In order to memorize or share these findings, it would be beneficial to let them annotate elements of the graph and leave comments. However, the visualization does not support any form of textual annotations in its current state. In the future, we would like to enhance Refinery's provenance data model to add textual notes to any element in the graph. We imagine to capture and save states of the visualization including current visible node levels, the layout, and configurations of the support views. This enhancement would enable users to present and share their findings they made more conveniently.

Analogous to the suggested annotations above, we think of adding actionable provenance support. Users could relaunch a particular analysis or rerun a workflow on a different set of input files directly from within the visualization. In the current state, reruns are possible but cumbersome to achieve. The user would have to switch back to Refinery's dataset view, choose a workflow, and the set of input files to be executed upon.

Our implementation of the DOI function is limited to attributes and user actions. We would like to extend the function and add distance as explained by Furnas [Fur86]. The DOI of neighbor nodes decreases with distance to the closest focal point. Eventually, nodes with a high DOI will also emphasize the neighborhood, resulting in a geometric distortion of the drawing space where interest directly correlates to space. Another enhancement to the DOI function is to offer users a list of favorite DOI presets. In order to determine useful presets we want to conduct a user study and derive common usage patterns.

Finding the right balance between multiple competing aggregation strategies can become quite challenging. The implementation of motif-based layering is rather specific in its current version. The user neither has manual control over the rules nor can it be turned off. Motif discovery is limited to the analysis level and only considers a handful of properties. We suggest to enhance motif-based layering in three important aspects: (1) apply it to workflows too, (2) broaden the list of supported motif properties to attributes such as cancer type, and (3) let the user decide from which properties a motif is derived.

List of Figures

Bibliography

[ABJ+04] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludaescher, and S. Mock. Kepler: An Extensible System for Design and Execution of Scientific Workflows. In *Proceedings of the IEEE Conference on Scientific and Statistical Database Management (SSDBM '04)*, pages 423–424. IEEE, 2004.

[ABJF06] I. Altintas, O. Barney, and E. Jaeger-Frank. Provenance Collection Support in the Kepler Scientific Workflow System. In *Provenance and Annotation of Data*, volume 4145, pages 118–132. Springer, 2006.

[ABL10] M.K. Anand, S. Bowers, and B. Ludaescher. Provenance Browser: Displaying and Querying Scientific Workflow Provenance Graphs. In *Proceedings of the IEEE Conference on Data Engineering (ICDE '10)*, pages 1201–1204. IEEE, 2010.

[AHSS14] J. Abello, S. Hadlak, H. Schumann, and H.-J. Schulz. A Modular Degree-of-Interest Specification for the Visual Analysis of Large Dynamic Networks. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '13)*, 20(3):337–350, 2014.

[Arc09] D. Archambault. Structural Differences Between Two Graphs Through Hierarchies. In *Proceedings of the IEEE Conference on Graphics Interface (GI '09)*, pages 87–94. Canadian Information Processing Society, 2009.

[AS06] T. Aittokallio and B. Schwikowski. Graph-based Methods for Analysing Networks in Cell Biology. *Briefings in Bioinformatics*, 7(3):243–255, 2006.

[AvH04] J. Abello and F. van Ham. Matrix Zoom: A Visual Interface to Semi-External Graphs. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '04)*, pages 183–190. IEEE, 2004.

[AvHK06] J. Abello, F. van Ham, and N. Krishnan. ASK-GraphView: A Large Scale Graph Visualization System. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '06)*, 12(5):669–676, 2006.

[AW95] C. Ahlberg and E. Wistrand. IVEE: An Information Visualization and Exploration Environment. In *Proceedings of the IEEE Conference on Information Visualization (Vis '95)*, pages 66–73. IEEE, 1995.

[BBDW14] F. Beck, M. Burch, S. Diehl, and D. Weiskopf. The State of the Art in Visualizing Dynamic Graphs. In *Proceedings of the Eurographics Conference on Visualization (EuroVis '14) – State of The Art Reports*, 2014.

[BC87] R. A. Becker and W. S. Cleveland. Brushing Scatterplots. *Journal of Technometrics*, 29(2):127–142, 1987.

[BC03] U. Brandes and S. R. Corman. Visual Unrolling of Network Evolution and the Analysis of Dynamic Discourse. *Information Visualization*, 2(1):40–50, 2003.

