PROPAGATION OF CHANGE AND VISUALIZATION OF CAUSALITY IN DEPENDENCY STRUCTURES

by

Saba Alimadadi Jani
B.Sc., University of Tehran, 2009

A Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science

in the
School of Interactive Arts and Technology
Faculty of Communication, Art and Technology

© Saba Alimadadi Jani 2013
SIMON FRASER UNIVERSITY
Spring 2013

All rights reserved.
However, in accordance with the *Copyright Act of Canada*, this work may be reproduced without authorization under the conditions for "Fair Dealing." Therefore, limited reproduction of this work for the purposes of private study, research, criticism, review and news reporting is likely to be in accordance with the law, particularly if cited appropriately.

APPROVAL

Name: Saba Alimadadi Jani

Degree: Master of Science

Title of Thesis: Propagation of Change and Visualization of Causality In Dependency Structures

Examining Committee:

Dr. Philippe Pasquier,
Assistant Professor, School of Interactive Arts and Technology, Simon Fraser University
Chair

Dr. Chris Shaw,
Associate Professor, School of Interactive Arts and Technology, Simon Fraser University
Senior Supervisor

Dr. Robert Woodbury,
Professor, School of Interactive Arts and Technology, Simon Fraser University
Supervisor

Dr. Halil Erhan,
Assistant Professor, School of Interactive Arts and Technology, Simon Fraser University
External Examiner

Date Approved: _____

Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the "Institutional Repository" link of the SFU Library website (www.lib.sfu.ca) at http://summit/sfu.ca and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author's written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

While licensing SFU to permit the above uses, the author retains copyright in the thesis, project or extended essays, including the right to change the work for subsequent purposes, including editing and publishing the work in whole or in part, and licensing other parties, as the author may desire.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library
Burnaby, British Columbia, Canada

revised Fall 2011

Ethics Statement

The author, whose name appears on the title page of this work, has obtained, for the research described in this work, either:

a. human research ethics approval from the Simon Fraser University Office of Research Ethics,

or

b. advance approval of the animal care protocol from the University Animal Care Committee of Simon Fraser University;

or has conducted the research

c. as a co-investigator, collaborator or research assistant in a research project approved in advance,

or

d. as a member of a course approved in advance for minimal risk human research, by the Office of Research Ethics.

A copy of the approval letter has been filed at the Theses Office of the University Library at the time of submission of this thesis or project.

The original application for approval and letter of approval are filed with the relevant offices. Inquiries may be directed to those authorities.

Simon Fraser University Library
Burnaby, British Columbia, Canada

update Spring 2010

Abstract

Analysts use visual analytics tools to gain insight through data. But one problem of these tools is the complexity of the analytics process itself. Sometimes the analyst has to repeat the same task during the analysis; to rerun some previous queries or to update the state of the system by the arrival of new data. CZSaw is a visual analytics tool that stores the process of analysis and visualizes it. The dependency graph of CZSaw allows parts of the analytical process to be reused on new data to prevent repetition of those parts. This document describes the design and development of CZSaws dependency graph as well as its visualization. A major challenge is visualization of causality in the graph when an update gets propagated. We compare three different causality visualization methods in terms of task completion duration and accuracy in a study.

Keywords: *Visual Analytics, Information Visualization, Causality Visualization, Dependency Graph, Parametric Modeling*

To My Parents

Acknowledgments

First and foremost, I wish to express my utmost gratitude to my senior supervisor, Dr. Chris Shaw. I am indebted to him for his continuous support, insight and encouragement throughout the course of this project.

I owe my deepest gratitude to my supervisor, Dr. Robert Woodbury, for his inspiration and invaluable comments.

I consider it an honor to work with Dr. Halil Erhan, my committee member, and am most grateful to him.

It gives me great pleasure in acknowledging the support and help of all my dear friends.

I wish to thank my beloved brothers, Siavash and Soroush. I am sincerely grateful to you for your constant love and support.

I cannot find words to express my gratitude to my amazing parents, Parviz and Parvin. I am eternally grateful to you for your unconditional and endless love, guidance and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Visual Analytics

Thomas and Cook described visual analytics (VA) as *"the science of analytical reasoning facilitated by interactive visual interfaces. People use visual analytics tools and techniques to synthesize information and derive insight from massive, dynamic, ambiguous, and often conflicting data; detect the expected and discover the unexpected; provide timely, defensible, and understandable assessments; and communicate assessment effectively for action."* [79]

Analysts have some problems using current visual analytics tools. One of the major problems is the complexity of the analytics process itself. This complexity can arise of massive amounts of data, or long durations of time needed for performing the analysis. Particularly, one of the problems that analysts encounter during analysis is repeating the same task, especially when they are exploring the data. Or sometimes the analysts want to rerun some previous queries, or they just want to update the current state of the system by arriving new data, instead of repeating the whole process again.
Moreover, analysts are in need of ways to represent the analytical problem better. Designing systems that make the task of analysts easier can greatly improve their performance in the analysis process.

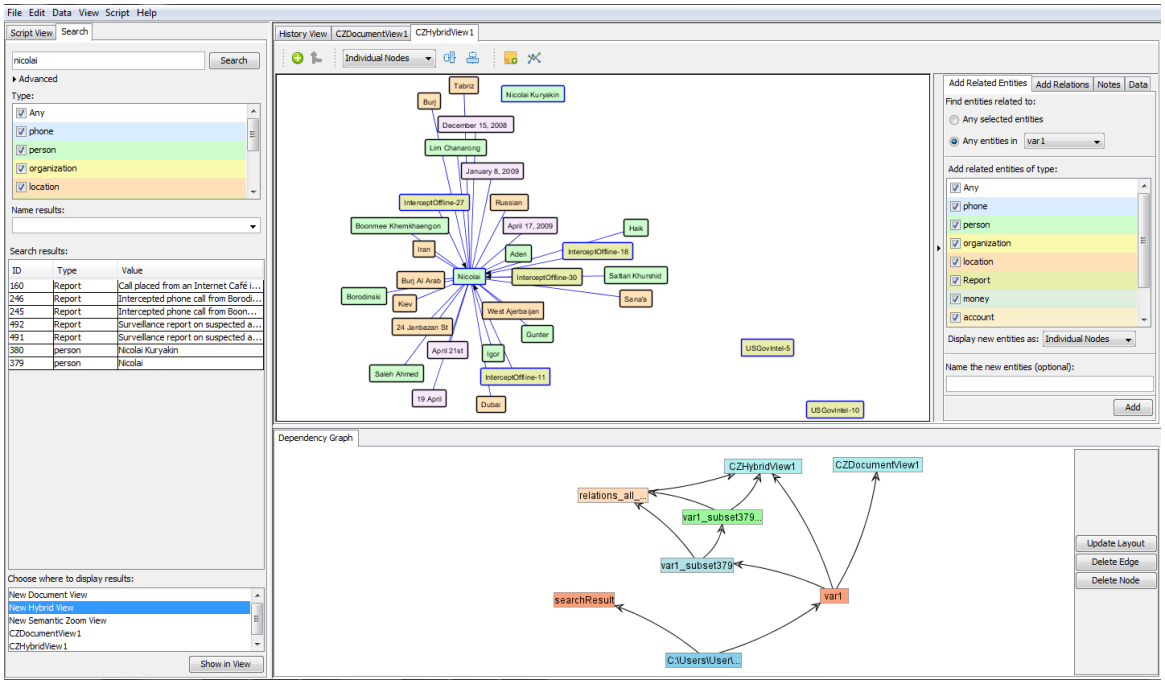


Figure 1.1: A screenshot of CZSaw tool

1.2 CZSaw

CZSaw was designed to address the aforementioned drawbacks of the existing visual analytics tool. According to Kadivar et al. [44], CZSaw, is a visual analytics tool that stores the analysis process and visualizes it using multiple views. So, when the analysts look through the history of their analysis, they can find repetitive subtasks and they can replay the steps they want on different data. CzSaw also has various views for visually representing data, inspired by Jigsaw [76], that help analysts to comprehend and investigate the data. A screenshot of CZSaw's environment is shown in figure 1.1.

All of the interactions of user with the system are stored by CzSaw's scripting language as transactions, which provide the ability to explicitly program the process. Steps of the analysis done so far can be replayed by re-running the related parts of the script. Also, CZSaw builds a dependency graph to help analysts gain a better understanding of the analytics process. Nodes of this graph are variables in the system, which are either defined by user, or defined by system for showing user variables. Links between nodes are the

dependency relations between the variables related to those nodes. Finally, CzSaw provides a visual history of user interactions, so that the users can recall the sequence of their interactions with the system, and in our case, they can find the repetitive patterns of their work by looking through the history, and replay them on new data.

1.3 Dependency Graph

The core of this project is a new dependency graph for CZSaw. This propagation-based structure records dependency function, and stores and displays the latest state of the system. In this directed acyclic graph, nodes are the variables in the system and the directed edges show the dependency relations between nodes. This graph always keeps its structure and content updated, meaning that if a change happens to the content or the structure of the graph, it automatically gets propagated to all directly and indirectly dependent nodes that may get affected. During this document, we describe how the dependency graph was designed and implemented and was integrated into CZSaw. Moreover, we discuss the new view that was designed for dynamically visualizing the dependency graph through the analysis process in CZSaw. We propose a second version of the dependency graph that optimizes the algorithm and improves the code structure.

1.4 Causality Visualization

The most important characteristic of the dependency graph is its ability to automatically and dynamically propagate the change in the graph. The graph contains a number of causal relationships. Meaning that a change in a node may cause an effect in another node. The users should be able to detect these cause-and-effect relations, and to have awareness of the changes that happen to the graph based on these relations. One analyst action could trigger other actions on other nodes based on dependency relations. Analysts should be able to notice the consequences of their actions on the system, based on the dependency graph that models the system.

The concept of cause and effect is used by people on an every-day basis. Visualizing the cause-and-effect relations, or causality visualization, helps the viewer to identify those relations, and gain a better understanding of the underlying system. A big challenge in this area is visualizing causality in context, which means a new dimension of causality gets added

to an existing visualization without changing the underlying structure. This is the desired case for CZSaw's dependency graph, where the graph already contains the data about the system variables and their relations, and that data should not be altered. During this project, we proposed two new methods for visualizing causality through animation. One of those methods is based on color and size of the nodes and was used in CZSaw's dependency graph. The other proposed method is motion-based. We discuss the study we conducted to investigate these new methods and compare them with a well-known technique.

1.5 Structure of the Document

The literature of research in this area is reviewed in Chapter 2. This project lies at the intersection of different fields of research such as visual analytics, graph drawing, causality visualization and parametric modeling. Hence, the literature review covers the related work in each of these fields. Chapter 3 is dedicated to describing the design and the development of CZSaw's dependency graph. The design and implementation of the visualization and the second version of the dependency graph are also described in this chapter. Chapter 4 presents the user study conducted on causality visualization feature of the dependency graph. The design and implementation of the prototype used in the study is also included in this chapter. Finally, Chapter 5 discusses the results of the study along with their implications.

Chapter 2

Literature Review

As mentioned in Chapter 1 the literature around this work can be reviewed from different perspective. The background of visual analytics and the CZSaw visual analytics tool is crucial to knowing the context in which this dependency graph and its visualization were built. And since this dependency structure is a graph that needs to be visualized, the background work on graph drawing from the aspect of information visualization became part of this review. The most important feature of the dependency graph is the propagation of change. Hence, for gaining knowledge on the background logic of the structure, the recent works on parametric modeling were useful. For visualizing the propagation of update, the causality visualization literature was reviewed. This Chapter provides a summary of the relevant work in each of the aforementioned fields.

2.1 Visual Analytics and CZSaw

Visual analytics tools are used to help analysts gain insight from data [79]. One problem of conventional visual analytics tools is the complexity of the analytics process itself. Sometimes the analyst has to repeat the same task during the analysis; he may want to rerun some previous queries or update the state by deriving new data.

Trying to solve this problem, CZSaw visual analytics (VA) tool was introduced by Kadivar et al. [44]. CZSaw, is *"a visual analytics tool that captures and visualizes both the analysis process, and the history of user interactions"*. CZSaw builds a dependency structure, a propagation graph as described in the next Section, that represents the latest state

of the system and allows users to replay and reuse parts of the analysis. The dependency graph is also responsible for propagating the change in the system and updating all variables in the system that were affected by a change or update in one entity.

CZSaw data views for representing data are inspired by Jigsaw [76], which is a visual analytics tool that visualizes documents and the elements inside them. It provides a number of coordinated views for visualizing text documents. In addition, CZSaw records and represents the sensemaking process. Pirolli et al. [69] describe the sensemaking process and capturing and visualizing the sensemaking cycles, and they describe intelligence analysis as an example of sensemaking. Yi et al. [91] argue that interaction is important in the field of information visualization and introduce seven categories of techniques used for making interactive visualizations, which are organized based on user's intent, instead of low-level detailed interaction. Robinson et al. [73] discuss capturing low-level changes to the state of the system and saving them in order to be able to use them later. Amar et al. [7] introduce ten low-level analysis tasks in information visualization tools for understanding the data.

In addition to common capabilities of VA tools, CZSaw visually represents the history of the user's interactions with the system. Plaisant et al. [70] argue that reviewing past events and providing users with a recorded history of their actions has a positive effect on active collaboration. It also helps learners to improve their progress by viewing their previous steps and making revisions. Gotz et al. [32] present insight provenance, and historical record for the process and methods that lead to deriving insight from data according to semantically meaningful analysis actions. Heer et al. [35] state that interactive tools for visualizing history could cover a vast range. They present a method of displaying the history for Tableau. For increasing user support, Kreuseler et al. [49] provide undo and redo for their tool using a captured history. Hanrahan et al. [34] present a history model and methods for capturing and displaying history for Tableau, which is similar to CZSaw's history view. Unlike CZSaw, they only show one branch in their representation of history. M. Derthick et al. [20] provide browsing for the history in addition to selective undo/redo. Having branching allows them to explore and compare multiple scenarios. Like CZSaw, they use a tree structure for the history.

The next Section, Section 2.2, addresses a very seemingly different literature related to the current work. Parametric modeling provides the basic structure behind dependency graphs,

or propagation graphs, since they propagate every change through a subsequent part of the graph and keep the graph updated at all time.

2.2 Parametric Modeling and Propagation Graphs

CZSaw's dependency graph is in essence a parametric system. So in this Section, the background of parametric modeling and propagation graphs is discussed. An important current trend in computer science and especially in computer-aided design (CAD) systems is the parametric design of the artifacts, not just the representation of the artifact itself. A parametric design allows change in the system. Parameterization has many benefits; it can decrease the time and effort for applying changes to the design and allow reuse. It can help the users to understand the conceptual structure better, and it can also facilitate the discovery of new design forms. But on the other hand, parameterization may increase the complexity of design decisions on a particular artifact and may cause additional effort for the designers as well as the developers of the system [5].

Examples of systems that are based on parametric design include spreadsheets, software engineering diagrams like the data flow diagram, project management tools, etc. A propagation-based system, shows the importance of *"determining how the relationships are processed in addition to specifying the relationships the relationships themselves"*, according to [5]. Generative Components (GC) is such a system, described in the same paper.

According to Woodbury and Aish [89], *"in parametric modeling, patterns can be used to describe a tactical level of work, above mechanics and below design"*. Patterns provide generic approaches to the different categories of problems in a field. This paper describes three patterns in the field of architecture using an example model. The term design patterns' has its origins in the architecture field [6], and was later used in software engineering community for categorizing abstract groups of code in the program structure. Patterns usually consist of the pattern name, the problem they are trying to address, an abstract solution, and the consequences of applying the solution. Woodbury and Aish [89] try to help designers use propagation-based parametric modeling by developing patterns. As mentioned before, a propagation-based system represents the latest state of design as a directed acyclic graph (DAG), with the two algorithms for sorting the graph and propagation of change. More

information on patterns and their usage in design systems can be found in [80], [26] and [21].

Qian et al. [72] argue that understanding the recurring patterns in the design work and making them explicit will help designers to learn better. It will also improve the result of the design work. In their study, designers acted as tutors. With observing and discussing the process of design, they found that they were using previously known patterns and also discovering new ones.

2.2.1 Graphs and Propagation-Based Systems

Propagation-based systems are useful in increasing the efficiency and performance. They are also predictable and algorithmically easy. They are normally represented as directed acyclic graphs (DAGs), using two major algorithms for ordering of the graph and for propagation of value in the structure [89]. The following Section briefly describes graphs and their representations, and then shows the usage of graphs in propagation-based systems.

Graphs

Graphs are one of the most common and popular data structures in computer science. Graph algorithms are among the most fundamental and popular research areas in the field [19]. A graph is a structure consisting of vertices (or nodes) joined together by edges (or links). A graph G is a pair (V, E) where

V is a finite set of elements, called vertices

E is a finite set of pairs of vertices, called edges

If the pairs of E are unordered, then the graph is undirected. And if the pairs of E are ordered, then the graph is directed.

Directed Acyclic Graphs (DAG) A DAG is a directed graph with no directed cycles. Meaning that there is one can not start at a vertex v and follow a sequence of directed edges that eventually leads back to v again. The edges in a DAG can be drawn in a way that all have the same direction, which means there are no *directed* cycles in the graph.

Dependency Graphs

As stated before, dependency structures are commonly represented as DAGs. In a dependency graph, the nodes of the graph are objects which contain variables and constraints among the variables. The edges show the dependencies between the nodes (and their variables) based on the constraints. Each node has an algorithm for updating the node based on its type, for calculating the values of the node variables according to the constraints. A propagation graph represents a collection of instances, with each instance being the graph that is the result of assigning values to independent nodes. For computing an instance, nodes are first sorted based on the topology of the graph, and then the updated values are propagated in the graph [5].

A large group of conventional computer-aided design (CAD) systems can be categorized into two main groups. The first category, the drawing tools, focuses on geometric objects and/or drawings, which decreases adaptability and makes applying a change in the design difficult and labor intensive. The second category, the so-called intelligent tools, invokes the help of object oriented design, but the tools in this category typically frustrate innovation by putting the focus on documentation. A more recent approach to this problem is to use parametric modeling represent and visualized variations in design, in both the interface and the background computations. The importance of this is particularly revealed when the change and variation is continuous in the design process [72].

As mentioned before, our dependency structure on the system variables is a propagation graph. To be useful to analysts, this needs to be visualized in CZSaw. The next Section (2.3) is dedicated to reviewing the methods and techniques of visualizing graphs in the literature.

2.3 Graph Visualization

As described in the Section 2.2.1 a graph is a data structure $G(V, E)$ where V is a set of vertices connected by a set of edges E. Dependency graphs are normally represented as graphs, or to be more specific directed acyclic graphs. In this Section, different methods, techniques and branches of graph visualization will be discussed.
In their book “Bayesian Artificial Intelligence” [47], Korb and Nicholson state that graphical models have many benefits; they simplify computation, hide parameter details, and enhance

intelligibility for elicitation, validation and explanation. The book introduces the work on charts for legal reasoning by Wigmore [85] as the first systematic use for qualitative graphs. It also states that Wright's work on building models for analyzing population genetics and his statistical method of path analysis are probably the first systematic uses of quantitative graphs [90].

Graph visualization research has received a lot of attention and is still an active area where new challenges constantly arise for different methods and different application areas. Visualizing, exploring and navigating graphs become more complex when graphs become larger and more abstract. Through their survey, Herman et al. [36] provide a different perspective on traditional methods and issues in graph drawing, from an information visualization perspective. There is an increasing need for visualizing extremely complex data sets as graphs, exploring and navigating them, and performing queries such as search on them [87]. In their book [45], Kaufmann and Dorothea talk about graph drawing and its applications, special types of graphs, 3D graphs and much more.

2.3.1 Applications

Graph visualization in general has been used in many applications from many different fields. A simple example is a file hierarchy of a computer system, which can be shown as a tree. The common interactions and queries related to tree visualization system also applies in computer file hierarchy. Tasks such as exploring, navigating a path, searching for a specific file and other similar tasks are performed by computer users on a daily basis. Some of the other applications and examples of graph visualization that are related to current work include: CAD systems and parametric modeling, visual analytics, software engineering, and many more. Many of the applications have a hierarchical format, but this is not the case for many other applications, and more general graphs are widely used [36] [51] [87].

DAG applications

DAGs also have been used in various applications. Among them is the book by Leymann [58] that describes web services compositions. The meta model underlying their flow language has a syntax, implementing a special kind of DAG.

2.3.2 Graph Quality Criteria

The book *Drawing Graphs, Methods and Models* presents criteria for determining graph quality [45]. According to this book, all graph visualizations can be divided into two categories: static and dynamic graph drawing. The standard features for static graph layouts include:

- minimum edge crossings
  - planarity
- minimum node overlaps
- layout symmetry

These challenges and various attempts for solving them are discussed further in this Section. Also, it is worthwhile to mention that for dynamic or animated layouts, determining the stopping condition after a transition in the layout state is an important concern.