[BCC+05] L. Bavoil, S.P. Callahan, P.J. Crossno, J. Freire, C.E. Scheidegger, C.T. Silva, and H.T. Vo. VisTrails: Enabling Interactive Multiple-View Visualizations. In *Proceedings of the IEEE Conference on Visualization (Vis '05)*, pages 135–142. IEEE, 2005.

[BE12] C. G. Begley and L. M. Ellis. Drug Development: Raise Standards for Preclinical Cancer Research. *Nature*, 483(7391):531–533, 2012.

[Ber83] J. Bertin. *Semiology of Graphics*. University of Wisconsin Press, Paris, 1983.

[BETT98] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall PTR, 1998.

[BI15] C. G. Begley and J. P. A. Ioannidis. Reproducibility in Science Improving the Standard for Basic and Preclinical Research. *Circulation Research*, 116(1):116–126, 2015.

[BJM04] W. Barth, M. Juenger, and P. Mutzel. Simple and Efficient Bilayer Cross Counting. *Journal of Graph Algorithms and Applications*, 8(2):179–194, 2004.

[BK02] U. Brandes and B. Koepf. Fast and Simple Horizontal Coordinate Assignment. In *Proceedings of the International Symposium on Graph Drawing (GD '01)*, volume 2265 of *Lecture Notes in Computer Science*, pages 31–44. Springer, 2002.

[BKC+01] D. Blankenberg, G. V. Kuster, N. Coraor, G. Ananda, R. Lazarus, M. Mangan, A. Nekrutenko, and J. Taylor. Galaxy: A Web-Based Genome Analysis Tool for Experimentalists. In *Current Protocols in Molecular Biology*. John Wiley & Sons, Inc., 2001.

[BLS99] A. Brandstaedt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, 1999.

[BMMS91] A. Buja, J.A. McDonald, J. Michalak, and W. Stuetzle. Interactive Data Visualization Using Focusing and Linking. In *Proceedings of the IEEE Conference on Visualization (Vis '91)*, pages 156–163. IEEE, 1991.

[BN01] T. Barlow and P. Neville. A Comparison of 2-D Visualizations of Hierarchies. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '01)*, pages 131–138. IEEE, 2001.

[BOH11] M. Bostock, V. Ogievetsky, and J. Heer. D3 Data-Driven Documents.

IEEE Transactions on Visualization and Computer Graphics (InfoVis '11), 17(12):2301–2309, 2011.

[Bra01] J. Branke. Dynamic Graph Drawing. In *Drawing Graphs: Methods and Models*, volume 2025 of *Lecture Notes in Computer Science*, pages 228–246. Springer Berlin Heidelberg, 2001.

[BSP+93] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. Toolglass and Magic Lenses: The See-through Interface. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '93)*, SIGGRAPH '93, pages 73–80. ACM, 1993.

[BWK00] M. Baldonado, A. Woodruff, and A. Kuchinsky. Guidelines for Using Multiple Views in Information Visualization. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 110–119. ACM, 2000.

[BYB+13] M. Borkin, C. Yeh, M. Boyd, P. Macko, K. Gajos, M. Seltzer, and H. Pfister. Evaluation of Filesystem Provenance Visualization Tools. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '13)*, 2013.

[Car03] M. S. T. Carpendale. Considering Visual Variables as a Basis for Information Visualisation. Technical Report, Department of Computer Science, University of Calgary, 2003.

[Car12] S. Card. Information Visualization. In *The Human-Computer Interaction Handbook*, pages 544–582. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 2012.

[CC05] B. Craft and P. Cairns. Beyond Guidelines: What Can We learn from the Visual Information Seeking Mantra? In *Proceedings of the International Conference on Information Visualization (IV '05)*, pages 110–118. IEEE, 2005.

[CEH+09] M. Chen, D. Ebert, H. Hagen, R.S. Laramee, R. van Liere, K.-L. Ma, W. Ribarsky, G. Scheuermann, and D. Silver. Data, Information, and Knowledge in Visualization. *IEEE Computer Graphics and Applications*, 29(1):12–19, 2009.

[CFS+06] S.P. Callahan, J. Freire, E. Santos, C.E. Scheidegger, C.T. Silva, and H.T. Vo. Managing the Evolution of Dataflows with VisTrails. In *Proceedings of the IEEE Workshop on Workflow and Data Flow for Scientific Applications (SciFlow '06)*, page 71. IEEE, 2006.

[CG72] E. G. Jr. Coffman and R. L. Graham. Optimal Scheduling for Two-Processor Systems. *Acta Informatica*, 1(3):200–213, 1972.

[Cha77] J. M. Chambers. *Computational Methods for Data Analysis*. Wiley, 1977.

[Che73] H. Chernoff. The Use of Faces to Represent Points in K-Dimensional Space Graphically. *Journal of the American Statistical Association*, 68(342):361–

368, 1973.

[Cim06] R. Cimikowski. An Analysis of Some Linear Graph Layout Heuristics. *Journal of Heuristics*, 12(3):143–153, 2006.

[CKB08] A. Cockburn, A. Karlson, and B. B. Bederson. A Review of Overview+Detail, Zooming, and Focus+Context Interfaces. *ACM Computing Surveys*, 41(1):2, 2008.

[CM97] S.K. Card and J. Mackinlay. The Structure of the Information Visualization Design Space. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '97)*, pages 92–99. IEEE, 1997.

[CMS99] S. K. Card, J. D. Mackinlay, and B. Shneiderman, editors. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers Inc., 1999.

[CPC+12] P. Chen, B. Plale, Y.-W. Cheah, D. Ghoshal, S. Jensen, and Y Luo. Visualization of Network Data Provenance. In *Proceedings of the IEEE Conference on High Performance Computing*, pages 1–9. IEEE, 2012.

[DCBE+07] S. Davidson, S. Cohen-Boulakia, A. Eyal, B. Ludaescher, T. McPhillips, S. Bowers, and J. Freire. Provenance in Scientific Workflow Systems. *IEEE Data Engineering Bulletin*, 30(4):44–50, 2007.

[Deo74] N. Deo. *Graph Theory with Applications to Engineering and Computer Science*. Prentice-Hall, Inc., 1974.

[DF08] S. B. Davidson and J. Freire. Provenance and Scientific Workflows: Challenges and Opportunities. In *Proceedings of the ACM SIGMOD Conference on Management of Data (SIGMOD '08)*, pages 1345–1350. ACM, 2008.

[DK08] J. Dokulil and J. Katreniakova. Edge Routing with Fixed Node Positions. In *Proceedings of the International Conference on Information Visualisation (IV '08)*, pages 626–631. IEEE, 2008.

[DPS02] J. Díaz, J. Petit, and M. Serna. A Survey of Graph Layout Problems. *ACM Computing Surveys*, 34(3):313–356, 2002.

[Ead84] P. Eades. A Heuristics for Graph Drawing. *Congressus Numerantium*, 42:146–160, 1984.

[EDG+08] N. Elmqvist, Thanh-Nghi Do, H. Goodell, N. Henry, and J. Fekete. ZAME: Interactive Large-Scale Graph Visualization. In *Proceedings of the IEEE Pacific Visualization Symposium (PacificVis '08)*, pages 215–222, 2008.

[EF10] N. Elmqvist and J.-D. Fekete. Hierarchical Aggregation for Information Visualization: Overview, Techniques, and Design Guidelines. *IEEE Transactions on Visualization and Computer Graphics*, 16(3):439–454, 2010.

[EKLN04] C. Erten, S. G. Kobourov, V. Le, and A. Navabi. Simultaneous Graph Drawing: Layout Algorithms and Visualization Schemes. In *Proceedings of*

the International Symposium on Graph Drawing (GD '03), pages 437–449. Springer Berlin Heidelberg, 2004.

[ELMS91] P. Eades, W. Lai, K. Misue, and K. Sugiyama. *Preserving The Mental Map of a Diagram.* International Institute for Advanced Study of Social Information Science, Fujitsu Limited, 1991.

[FB95] G. W. Furnas and B. B. Bederson. Space-scale Diagrams: Understanding Multiscale Interfaces. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI '95)*, pages 234–241. ACM Press/Addison-Wesley Publishing Co., 1995.

[Fie00] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* PhD Dissertation, University of California, Irvine, 2000. AAI9980887.

[FP02] J. Fekete and C. Plaisant. Interactive Information Visualization of a Million Items. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '02)*, pages 117–124. IEEE, 2002.

[FR91] T. M. J. Fruchterman and E. M. Reingold. Graph Drawing by Force-directed Placement. *Software, Practice and Experience*, 21(11):1129–1164, 1991.