2.3.3 Tree Visualization

An undirected graph is considered *connected* if there is a path between any two arbitrary vertices. A tree is a type of connected graph that contains no cycles. Trees are a special kind of graph that show hierarchies well. Trees typically have many leaves, which crowds their visualization in lower tree levels. A famous algorithm for drawing tidier trees was first presented by Reingold and Tilford [23], which takes into account the aesthetics of the tree as well as being efficient on the drawing space. The algorithm can also be generalized to forests (collection of trees). Latour is a tree visualization system introduced in [37] that provides spanning trees for DAGs using weights based on the layer numbers. The paper presents the characteristics of this system and the principles that can be generalized to graph based information visualization systems.

2.3.4 DAG Visualization

The same factors mentioned in Section 2.3.2 also apply to DAG layouts. The literature in this area is more focused in the following factors: removing node overlaps, reducing the total area, reducing graph visual complexity. In both trees and DAGs hierarchies play an

important role. CZSaw's dependency graph is a DAG that is displayed as a hierarchy of different layers based on its topological sorting. A part of the research in graph visualization area is dedicated to the graphs with layered structures. Junger and Mutzel [41] present an algorithm for the problem of two layer edge crossing minimization. Even though the general problem is NP hard, they address the issue by keeping one layer fixed. Sugiyama et al. [78] provide a layout for general directed graphs considering both layering, and the position and order of the nodes in the same layer.

2.3.5 Graph Layout

The survey by Herman et al. [36] on graph visualization provides an overview on traditional layouts for drawing graphs.Battista et al. [10] present a survey on what a good graph drawing is, using hundreds of papers. One important point is that according to the type of graphs, there are different properties and classification of the layouts that apply to them. For example, planarity is a graph property, which indicates whether a graph can be drawn on the plane without any edge crossovers. Layout algorithms can be classified with regards to the type of layout they generate. They can also be classified based on their underlying methodology, e.g. algorithms that use force-directed models. The set of factors and problems related to each layout is affected by the class that the graph and its layout belong to. Spring layouts are a group of non-deterministic force-directed layouts [36]. Force-directed layouts try to locate the positions of the nodes in a way that all edges share properties related to the "force" modelled. They also try to minimize edge crossings. For visualizing undirected graphs, spring layout algorithms are considered effective tools. They particularly display the symmetric properties of graphs [22]. The spring algorithms assume an spring with attractive and/or repulsive spring forces between the nodes of the graph, and modify these forces to reduce the amount of node overlap as much as possible [51].

As for DAG layouts, Kumar and Zhang [51] provide a new stopping condition for E-Spring algorithm [52], which works on clustered DAGs. The algorithm significantly reduces the distances between nodes at the equilibrium state. This results in minimizing the total allocated graph space, which is the main contribution of the paper. They take a step forward by using node interleaving and thus optimizing the area further.

Dynamic Graph Drawing

In today's application, there are many cases when the data is not completely ready at the time of the start of the application and will change throughout the application. Also, many applications aim to support user interactions. Hence, more recent graph drawing algorithms support dynamic and incremental graph visualization algorithms and layouts. CZSaw's dependency graph is constantly changing throughout the process of the application and allows interactivity, and so it should be able to support dynamic drawing.

Frishman and Tal's paper [29] includes the necessary procedure for building a dynamic layout from scratch. The procedure concludes from six steps: initialization, merging, pinning, geometric partitioning, layout (modified force-directed algorithm), and animation. This paper has a few contributions, such as efficient dynamic algorithm and use of GPU; it also presents a visualization of evolution of discussion threads.

Incremental drawing of DAGs is also discussed in [68]. This paper discusses incremental exploration and navigation in graphs, in addition to the ability to modify the graph through incremental layouts while maintaining dynamic stability. The proposed new heuristic is applied in the DynaDAG system.

But there is very little research for the occasions where the whole structure of the graph is not known prior to drawing the graph. Gorg et al. [33] develop an algorithm for drawing hierarchical graphs dynamically while preserving the *mental map*. The algorithm provides a balance between the graph quality and the dynamic stability.

Mental Map Misue et al. [63] propose the concept of mental map in their paper. They claim that interactive systems need a facility to adjust a layout when it goes through a change. The layout should not be created again because it will completely rearrange the content and thus destroys the users' mental map of the diagram. Some layout adjustment methods that help users keep their mental map of the visualization are discussed in the paper. One way of mental map preservation is to rearrange a diagram to avoid overlapping nodes. Another method needs changing the user's focus of interest. Preserving the viewer's mental map along with the stability of the layout is further discussed in this paper.

2.3.6 Problems in Graph Visualization

There are many challenges that arise while dealing with information visualization for graph-structured data. Among the most important issues in this area, we can name edge congestion and node overlapping problems that are discussed in this Section.

Edge Congestion

Many challenges in graph layouts are described in Battista's survey [10]. A major problem in graph drawing is edge congestion. The positioning of edges could have a high effect on the way people read graphs [71]. The problem increases in difficulty when size and complexity of graph increases. Optimal solutions to the edge congestion problem are difficult to find [36]. There have been several attempts and approaches to the edge crossing problem. The methods generally fall into one of the following categories [87].

a) Layout. Knowing that graph layouts, manual or automatic, can potentially reduce edge crossings has lead to several techniques in this area. But the optimal solution has shown to be difficult to find using these techniques [10] [86] [36].

b) Filtering. This approach tries to keep only the important edges in the layout to reveal the important relationships, by removing the unimportant relations between the nodes [30] [18]. But a problem with this approach is that the viewers may become confused or lose how the nodes relate to each other when some of the edges are invisible. Also, this means that the layout algorithm should a way to distinguish important' edges from the unimportant' ones.

c) Magnification. Another approach to solve the edge crossing problem is to enlarge areas of the graph, linearly or non-linearly. The methods in this category include Magic Lenses [15], zoomable user interfaces [14], insets [83], fisheye views [30], and other distortion based techniques [46] [57]. A common problem in all these approaches is that enlarging does not necessarily disambiguate edge congestion.

According to Laguna et al. [54], an important problem in graph drawing is minimizing edge crossing in a multi layer hierarchical graph. In their paper, they present a search method that works based on a local optimum finding while insertion. They also have an extensive review of the relevant work in this field.

One approach to minimizing edge crossings is to change the layout of the original graph, by means of changing the position of the nodes in a way that reduces the edge crossings or occlusions [87]. But this approach is shown to be difficult [10]. In general, optimal solutions for managing edge layout and edge congestion in large graphs are hard to find. Also, this approach cannot be applied to all types of graphs. Even if reorganizing the nodes of a graph to reduce edge crossings was applicable to the graph in theory, there's still a good possibility that much meaning about the data and the structure would be lost. This specially becomes more challenging in CZSaw's dependency graph visualization, since the layout preserves the hierarchy based on the nodes' levels. This further constrains algorithms for minimizing edge crossings to have more restraints, since not preserving nodes' levels could be harmful to the information the directed acyclic graph is conveying. However, this is for the initial automatic layout by CZSaw's dependency graph view, and the users have the ability to reposition the nodes later on.

But when we keep the original set of node positioning, there's still the possibility of loss of meaning because of the way edges overlap and can obscure nodes and other visual data. Wong et al. [87] present a solution for this problem by generating interactively curved edges from the viewer's focus of attention.This method of curving the edges was used in CZSaw's dependency graph.
The problem of edge crossing also occurs in layered or hierarchical graphs in particular, since because of the nature of their structure, altering node positions is not always easy or possible. Laguna and Mart attempt to enter as little edge crossings in a 2-layer graph as possible by their greedy algorithm [53]. They conduct extensive computational experiments investigating performance, which show their algorithm to be highly influenced by the graph density. The problem gets more complicated as the graph grows in size.

EdgeLens [87] was introduced to address the problem of a high number of inter-connected nodes that cause edge congestion in complex graphs. It is an interactive technique that curves graph edges based on the viewer's focus of attention, keeping the positions of the nodes fixed that preserves the node layout. In the paper, they developed two methods, and ran a study to make a comparison between them. They also seek help from using transparency and color to have better outcome.

Wong et al. [87] present an interactive technique that reduces edge and node overlapping, and helps better revealing the graph structure, while preserving the node layout. They distort shape of the edges as was shown in figure **??** to reveal underlying data in the graph, so that the viewer can see how nodes are truly connected. They apply the distortion only to the edges, borrowing the idea from detail-in-context distortion based viewing. They build to types of EdgeLenses, using the bubble approach and the spline approach (smooth curves). The results of their user study shows that people are faster at completing path-finding tasks using the spline method versus the bubble method. Also, participants using the spline approach had far fewer incorrect paths, and they produced more optimal paths. In the end, all participants preferred the spline approach.

A planar graph is a type of graph that can be drawn in a way that there are no edges crossings. Planarity is one of the desired features in graph visualization. Spanning trees are sometimes used as the solution for planarity and edge crossing problems in large graphs; since the tree layout has the lowest complexity. The additional edges are added later based on need [36]. In his book, "Graphs, Networks and Algorithms" [42], Jungnickel describes how to detect the spanning trees for graphs. For DAGs, spanning trees could be found using edge weight based on the layer number in the work of Herman et al. [37].

Node Overlapping

Another common problem in graph visualization is node overlapping. The problem exists because many graph layout algorithms represent vertices as points without any dimensions. But in practice, nodes may have labels and hence there may be overlaps in the spaces needed for displaying the labels. A more detailed description of node overlap problem and the methods and approaches for reducing the overlap is stated in Kaufmann's book [45] and the Graph Drawing book by Battista et al. [11]. In spring layout algorithms all edges have quite the same length and they have as few overlaps as possible.
The problem of overlapping nodes shows itself better when the nodes must be labeled, especially if the visualization is dynamic. Huang et al. [39] provide a new approach for this situation, called Force-Transfer algorithm, along with a formal description of the problem. Lai's work [55] introduces an approach for graph layouts that through some adjustments removes node and edge crossings. The algorithm also takes into account the viewing area through boundary detection to fit the diagram.

2.3.7 Navigation

Regardless of whether the graphs are static or dynamic, hierarchical or not, and directed or undirected, the navigation issue has always been a challenge. Navigation and interaction are crucial in many applications in information visualization, especially when the size of the graph grows and the viewers need further assistance for revealing the underlying structure. Pan and zoom are traditional visualization tools. Zooming can be geometric or semantic; semantic zooming has to address the appropriate level of zoom on the content. Pan and zoom are conceptually fairly simple, but practically, when users have zoomed into an area, and want to pan, they usually need to zoom out, pan, and then zoom in again [36]. The solution to this problem was presented by Furnas and Bederson [31]. Their space-scale diagrams provide a framework yielding different zoom factor pan positions, which represents both a special world and its different magnifications.