[FS12] J. Freire and C. T. Silva. Making Computations and Publications Reproducible with VisTrails. *Computing in Science & Engineering*, 14(4):18–25, 2012.

[FSC+06] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo. Managing Rapidly-Evolving Scientific Workflows. In *Proceedings of the International Conference on Provenance and Annotation of Data (IPAW '06)*, Lecture Notes in Computer Science, pages 10–18. Springer Berlin Heidelberg, 2006.

[Fur86] G. W. Furnas. Generalized Fisheye Views. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '86)*, pages 16–23. ACM, 1986.

[GEY12] S. Ghani, N. Elmqvist, and J. S. Yi. Perception of Animated Node-Link Diagrams for Dynamic Graphs. *Computer Graphics Forum*, 31(3pt3):1205–1214, 2012.

[GFC04] M. Ghoniem, J.-D. Fekete, and P. Castagliola. A Comparison of the Readability of Graphs Using Node-Link and Matrix-Based Representations. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '04)*, pages 17–24. IEEE, 2004.

[GK06] E. R. Gansner and Y. Koren. Improved Circular Layouts. In *Graph Drawing*, number 4372 in Lecture Notes in Computer Science, pages 386–398. Springer Berlin Heidelberg, 2006.

[GKNV93] E.R. Gansner, E. Koutsofios, S.C. North, and K.-P. Vo. A Technique for Drawing Directed Graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, 1993.

[GNT10] J. Goecks, A. Nekrutenko, and J. Taylor. Galaxy: A Comprehensive Approach for Supporting Accessible, Reproducible, and Transparent Computational Research in the Life Sciences. *Genome Biology*, 11(8):R86, 2010.

[GRH+05] B. Giardine, C. Riemer, R. C. Hardison, R. Burhans, L. Elnitski, P. Shah, Y. Zhang, D. Blankenberg, I. Albert, J. Taylor, W. Miller, W. J. Kent, and A. Nekrutenko. Galaxy: A Platform for Interactive Large-scale Genome Analysis. *Genome Research*, 15(10):1451–1455, 2005.

[GS00] A. Gilliland-Swetland. *Introduction to Metadata: Pathways to Digital Information*. Getty Information Institute, 2000.

[HBO10] J. Heer, M. Bostock, and V. Ogievetsky. A Tour Through the Visualization Zoo. *Communications of the ACM*, 53(6):59, 2010.

[HC04] J. Heer and S. K. Card. DOITrees Revisited: Scalable, Space-Constrained Visualization of Hierarchical Data. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 421–424. ACM, 2004.

[HC07] J. Hunter and K. Cheung. Provenance Explorer-A Graphical Interface for Constructing Scientific Publication Packages from Provenance Trails. *International Journal on Digital Libraries*, 7(1-2):99–107, 2007.

[HFM07] N. Henry, J. Fekete, and M.J. McGuffin. NodeTrix: a Hybrid Visualization of Social Networks. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1302–1309, 2007.

[Hol06] D. Holten. Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, 2006.

[HSS15] S. Hadlak, H. Schumann, and H.-J. Schulz. A Survey of Multi-faceted Graph Visualization. In *Proceedings of the Eurographics Conference on Visualization (EuroVis '15) – State of The Art Reports*. The Eurographics Association, 2015.

[HVW09] D. Holten and J. J. Van Wijk. Force-Directed Edge Bundling for Graph Visualization. *Computer Graphics Forum*, 28(3):983–990, 2009.

[ID90] A. Inselberg and B. Dimsdale. Parallel Coordinates: A Tool for Visualizing Multi-dimensional Geometry. In *Proceedings of the IEEE Conference on Visualization (Vis '90)*, pages 361–378. IEEE, 1990.

[JM97] M. Juenger and P. Mutzel. 2-Layer Straightline Crossing Minimization: Performance of Exact and Heuristic Algorithms. *Journal of Graph Algorithms and Application*, 1(1):1–25, 1997.

[JS91] B. Johnson and B. Shneiderman. Tree-maps: A Space-filling Approach to the Visualization of Hierarchical Information Structures. In *Proceedings of the IEEE Conference on Visualization (Vis '91)*, pages 284–291. IEEE, 1991.

[Kai15] J. Kaiser. The Cancer Test. *Science*, 348(6242):1411–1413, 2015.

[Kei00] D. A. Keim. Designing Pixel-Oriented Visualization Techniques: Theory and Applications. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):59–78, 2000.

[KHG03] R. Kosara, H. Hauser, and D. L. Gresh. An Interaction View on Information Visualization. In *Proceedings of EUROGRAPHICS 2003 (EG '03) – State of The Art Reports*, pages 123–137. Eurographics Association, 2003.

[KKC14] N. Kerracher, J. Kennedy, and K. Chalmers. The Design Space of Temporal Graph Visualisation. In *Proceedings of the Eurographics Conference on Visualization (EuroVis '14) – Short Papers Track*, 2014.

[KKEM10] D. Keim, J. Kohlhammer, G. Ellis, and F. Mansmann, editors. *Mastering the Information Age - Solving Problems with Visual Analytics*. Eurographics Association, 2010.

[LAB+06] B. Ludaescher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and the Kepler System: Research Articles. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[LPP+06] B. Lee, C. Plaisant, C. S. Parr, J.-D. Fekete, and N. Henry. Task Taxonomy for Graph Visualization. In *Proceedings of the AVI Workshop on BEyond time and errors: novel evaluation methods for information visualization (BELIV '06)*, pages 1–5. ACM, 2006.

[LSG+15a] S. Luger, H. Stitz, S. Gratzl, N. Gehlenborg, and M. Streit. Interactive Visualization of Provenance Graphs for Reproducible Biomedical Research. In *Poster Compendium of the IEEE VIS Conference (InfoVis '15)*. IEEE, 2015.

[LSG+15b] S. Luger, H. Stitz, S. Gratzl, M. Streit, and N. Gehlenborg. Interactive Visualization of Provenance Graphs for Reproducible Biomedical Research. In *Poster Proceedings of the Symposium on Biological Data Visualization (BioVis '15)*. ISMB, 2015.

[LWW90] J. LeBlanc, M. O. Ward, and N. Wittels. Exploring n-Dimensional Databases. In *Proceedings of the IEEE Conference on Visualization (Vis '90)*, pages 230–237, 1990.

[MELS95] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout Adjustment and the Mental Map. *Journal of Visual Languages and Computing*, 6(2):183–210, 1995.

[Mes10] J. P. Mesirov. Computer Science. Accessible Reproducible Research. *Science*, 327(5964):415–416, 2010.

[MRSS+12] E. Maguire, P. Rocca-Serra, S.-A. Sansone, J. Davies, and M. Chen. Taxonomy-Based Glyph Design with a Case Study on Visualizing Workflows of Biological Experiments. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '12)*, 18(12):2603–2612, 2012.

[MRSS+13] E. Maguire, P. Rocca-Serra, S.-A. Sansone, J. Davies, and M. Chen. Visual Compression of Workflow Visualizations with Automated Detection of Macro Motifs. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2576–2585, 2013.

[MSOI+02] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network Motifs: Simple Building Blocks of Complex Networks. *Science*, 298(5594):824–827, 2002.

[Mun08] T. Munzner. Process and Pitfalls in Writing Information Visualization Research Papers. In *Information Visualization*, pages 134–153. Springer Berlin Heidelberg, 2008.

[Mun14] T. Munzner. *Visualization Analysis and Design*. CRC Press, 2014.

[MW95] A. R. Martin and M. O. Ward. High Dimensional Brushing for Interactive Exploration of Multivariate Data. In *Proceedings of the IEEE Conference on Visualization (Vis '95)*, pages 271–. IEEE, 1995.

[NJ04] S. Noel and S. Jajodia. Managing Attack Graph Complexity Through Visual Hierarchical Aggregation. In *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, pages 109–118. ACM, 2004.

[OEY+13] L. Omberg, K. Ellrott, Y. Yuan, C. Kandoth, C. Wong, M. R. Kellen, S. H. Friend, J. Stuart, H. Liang, and A. A. Margolin. Enabling Transparent and Collaborative Computational Analysis of 12 Tumor Types within The Cancer Genome Atlas. *Nature Genetics*, 45(10):1121–1126, 2013.

[OL03] M.C.F. Oliveira and H. Levkowitz. From Visual Data Exploration to Visual Data Mining: A Survey. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):378–394, 2003.

[pet15] dagre - Graph Layout for JavaScript, 2015. Accessed: 2015-10-02.