Focus + context methods were introduced to improve the issue of loss of context in navigating and interacting with the graph. Schaffer et al.'s work [74] takes into account multiple algorithms. Using semantic clustering of nodes, they represent clusters as glyphs and treat them as super nodes. They also describe the importance of clustering and consider the graphs that already have a hierarchical clustering. Following their work, Huang and Eades's system uses force-directed animated graph drawing to visualize large graphs [38]. Their system, DA-TU, also uses an animated clustering method.

Navigation and incremental exploration become more challenging for extremely dynamic graphs like the World Wide Web. North [68] covers these topics along with the ability to modify the graph, while preserving the dynamic stability. They generate incrementally stable layouts for visualizing dynamic graphs. Incremental drawing of DAGs is discussed in this paper, which proposes a new heuristic for such layouts, and its application in DynaDAG system. Zeiliger et al. [92] investigate incremental exploration and navigation in large graphs like WWW. The NESTOR NAVIGATOR system that was presented by Eklund et al. [24], is a graphical web browser that supports collaborative learning. With users' interactions with the tool, it incrementally builds an overview map of the graph. In spring layout algorithms, the process of assigning spring forces to nodes is usually repeated iteratively until it reaches

to an equilibrium state. Many algorithms use animation to display the transitions to help preserve the viewers' mental map [22].

To assist the navigation model, Li et al. introduce a layout for generating visualization of clustered graphs in dynamic graph drawing [59]. At the equilibrium state of the spring layouts, node overlaps are also gradually reduced. Huang et al.'s algorithm that was described before, also removes node overlaps in dynamic graph drawing [39]. Dynamic DAGs are also often animated for dynamic data. E-Spring algorithm [52] animation for visualizing clustered DAG in a way that preserves the viewers' mental map and eliminates node congestions. Frishman and Tal present a dynamic drawing of clustered graphs with previously unknown sequence of data [28]. Also, Lee et al. [56] propose an algorithm to release the user from the burden of relearning the redrawn layout after a change in the graph, by maintaining a high degree of mental map. This is suited for dynamic graphs where changes are constantly applied to the graph to reflect the evolution of the system behavior represented by the graph. According to Kumar and Garland [50], not many layout algorithms support dynamic time-varying graph drawing. As mentioned before, applying the layouts independently to each temporal state of the graph in time will destroy the mental map. In their paper, they present a new technique for hierarchically structured dense graphs. They also perform an abstraction of the underlying data which filters edges and nodes. Users can then drill down in the layout if they need more details. Brandes et al. [16] also adds smoothness to the animation.

2.4 Causality Visualization

Causality visualization is considered "one of the top 10 unsolved information visualization problems" by Chen [17]. An effective visualization helps detecting causality, proposing hypotheses and assessing them. In order to do so, users need to be able to interact with the visualization to perform the exploration and decision-making tasks on data.

Causal Maps and DAGs

The most common forms of visualizing the causal relations are still directed arrows [12] [81]. The combination of entities and arrows are normally represented as node-link diagrams or

graphs. Directed acyclic graphs (DAGs) are graphs with directed links between nodes that represent the causal relations. These graphs are also referred to as dependency graphs or causal maps [12] that represent ideas and actions as nodes that are connected together through arrows. Nadarkani and Shenoy [65] bring the idea of Bayesian networks and the domain knowledge of experts to causal maps and enhance causal maps through this.

2.4.1 Motion

Causal maps and directed acyclic graphs are widely used for representing causality. They perform very well for displaying the static causality relationships, but they stand short when it comes to visualizing the dynamic changes that happen to the structure and content of cause and effect relationships [12] [9]. This becomes a bigger challenge when today's applications are getting more dynamic and interactive every day.

Animation

In order to enhance the capabilities of causal maps or static dependency graphs, the notion of animation and motion were used in more recent applications. Neufeld et al. [66] show that animation helps in remembering paths in general.

Kraemer and Stasko [48] present the characteristics that are necessary for visualization techniques in their paper. Based on these rules, they built a system, called Parade, that uses animation for ordering display events of the system. Moses et al. [64] introduce a causal visualization system called VADE, as well as its conceptual visualization model.
Kadaba et al. [43] compared animation in directed graphs to both textual descriptions of causal maps, and static representation as a directed graph. They found that animation improves viewers' perception of causality, as well as their memory of the representation. They showed that adding animation to the previous representations helps the understanding of the path and the effect of the causal event. Elmqvist and Tsigas [25] presented new techniques called Growing Squares and Growing Polygons, using color, texture and animation. Through conducting user studies, they compare these two methods with conventional Hasse diagrams, which are simple directed acyclic graphs.

Visual Causal Vectors

The concept of visual causal vectors (VCV) was introduced by Ware et al. [84] in order to address the shortcomings of static dependency graphs. VCVs use animation in order to help the process of detection of the causal relations and paths. They try to help the users perceive the source of the change and its effects in the graph, by animating an auxiliary object that starts its movement from the source of the change, or the cause, and moves to the target node, or the effect, for each pair of cause and effect. They proposed three different approaches for VCVs, called pinball, prod and wave. In the *pinball* metaphor, a small token such as a ball, moves from the initiator of the change to the node that gets affected, and causes the second node to oscillate or swing. The *prod* method similarly causes the second node to oscillate, but it does that through extending a rod from the first node to the second one. The third visual causal vector metaphor launches a wave-like object to move from one node to another. The *wave* method then caused the target node to simulate floatation when it receives the wave.

Neufeld et al. [66] investigated the wave metaphor for visualizing causality in node-link diagrams, as well as a second type of animation called light "comets". They compare these with a directed-arrow representation to investigate whether causal visualizations would help the viewers to better remember the causal paths. The results of their study did not show any significant difference in favor of causal representations. However, they found animation to be more effective than static representations. In their work, they also addressed the issue of timing. Timing is one of the important factors when dealing with causality visualization, or the concept of cause and effect in human thinking in general.

A more recent and related work to the current project, is Bartram and Yao's work on animating causal overlays [9]. Their work investigates the effects of adding a layer of animated representation of causality, or visual causal vectors, to existing visualizations. They study these causal overlays through studies on perception of causality when motion is added to the objects in the visualization. They investigate how these methods convey the causality and also the strength of the effect of the causation.

Motion Attributes

There has also been some research on the types of motion attributes that are considered important for causality visualization. Motion type, phase, direction and velocity have been shown to be effective means for conveying the concept of cause and effect in dependency graphs [8] [82] [40].

Chapter 3

Dependency Graph

As mentioned in chapter 2 CZSaw's dependency structure is a propagation graph. This dependency graph was built with the intention of revealing the underlying model of the analytical process. It represents the current state of the system as a node-link diagram. In this graph, nodes are the variables in the system and links show the dependency relations between nodes. Nodes and links are added to the graph based on the results of the execution of CZSaw's script statements.

This structure provides the ability to propagate change in the system. When analysts want to apply (parts of) the same analysis process on different data, they can just update the value of an entity and the graph will update all dependent variables automatically. This capability saves analysts' time and effort, and also allows them to observe the effect of their actions. Having the dependency graph, users can find repetitive patterns in their analysis process and (especially using the history view) and re-use and replay them as described.

During this chapter, we first describe the architecture of CZSaw's dependency graph. The main functionalities and the important algorithms are described in the next section. The next section is dedicated to the dependency graph view in CZSaw. The chapter ends by proposing the second version of dependency graph, which was separately implemented, and improved the first version in terms of functionality, performance, and code structure.

3.1 Architecture

The architecture of the dependency graph support the model view controller (MVC) architecture pattern. The controller package handles all the background logic, while the view package is restricted to the visualization layer of the dependency graph.

3.1.1 Controller

This class contains the classes that implement the logic and the algorithms that run the dependency graph, as well as the different types of the nodes that can get added to the graph. This package is described through the main classes in this section. The main functionalities are described in the following section.

The Graph

The `CZDepGraph` class is the main class that contains most of the logic of the dependency graph. This class handles all the operations on the nodes and the edges. It is also responsible for updating the nodes and propagating the update through the graph. This class fires the instructions for the dependency graph view when an action happens in the graph to keep the visualization updated constantly.

The Nodes

The `CZDepNode` class represents the nodes that can get exist in the graph. It contains a generic attribute representing the value of different types of nodes. Objects of this class have an update method that evaluates the new value of the node n based on the nodes that n depends on. This method gets called through the update propagation process in the graph to update the value of individual affected nodes.

Types of Nodes

There are different classes for different types of nodes that can be added to the dependency graph, and they all extend the `CZDepNode` class. The list of the classes representing different types of CZSaw dependency nodes is summarized in table 3.1. These small classes were implemented by other members of the CZSaw team responsible for the CZSaw script.

Class	Description
CZConstantDepNode	Constant string value, not dependent upon any nodes
CZDataSourceDepNode	Data source, not dependent upon any other nodes
CZEntitiesByTypeDepNode	All entities of one type from
CZEntitiesDepNode	Entities specified by entity ids
CZEntitiesSubsetDepNode	Subset of another list
CZEntityListDepNode	List of entities
CZNoteDepNode	Data for a CZNote
CZRelatedDataDepNode	Based on a relation from one or more other lists
CZRelationDepNode	Based on a relation from one or more other lists
CZSearchDepNode	Data based on a search
CZSemanticSearchDepNode	Data based on a search
CZViewDepNode	Data view, depends upon the nodes displayed in it

Table 3.1: Classes representing different types of dependency graph nodes

3.1.2 View

The view package is responsible for the dependency graph view. It handles the visualization of nodes and edges, managing the layout and the smooth animation. It also overlays the highlighting causality visualization technique on the graph to help the viewers identify the dependency relations when the change gets propagated through the graph. The view is described further in this chapter. Here we just briefly mention the most important classes that contribute to the architecture.

Dependency Graph View

The `CZDependencyGraphView` class extends the `CZView` class, which is the super class for all CZSaw views. This is the main class for handling the visualization of the dependency graph. It uses the JUNG [2] framework for managing and displaying the graph. It uses a DAG layout to incorporate the topological ordering of the nodes to the visualization (the details about the layout are discussed in the visualization section of this chapter). When an action such as adding or removing a node, adding or removing an edge, or updating a node happens in the dependency graph, the appropriate method of the dependencyGraphView class gets invoked to update the visualization accordingly.