[PHG+13] R. Paula, M. Holanda, L. Gomes, S. Lifschitz, and M. E. Walter. Provenance in Bioinformatics Workflows. *BMC Bioinformatics*, 14(Suppl 11):S6, 2013.

[PSCO09] W. A. Pike, J. Stasko, R. Chang, and T. A. O'Connell. The Science of Interaction. *Information Visualization*, 8(4):263–274, 2009.

[ref15] Refinery Platform, 2015. Accessed: 2015-10-01.

[RESC15] E. Ragan, A. Endert, J. Sanyal, and J. Chen. Characterizing Provenance in Visualization and Data Analysis: An Organizational Framework of Provenance Types and Purposes. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '15)*, 22(1):31–40, 2015.

[RHR+10] M. Rohrschneider, C. Heine, A. Reichenbach, A. Kerren, and G. Scheuermann. A Novel Grid-Based Visualization Approach for Metabolic Networks with Advanced Focus&Context View. In *Proceedings of the International Symposium on Graph Drawing (GD '09)*, Lecture Notes in Computer Science, pages 268–279. Springer Berlin Heidelberg, 2010.

[RLG+06] M. Reich, T. Liefeld, J. Gould, J. Lerner, P. Tamayo, and J. P. Mesirov. GenePattern 2.0. *Nature Genetics*, 38(5):500–501, 2006.

[RMF12] S. Rufiange, M. McGuffin, and C. Fuhrman. TreeMatrix: A Hybrid Visualization of Compound Graphs. *Computer Graphics Forum*, 31(1):89–101, 2012.

[RS07] N. D. Rio and P. P. da Silva. Probe-It! Visualization Support for Provenance. In *Advances in Visual Computing*, number 4842 in Lecture Notes in Computer Science, pages 732–741. Springer Berlin Heidelberg, 2007.

[RSBM+10] P. Rocca-Serra, M. Brandizi, E. Maguire, N. Sklyar, C. Taylor, K. Begley, D. Field, S. Harris, W. Hide, O. Hofmann, S. Neumann, P. Sterk, W. Tong, and S.-A. Sansone. ISA Software Suite: Supporting Standards-compliant Experimental Annotation and Enabling Curation at the Community Level. *Bioinformatics*, 26(18):2354–2356, 2010.

[SA06] B. Shneiderman and A. Aris. Network Visualization by Semantic Substrates. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):733–740, 2006.

[SB03] B. Shneiderman and Benjamin B. Bederson. *The Craft of Information Visualization: Readings and Reflections*. Morgan Kaufmann Publishers Inc., 2003.

[SB13] M. Streit and O. Bimber. Visual Analytics: Seeking the Unknown. *Computer*, 46(7):20–21, 2013.

[Sch10] H.-J. Schulz. *Explorative Graph Visualization*. PhD Dissertation, University of Rostock, 2010.

[SFC07] C. T. Silva, J. Freire, and S. P. Callahan. Provenance for Visualizations: Reproducibility and Beyond. *Computing in Science & Engineering*, 9(5):82–89, 2007.

[SGL+14] H. Stitz, S. Gratzl, S. Luger, N. Gehlenborg, and M. Streit. Transparent Layering for Visualizing Dynamic Graphs Using the Flip Book Metaphor. In *Poster Compendium of the IEEE VIS Conference (InfoVis '14)*. IEEE, 2014.

[Shn96] B. Shneiderman. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *Proceedings of the IEEE Symposium on Visual Languages (VL '96)*, pages 336–343, 1996.

[SKS+08] C. Scheidegger, D. Koop, E. Santos, H. Vo, S. Callahan, J. Freire, and C. Silva. Tackling the Provenance Challenge One Layer at a Time. *Concurrency and Computation: Practice and Experience*, 20(5):473–483, 2008.

[SM11] M. I. Seltzer and P. Macko. Provenance Map Orbiter: Interactive Exploration of Large Provenance Graphs. In *Proceedings of the USENIX Workshop on the Theory and Practice of Provenance (TaPP '11)*, 2011.

[SMM12] M. Sedlmair, M. Meyer, and T. Munzner. Design Study Methodology: Reflections from the Trenches and the Stacks. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '12)*, 18(12):2431–2440, 2012.

[SNTH13] G. K. Sandve, A. Nekrutenko, J. Taylor, and E. Hovig. Ten Simple Rules for Reproducible Computational Research. *PLoS Computational Biology*, 9(10):e1003285, 2013.

[Spe07] R. Spence. *Information Visualization: Design for Interaction.* Prentice-Hall, Inc., 2007.

[SPG05] Y. L. Simmhan, B. Plale, and D. Gannon. A Survey of Data Provenance in e-Science. *ACM SIGMOD Record*, 34(3):31–36, 2005.

[SS06] H.-J. Schulz and H. Schumann. Visualizing Graphs - A Generalized View. In *Proceedings of the IEEE Conference on Information Visualisation (IV '06)*, pages 166–173. IEEE, 2006.

[STT81] K. Sugiyama, S. Tagawa, and M. Toda. Methods for Visual Understanding of Hierarchical System Structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109–125, 1981.

[SYH+13] Z. Sun, P. Yue, L. Hu, J. Gong, L. Zhang, and X. Lu. GeoPWProv: Interleaving Map and Faceted Metadata for Provenance Visualization and Navigation. *IEEE Transactions on Geoscience and Remote Sensing*, 51(11):5131–5136, 2013.

[SZ00] J. Stasko and E. Zhang. Focus+ Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualizations. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '00)*, pages 57–65. IEEE, 2000.

[TAS09] C. Tominski, J. Abello, and H. Schumann. CGV—An Interactive Graph Visualization System. *Journal of Computers & Graphics*, 33(6):660–678, 2009.

[TC05] J. J. Thomas and K. A. Cook, editors. *Illuminating the Path: The Research and Development Agenda for Visual Analytics.* IEEE, 2005.

[vdEHBvW16] S. van den Elzen, D. Holten, J. Blaas, and J. J. van Wijk. Reducing Snapshots to Points: A Visual Analytics Approach to Dynamic Network Exploration. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):1–10, 2016.

[vHP09] F. van Ham and A. Perer. "Search, Show Context, Expand on Demand": Supporting Large Graph Exploration with Degree-of-Interest. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '09)*, 15(6):953–960, 2009.

[vHSD09] F. van Ham, H.-J. Schulz, and J. M. Dimicco. Honeycomb: Visual Analysis of Large Scale Social Networks. In *Human-Computer Interaction – INTERACT 2009*, number 5727 in Lecture Notes in Computer Science, pages 429–442. Springer Berlin Heidelberg, 2009.

[vHvW04] F. van Ham and J.J. van Wijk. Interactive Visualization of Small World Graphs. In *Proceedings of the IEEE Symposium on Information Visualization (Infovis '04)*, pages 199–206. IEEE, 2004.

[Wat02] M. Wattenberg. Arc Diagrams: Visualizing Structure in Strings. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '02)*, pages 110–116. IEEE, 2002.

[WGK10] M. Ward, G. Grinstein, and D. Keim. *Interactive Data Visualization: Foundations, Techniques, and Applications*. A. K. Peters, Ltd., 2010.

[WHF+13] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, and C. Goble. The Taverna Workflow Suite: Designing and Executing Workflows of Web Services on the Desktop, Web or in the Cloud. *Nucleic Acids Research*, 41(W1):W557–W561, 2013.

[WMK06] D. Weiskopf, K.-L. Ma, and R. Kosara. SciVis, InfoVis - Bridging the Community Divide? In *Proceedings of the IEEE Conference on Visualization (Vis '06)*. IEEE, 2006.

[YDG+16] V. Yoghourdjian, T. Dwyer, G. Gange, S. Kieffer, K. Klein, and K. Marriott. High-Quality Ultra-Compact Grid Layout of Grouped Networks. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '15)*, 22(1):339–348, 2016.

[ZMC05] S. Zhao, M. J. McGuffin, and M. H. Chignell. Elastic Hierarchies: Combining Treemaps and Node-Link Diagrams. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '05)*, pages 57–64. IEEE, 2005.

Stefan Luger, BSc

Curriculum Vitae

Höhenweg 2
4181 Oberneukirchen
Austria
+43 (650) 551 3391
lug.stefan@gmail.com
sluger.github.io

Education

10/2013–03/2016 **Diplom-Ingenieur, Computer Science**, *Johannes Kepler University (JKU)*, Linz, Austria.