Control Panel

The `CZDepGraphControlPanel` class is a control panel on the right side of the dependency graph view. The basic interactions with the graph are handled with the mouse. But the more advanced interactions are managed through this control panel. It is responsible for performing the user commands on the graph, such as updating the layout, filtering the constants, etc. It's other important responsibility is to display the information about the node selected by mouse click. When the user clicks on node n of the dependency graph view, the control panel displays its name, type, and value. It also displays two lists that contain the nodes depending on n and the nodes that n depends upon.

Node View and Edge View

The `CZDepNodeView` and the `CZDepEdgeView` classes are used to represent the graph nodes and edges in the view.

Updating Node

When a node gets updated in the graph, first of all, this change is visualized via animation (highlighting the node). Then the update gets propagated to directly and indirectly dependent nodes and their update gets animated as well. The animation takes time and the rest of the execution cannot be delayed until an animation is finished. Also, the updates can happen concurrently in the graph and they do not happen sequentially. Due to aforementioned reasons, a separate Thread is allocated for each updating node. The `UpdatingNode` class extends the `Thread` class and starts its life cycle when a node gets updated by user instruction or be receiving the propagated updated from another node. The thread gets terminated after the animation is finished.

3.2 Design and Implementation

The dependency graph was implemented in Java programming language and API, like the rest of the CZSaw visual analytics tool.
Most of the functionalities of the dependency graph initiate from the script. When CZSaw's script is parsed and the script statements are getting executed, appropriate dependency graph methods get called. The main methods for interacting with the dependency graph

include adding nodes and dependency relations between them, removing nodes and dependencies updating a node and propagating the update to the whole graph.

3.2.1 Main Use Cases

The main functionalities of CZSaw's dependency graph are described from a high-level point of view.

Adding a Node

When a new entity gets created in the system, a node representing that entity gets added to the dependency graph. The new entity could be user defined or system defined. CZSaw defines its own set of nodes that can be added to the dependency graph. Different types of dependency nodes are summarized in table 3.1.

Example scenario:
An example of a scenario that adds a new node to the dependency graph is shown in figure 3.1. In this example, the analyst searches for the term "nicolai", which is the name of a person of interest that the analyst wants to investigate. The analyst chooses to give the result of the search a name ($var1$). By pressing the search button, the search will be performed and the results are put into $var1$. By execution of this statement, a dependency node depicting $var1$ is added to the dependency graph.

Adding a Dependency Relation

For making a dependency relation between two nodes, a directed edge could be added to the graph. An edge should be added between two existing nodes. However, there is another method that combines the node and edge insertion to the graph. This method gets two nodes and an edge as input, creates the nodes if they do not already exist, and then builds the dependency relation.

Example scenario:
For describing an example scenario for adding a dependency relation to the graph we follow

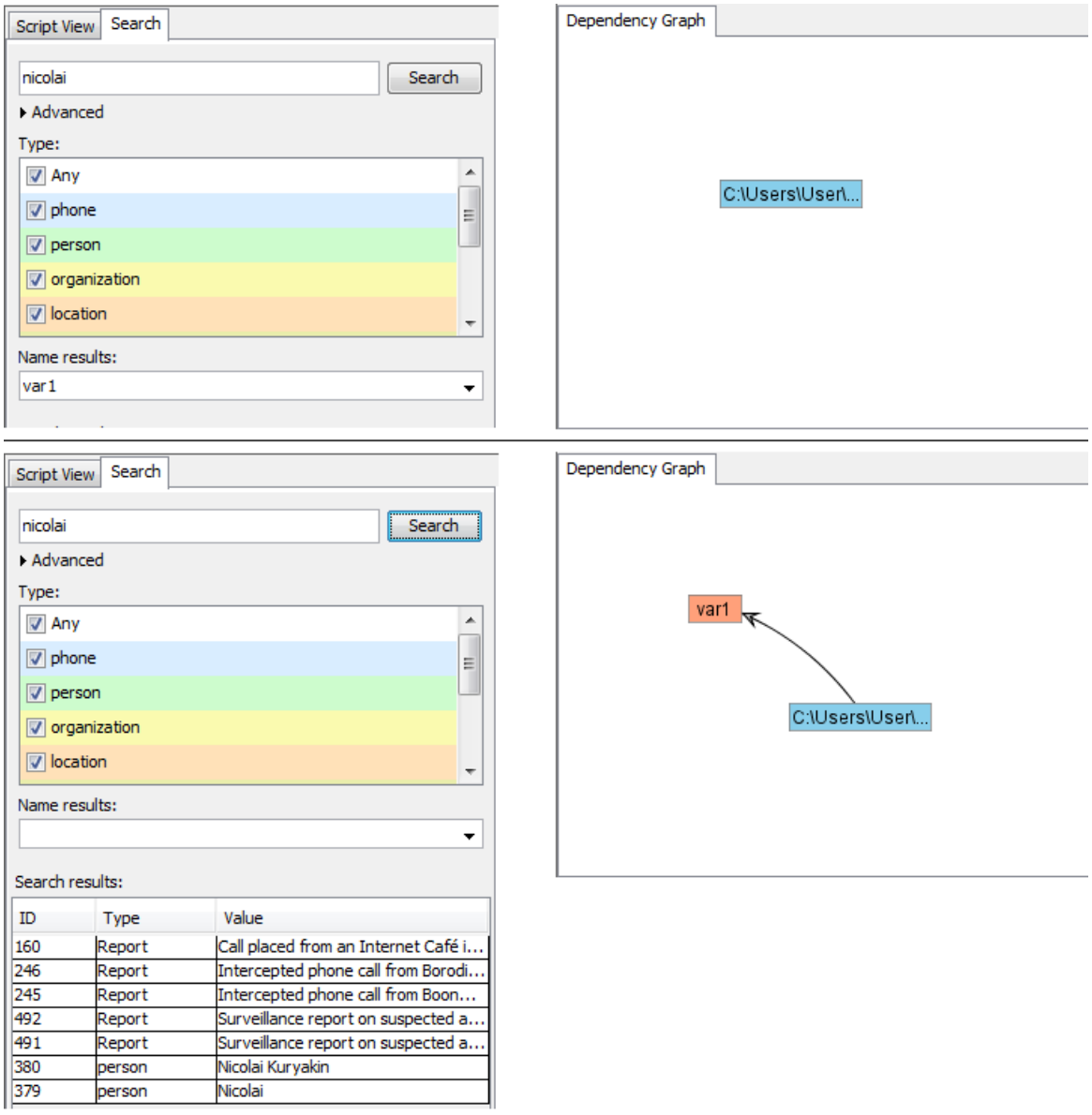


Figure 3.1: An example scenario that adds a node to the dependency graph. In the top part analyst enters the search term and name of the result. In the bottom part the analyst presses search and the respective search node gets added to the graph.

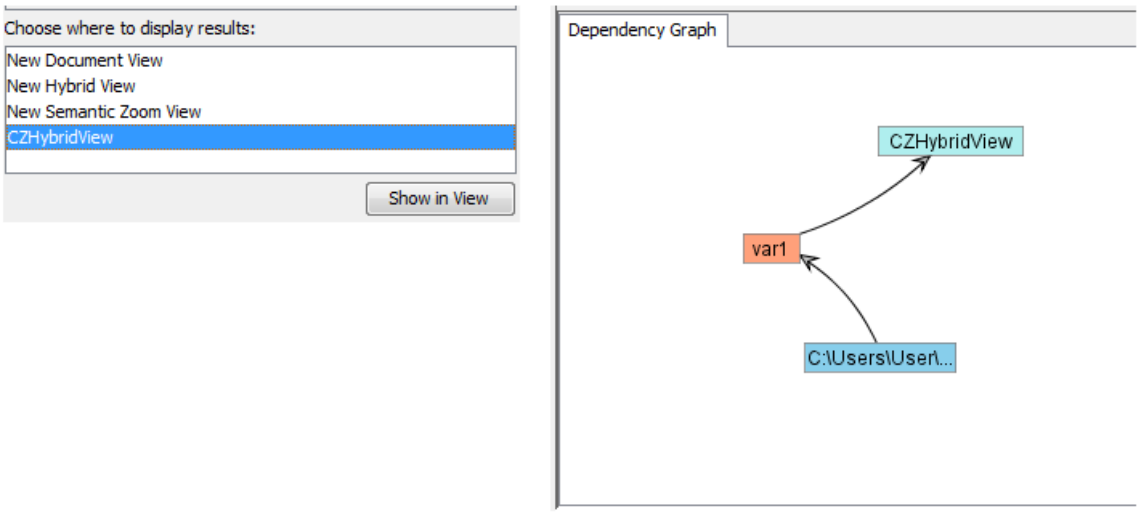


Figure 3.2: This example shows that by visualizing a variable in a CZSaw view, a dependency relation gets created.

the example shown in figure 3.1, where a search result variable was added to the graph as a new node. Consider there is a CZSaw view where we want to display the search results in. By choosing to display the search results in that view, we are indeed creating a dependency relation from the search result node to the view node. Since for the view to be visualized, it needs all its content to be present, including the search node. And any change to the search node will cause an update to the view. Hence, the view is dependent upon the search results and this is represented in the dependency graph using a dependency edge from the search node to the view node. This example is shown in figure 3.2.

Removing a Node

At the time of writing this document, the dependency graph only supports removing nodes that do not have any child nodes (nodes that do not depend on any other nodes). It also supports removing the nodes that only have only leaves as their children, which in the case of the dependency graph are all constant nodes. In future, the current implementation could be extended to be able to remove any node recursively. We also want node removal that does not remove its dependent nodes. The formerly dependent nodes then become invalid and must be repaired.

Updating a node

When an update happens on a node n, after the value of the node is updated, the change get propagated through the graph to the nodes that depend on n, so that they can reevaluate their value based on updated inputs. We next describe the algorithm for propagating the update in the graph and maintaining an updated graph after every change to the structure or the content of the graph.

Propagation of change

algorithm The change in the graph must be propagated to all nodes that directly or indirectly depend on the update node n, which means the ordering of all nodes relative to n matters. The first step in change propagation is to topologically sort the graph to have these orderings. The nodes that are topologically upstream to n are potentially dependent on n. They each get examined to check whether their value really depends on n's value or not. Having the final list of nodes that depend upon n, the update gets propagated to them level by level, leaving the nodes at each level with updated values.

topological sort The dependency graph is a directed acyclic graph (DAG). Directed acyclic graphs can be topologically sorted using a depth-first search algorithm. For a DAG $G = (V, E)$, a topological sort is an ordering of all its nodes, in a linear manner, based on the directed edges. On the list of sorted vertices, item u appears before v, given that G contains and edge (u, v). Only directed *acyclic* graphs can have a topological sort, since if there are directed cycles in the graph, the graph cannot be ordered linearly. A topological sort of a graph can be considered as a sorted horizontal list of all vertices, such that all directed edges have the same direction.

The following algorithm taken from the "Introduction to Algorithms" book [19] is used to topologically sort a directed acyclic graph.

TOPOLOGICAL-SORT(G)
1 call DFS(G) to compute finishing times $f[v]$ for each vertex v
2 as each vertex is finished, insert it onto the front of a linked list
3 return the linked list of vertices

This algorithm uses the DFS(G) algorithm also described in the Introduction to Algorithms book [19], which is an algorithm for depth-first search. The algorithm works by exploring unexplored edges from the latest discovered vertex v that still has edges that have not been navigated. The search backtracks after all of the outgoing edges of vertex v are discovered. The algorithm then proceeds to exploring undiscovered vertices. DFS(G) calls the method DFS-VISIT(U) that recursively traverses the graph and marks the discovered vertices. The next two algorithms are also taken from the "Introduction to Algorithms" book [19].

```
DFS(G)
1 for each vertex u ∈ V[G]
2     do color[u] ← WHITE
3         π[u] ← NIL
4 time ← 0
5 for each vertex u ∈ V[G]
6     do if color[u] = WHITE
7         then DFS-VISIT(U)
```

```
DFS-VISIT(U)
1 color[u] ← GRAP // White vertex u has just been discovered
2 time ← time + 1
3 d[u] ← time
4 for each v ∈ Adj[u]//Exploreedge(u, v)
5     do if color[v] = WHITE
6         then π[v] ← u
7             DFS-VISIT(V)
8 color[u] ← BLACK // Blacken u; it is finished
9 f[u] ← time ← time + 1
```

3.2.2 Visualization

CZSaw visual analytics tool supports multiple views to look into the data from different windows. A separate view was designed and developed for the dependency graph. Unlike other CZSaw views that provide a visualization for the actual data, the dependency graph

view provides a representation of the model of the analysis work. As mentioned before, the dependency graph contains the variables in the system and the dependency relations between them. These variables could be variables generated specifically by the analysts, or by the system during the process of the analysis.

Implementation

The framework used for developing the visualization was the Java Universal Network/Graph Framework (JUNG) [2]. JUNG provides a language for visualization and modeling of a graph. It is written in Java and can be used and extended for programs that need representations of their entities and relations. JUNG supports a variety of graphs, such as directed or undirected graphs, trees, etc. It also provides multiple layouts for these graphs.

Layout

The layout used for the dependency graph visualization was the DAG layout of JUNG. When the layout is rendered, the edges are all sorted in one direction (upwards). The directed edge $E(n1, n2)$ between nodes $n1$ and $n2$ represents the flow of data from node $n1$ to node $n2$, where $n2$ is dependent upon $n1$. Having this layout, a node n depends directly on its children (downstream nodes), and indirectly on the children of its children.
Also, the nodes are positioned on parallel layers in the DAG. The layers are based on the topological level of the nodes. If node n is positioned at a higher level, it means that n has more dependencies on other nodes, and less nodes depend on n. The leaf nodes of the graph do not depend on anything. If a node is located at the highest level of the graph, it means that no other node depends upon it. A screenshot of the dependency graph view is shown in figure 3.3 (a).

Vertices

The vertices of the graph are the user or system defined variables. These variables could be variables representing the results of search queries, instances of CZSaw views, lists of variables, etc. (Table 3.1).
In the dependency graph view, each node is labeled using its unique *idStr*. The labels longer than a threshold are displayed in a shortened way in the graph. However, the users can view the original *idStr* by hovering the mouse over each node. The size of the node is calculated

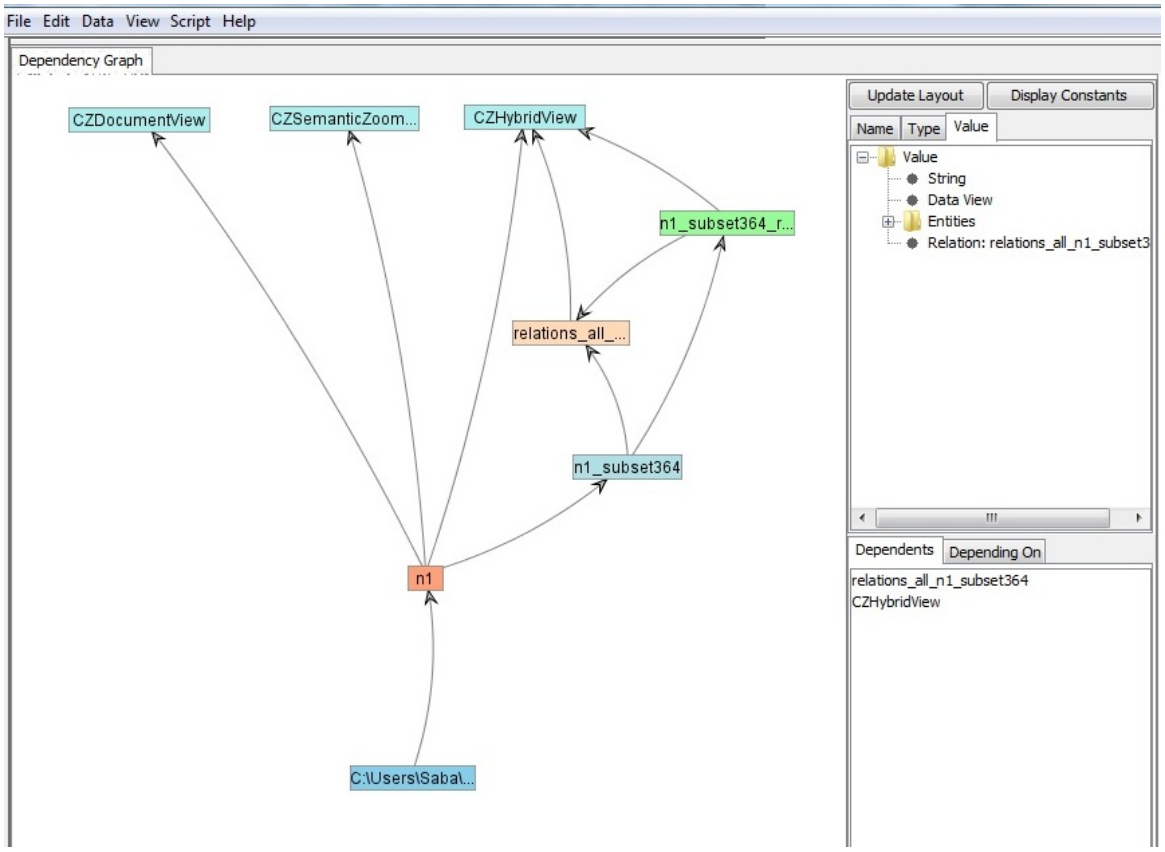


Figure 3.3: An overview of the dependency graph view. (a) The left side is where the graph is displayed. (b) The right side is the control panel

proportional to the length of the *idStr*. The color of each dependency node is selected based on its type. Different types of dependency graph nodes are described in table 3.1
The layout algorithm of JUNG's DAG layout tries to minimize node overlappings.

Edges

The directed edges of CZSaw's dependency graph view are the static means of displaying a dependency relation between two nodes. The JUNG's layout tries to minimize edge crossings, but due to the special nature of the layered DAG layout of CZSaw's dependency graph, there are cases that edge crossings become inevitable. The edges are grey colored and partially transparent to prevent them from blocking other edges and vertices, should edge crossings happen in the graph. The edges are curved in order to help better detection of edges when they have overlaps, as discussed in chapter 2.

Basic Interaction

Multiple ways of interacting with graphs and generally any visualization are supported by JUNG. CZSaw's dependency graph view implements the following basic interactions with the graph using JUNG, using the mouse.

- Picking
- Scaling (zooming)
- Transforming (rotation, shearing)
- Translating (panning)

Advanced Interaction

In addition to the basic interactions with the graph as described above, that could be applied to any sort of graph, more advanced interactions were built into the dependency graph view that better serve this particular dependency graph. A control panel was implemented for the view to give the user control over these interactions 3.3 (b).
The control panel is responsible for the following instructions.

- **Layout update.** During the process of the analysis in CZSaw, analysts may resize the dependency graph view and then update the graph to fit the new size. They may

also manually move the nodes and change the layout, and later decide to retrieve the original layout. In either case, the analysts need to be able to update the layout and redraw it upon request. For supporting this, an *update layout* button was added to the control panel.

- **Node information display.** For each node, the analyst may be interested to view detailed information about the node. By clicking on node n, the information about node n will be displayed in a panel in the dependency graph's control panel. The panel consists of the information about the node itself, along with some information about connected nodes. The information about the node includes the name of the node, its type and its value. While some information about nodes connected to n can give the analyst a better understanding of n and the underlying structure of the graph. This information includes two lists, one for nodes dependent on node n, and another one for the nodes that n depends on. Also, by clicking a node, the connected edges are highlighted to better represent the dependency relations and the structure of the graph at the point of focus (figures 3.4, 3.5, 3.6. 3.7 and 3.8).

- **Node filtering.** In CZSaw's dependency graph view, the nodes in the higher levels of the graph are more general and abstract entities, while the more downstream nodes are closer to the actual raw data. The key terms for searches and other constant attributes of other queries are all stored as the leaves of the graph. Hence, it is reasonable to think that one may want to hide some of these nodes in order to make the graph more readable. To support this filtering, a functionality was implemented for the dependency graph view to hide an arbitrary level of nodes from the bottom of the graph. The number of levels that can be filtered is set to one by default, which filters only the leaves of the graph, which are the constant values. A toggle switch was added to the view's control panel to give the analyst the option to hide the constants, or make them visible again at any time during the process. Note that by filtering the nodes, they are not eliminated from the graph, they just get invisible to the viewer.