Master's Thesis

Title *Interactive Visualization of Provenance Graphs for Reproducible Biomedical Research*

Supervisors Marc Streit at JKU and Nils Gehlenborg at **Harvard, Boston, MA, USA.**

Summary In cooperation with the Harvard Medical School, I developed a novel graph visualization for the Refinery Platform (*refinery-platform.org*). Skills: Data Visualization, Graphs, Full Stack Web Development, Linux, Scrum, Git

Project in Pervasive Computing

Workflow Visualization for The Refinery Platform. Web-based visualization and front end design at JKU and **Harvard**. Skills: JavaScript, HTML, CSS, D3.js, SVG, JSON, Linux, Scrum, Git

10/2010–09/2013 **BSc, Informatics**, *JKU*.

Bachelor's Thesis

Constraint Generation and Partial Fixing for UML Models through Transformation. Skills: Java, XML, UML, ATL, OCL, VCS

10/2005–01/2009 **Bioinformatics**, *University of Applied Sciences Upper Austria*, Hagenberg.

Visualization Project

RNAVis. In cooperation with the Medical University of Vienna, I created a visualization framework for RNA data sets. Skills: Visualization, Java, XML, Prefuse, VCS

10/1999–10/2004 **Computer Science**, *Höhere Technische Bundeslehranstalt*, Neufelden, Austria.

Experience

08/2013–07/2015 **IT System Administrator**, *JKU*.

Server, soft-, and hardware infrastructure setup and maintenance. Setup and hosting of video conferences. Project assistant in the Lightfield displays domain. Skills: Linux, Hardware, 3D printing, AutoCAD, 3ds Max, Scripting.

03/2014–06/2014 **Teaching Assistant**, *JKU*.

Guiding students during their hands-on lab exercises on computer graphics. Skills: C++, OpenGL, GLSL, Cuda, Visualization

03/2013–06/2013 **Teaching Assistant**, *JKU*.

09/2011–05/2012 **IT System Administrator**, *TMS Transport und Montagesysteme GmbH*, Linz.
IT management responsibilities. Skills: SCCM, SQL, Scripting, Office, Windows

04/2008–08/2009 **Software Developer**, *Krankenhaus der Barmherzigen Schwestern*, Linz.
Bioinformatics software interface for Microarray analysis. Skills: Java, SWT

07/2007–09/2007 **Web Developer**, *BlueGnome Ltd.*, **Cambridge, UK**.
Extending a Laboratory Information Management System. Skills: CGI, Linux

Languages

German **Native language**

English **Full professional proficiency**

Computer Skills

Web Full Stack Web Development, JavaScript, HTML, CSS, jQuery, Django, AngularJS, REST, Node.js, GWT, Virtual Environments

Data Information Visualization, D3.js, SVG, Graphs, Exploratory Data Analysis, Machine Learning, Python, R, Prolog, SQL, NoSQL

Software Eng. Scrum, Requirements, Architectures, Design Patterns, Model-driven Engineering, Software Quality, Testing, Team-oriented, Git

Programming Java, C, C++, C#, Haskell, Perl

Miscellaneous Linux, Office, LaTeX, Bioinformatics, Virtual Reality, IT, Hardware, Medic

Interests

Technology, Hardware, Linux, Astronomy, Volleyball, Music, Movies

Honors & Awards

Best Poster Award S. Luger, H. Stitz, S. Gratzl, N. Gehlenborg, M. Streit. *Interactive Visualization of Provenance Graphs for Reproducible Biomedical Research.* In Poster Compendium of the IEEE VIS Conference (InfoVis'15). Chicago, IL, USA. IEEE, 2015. **Presented at InfoVis'15, Chicago, IL, USA in October 2015.**

Study Grant For my Master's Thesis and the visit to IEEE VIS Conference (InfoVis '15), I have received a Study Grant from JKU Linz.

References

Marc Streit
Institute of Computer Graphics
Johannes Kepler University Linz
4040 Linz, Austria
✉ *marc.streit@jku.at*
☎ *+43 (732) 2468 6635*

Nils Gehlenborg
Dept. of Biomedical Informatics
Harvard Medical School
Boston, MA 02115
✉ *nils@gehlenborg.com*
☎ *+1 (617) 432 1535*

Sworn declaration

I hereby declare under oath that the submitted Master's Thesis has been written solely by me without any third-party assistance, information other than provided sources or aids have not been used and those used have been fully documented. Sources for literal, paraphrased and cited quotes have been accurately credited. The submitted document here present is identical to the electronically submitted text document.

Linz, February 2016

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe. Die vorliegende Masterarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, Februar 2016