- **Node deletion.** There are cases during the analysis process, where the analyst creates variables by mistake, or no longer needs variables that were generated before. In order to help the problem of the graph getting crowded and unreadable, a partial node removal functionality was added to the dependency graph. This algorithm only supports the removal of the nodes that no other nodes depend upon. The reason behind

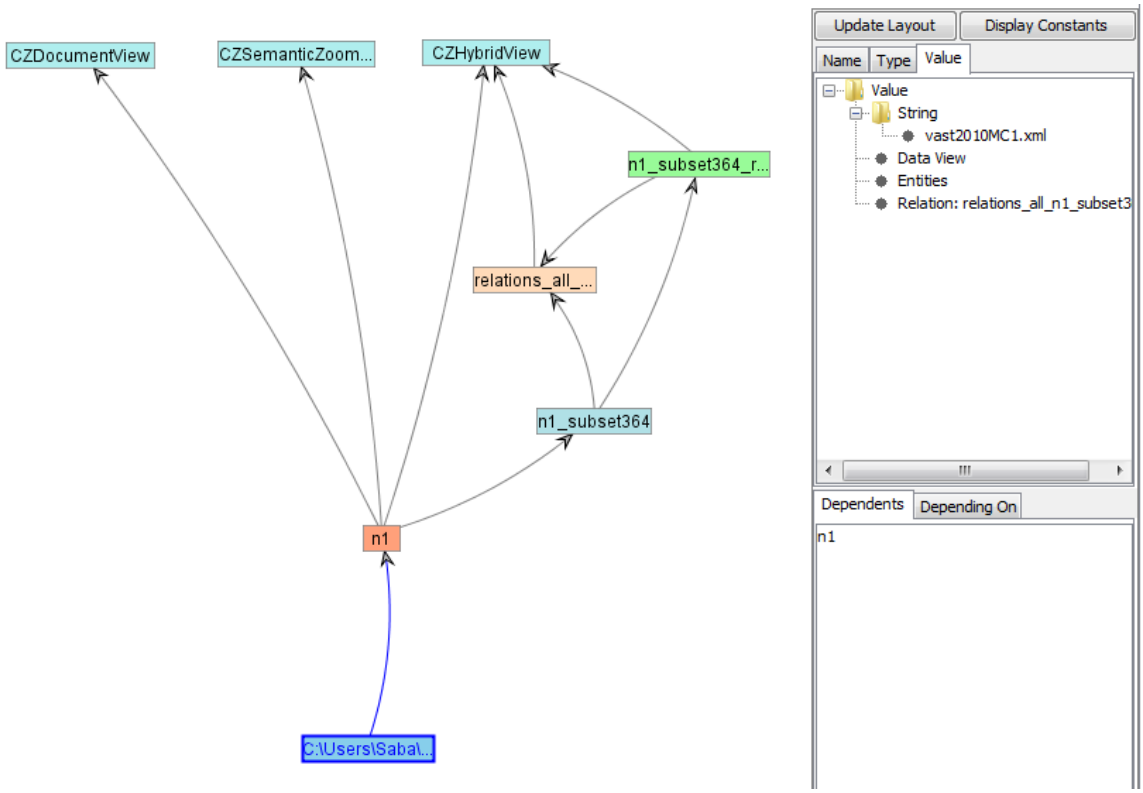


Figure 3.4: Displaying the information about a node with a string value.

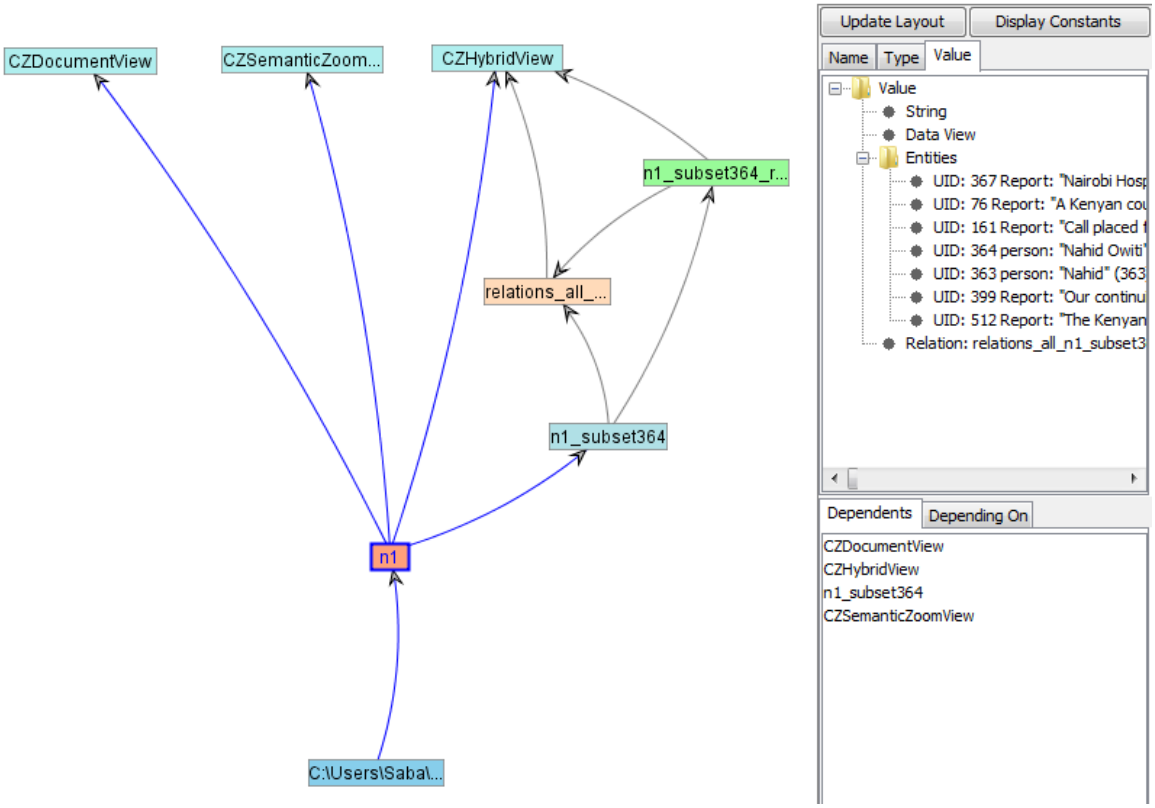


Figure 3.5: Displaying the information about a node containing a list of entities.

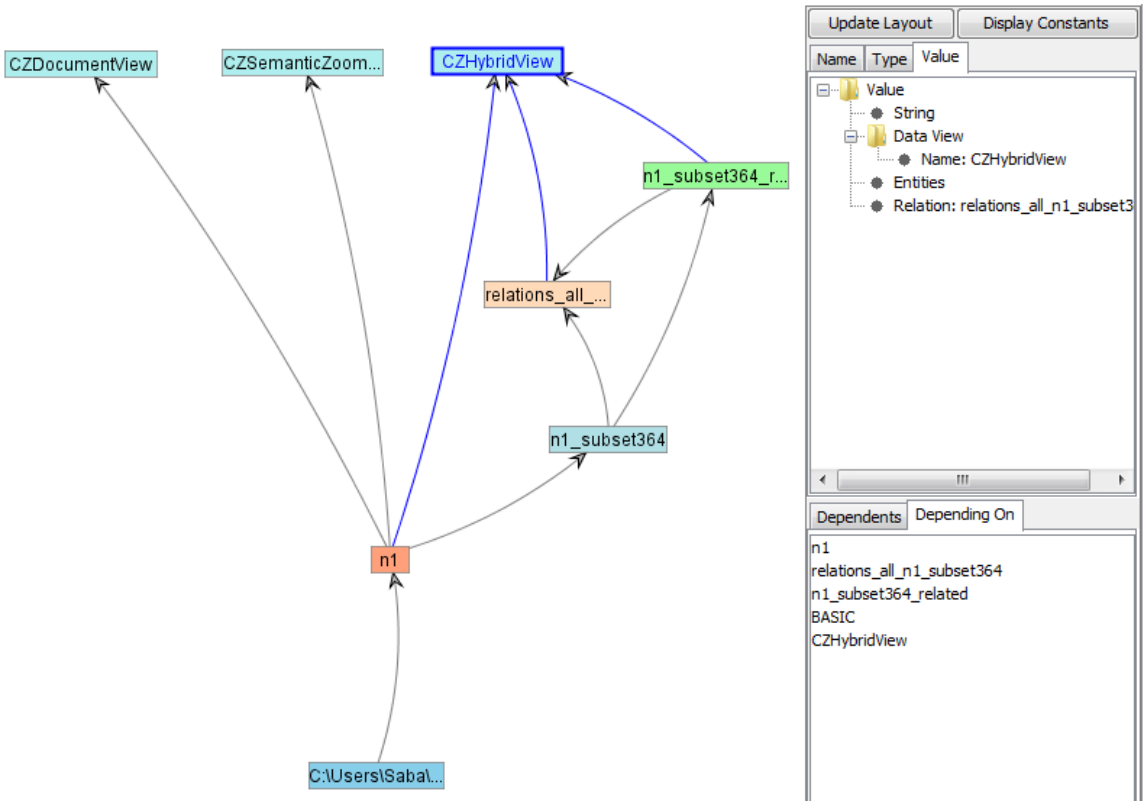


Figure 3.6: Displaying the information about a data view node.

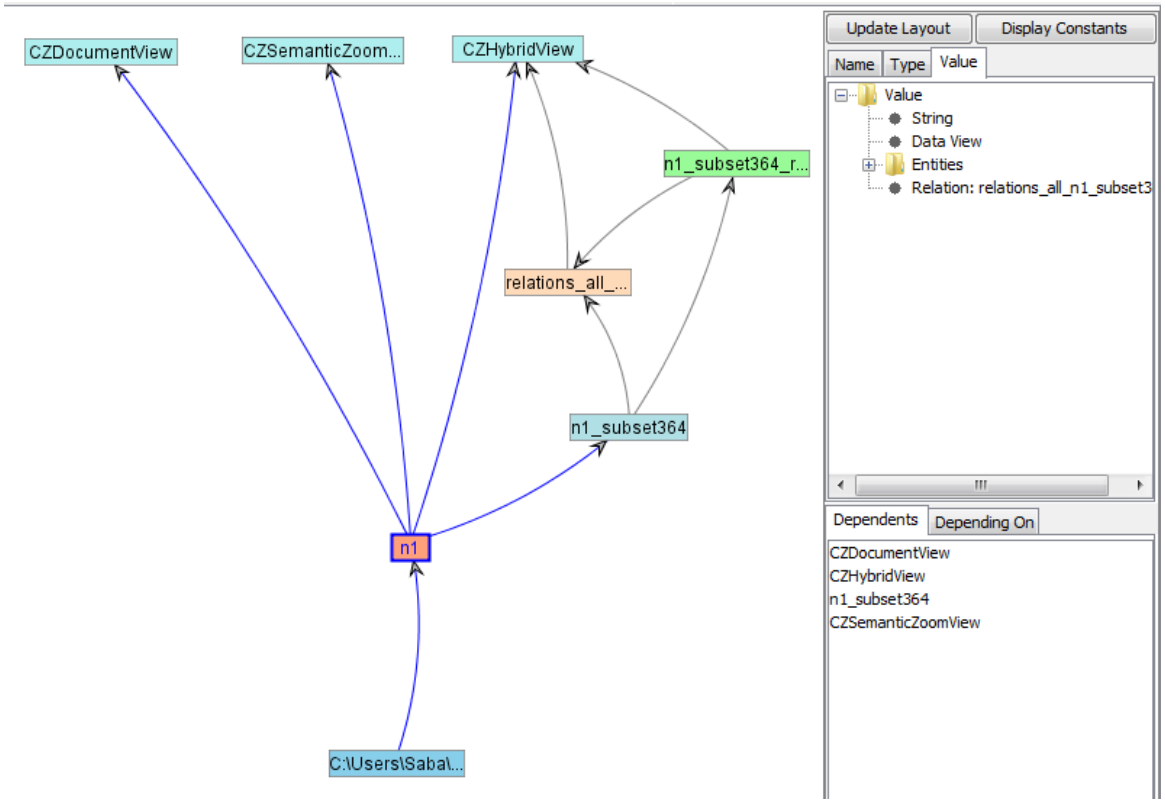


Figure 3.7: Displaying the nodes depending on the selected node.

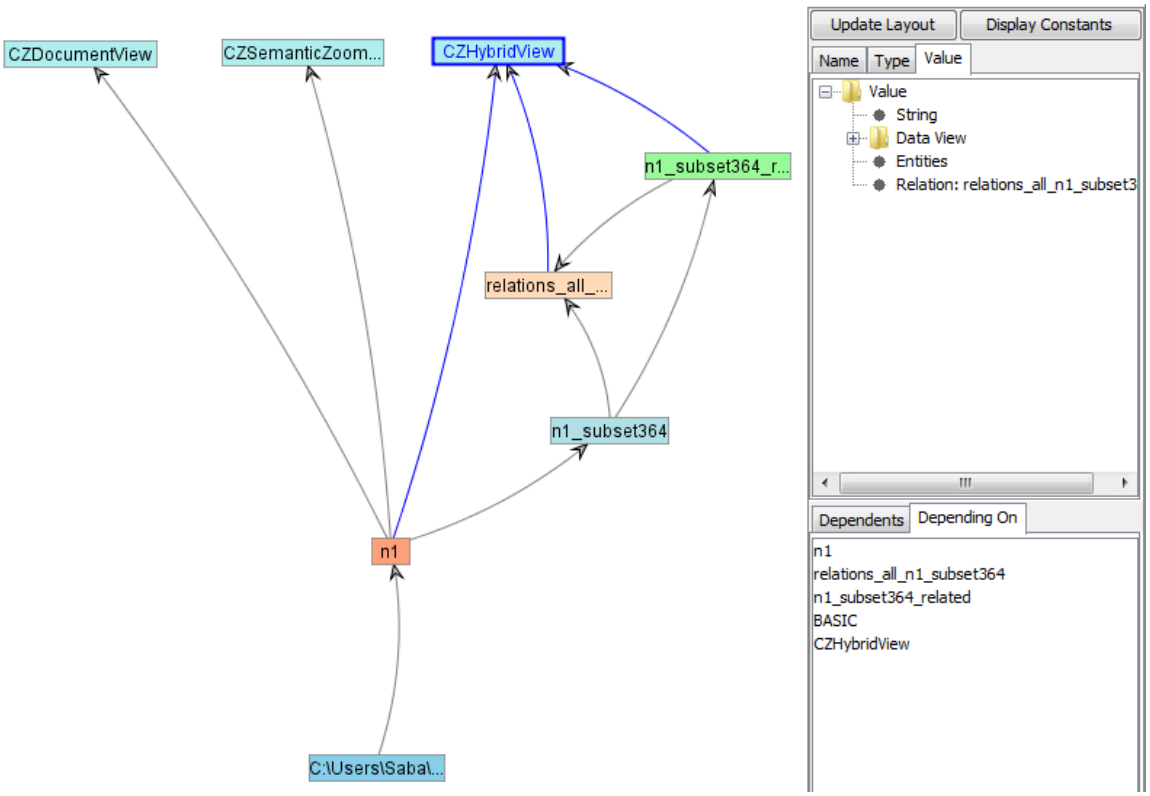


Figure 3.8: Displaying the nodes on which the selected node depends.

this decision is that if there are other nodes dependent upon node n, by removing n, those nodes become invalid since their value is calculated based on the value of n. If node n gets removed, its downstream nodes remain unchanged, since their value is not dependent upon n. However, if all child nodes of n are constants, they also get deleted by removal of n. This is because in many cases there is no longer any reasons for the existence of such nodes, as they are only parameters for evaluation of the value of n and have no other use. However, in future the graph should support removal of the parent node while preserving the now unconnected child nodes.

- **Node update.** One main feature of the dependency graph is the ability to update the nodes and propagating the update through the graph. In order to increase the degree of user interaction with the view, we wanted the analysts to be able to update some nodes directly from the view. The original way of updating the value of an entity in CZSaw is through searching for an existing *search entity* and assigning it a new search key. So the results with respect to the new key will be put into the old variable and it will be updated. So the update is currently only implemented for the constant parameter of a search result, which will update the search result and the whole graph accordingly. The option was provided for the analysts to update the search key, which is a string, through the dependency graph view. They are able to do so by clicking on the search node, which allows for the information to be displayed in the control panel. Then they could choose the search key in the panel and update it.
  Note: the node deletion and updating through the dependency graph view are yet not captured in the script and hence are not persistent and only exist in the current working session. In near future these changes will be captured in the script and become persistent.

Causality visualization

The goal of visualizing causality in the dependency graph was to allow the viewers to detect causal chains in the graph when a change gets propagated in the graph. The challenge is to provide a light-weight new dimension for the existing visualization. The dimension of causality must be integrated to the current graph, without damaging the data conveyed in the graph. In the case of CZSaw's dependency graph, the nodes' levels and parts of the graph's structure and layout are also contributing to the data presented to the analyst.

Hence, the method proposed for displaying the change propagation in the graph was to use the color and the width of the node outline. When a node gets updated, the outline gets highlighted yellow, and at the same time the outline gets thicker to emphasize the change. The node stays highlighted for a brief period of time, and then goes back to normal. Then the change gets propagated level by level to the dependent nodes, which will highlight and then pass the update. The reason that we only highlight the border as opposed to other existing systems that highlight the whole node, is that the node colors are set based on the types of the nodes and therefore convey information to the viewer. Hence, we do not change the color of the node itself. However, for helping the change in the color of the outline to be perceived easier, we increase the width of the outline temporarily at the time of highlighting it.

3.3 Dependency Graph Version 2

The implementation of the dependency graph was integrated into CZSaw successfully. The later updates were integrated into a new version of dependency graph. The dependency graph version 2 improves the first version in terms of functionality, performance and the architecture of the code. It provides a more detailed dependency structure that represents the dependency graph based on ports, which are more fine-grained elements than nodes. In the *"Elements of Parametric Desgin"* book [88], Woodbury introduces the concept of ports in the dependency graphs. He represents inputs and outputs of each node at a fine-grained level using ports. This allows the viewer to better trace the active dependencies in the graph and prevents many unnecessary updates in the graph as will be described later. It also allows for multiple outputs that was not supported before, and removes directed cycles that do not exist in the data but appear in the graph if the dependency is displayed only among the nodes. This version also introduces the notion of operations in the nodes as first-class citizens. This allows for future implementations to have multiple operations defined on one node, which supports parametric modeling better.

3.3.1 Architecture

Like dependency graph version 1, if we look at the dependency graph structure from a high-level point of view, we will see nodes and edges. Nodes contain the content (the data) and edges represent the dependency relations between the nodes.

The important classes inside the code are briefly described below, along with their most important fields and functionalities.

Node

Nodes generally contain the data in the graph. A node can have an operation, input node(s), and output node(s). The output nodes are the nodes that depend on this node. The input nodes on node n, however, are the nodes that n depends upon. A node can have none/some/all of these fields set. They can be set when the node is instantiated, or can be added later on in the analysis process.

Operation

Operation is a main part of each node. Each operation can have OperationInfo, input and output ports. OperationInfo contains some information about the current operation and can be used (or extended) by the user. Ports are value containers; input ports are the ports that the operation needs in order to perform. As a result of the operation being performed, it can affect the output port. In both cases, operation can use any subset of the input/output ports. Each operation that the user implements must extend the dependency graph's Operation class. Therefore the user is obligated to implement the evaluate() method. In this method, the user defines what this special type of operation actually does, setting the output port(s) using input port(s).

Port

This class mainly contains a value. The value is generic so the value itself and the appropriate operations could be set and defined based on user's needs. The ports are used as input/output for the operations. In the second version of the dependency graph, updates happen based on ports and not nodes. This improves the performance of the update propagation by only updating the nodes that are dependent upon the specific port of the updated node. So the fact that node $n2$ is dependent upon $n1$ is not enough to update $n2$ based on any change in $n1$, unless the change was on the specific port that contributes to $n2$.

Dependency Graph

The dependency graph handles nodes and the dependency relations between them. It has an update method that propagates the change through the whole graph based on an updated node; resulting in all relevant values to be updated. The update process first performs a topological sort for all the nodes in the graph based on the updated node. It then fires the update for each node that depends on the updated node, directly or indirectly, based on the sorted list. And the update happens based on ports being modified (not based on nodes alone).

Events

Every meaningful action that happens in the dependency graph (such as adding a node, adding an edge, or update) fires an appropriate event, to be used by another party that wants to use the dependency graph, e.g. a visualization, another software, etc. in the events package, there is a base class for all dependency graph costume events, and all dep graph events extend this base class. There is also a listener class for dependency graph events that catches them and performs an appropriate action. The users can change the action performed for each event based on their needs.

3.3.2 Improvements Over Version 1

Dependency graph version two's implementation has the a number of advantages in comparison with the initial implementation in the first version. Among the main improvements in the second version are the following:

Ports and Operators

In dependency graph version 2, the notion of ports and operators were introduced as first class citizens in the design.

Event Handlers (Separation of Layers)

The implementation of dependency graph version 2 follows the Model View Controller (MVC) design pattern. The view layer is completely separated from the controller layer that handles the underlying logic of the graph. This was done using the event handler

mechanism in Java programming language. In current implementation, an event gets fired for each important action that happens in the dependency graph, such as adding a new node, adding a dependency relation between two nodes, updating a node, etc. This approach provides full decoupling between the logic in the controller layer and the visualization in the view layer. Hence, different visualizations can be plugged in to the implementation of the dependency graph.

Test-Driven Development (Unit Testing)

During the implementation of the first version of the dependency graph, an important challenge was validation and verification of the graph, when parts of the code were changed or new functionalities were added. In this case, the new or updated parts of the implementation had to be tested manually. Validation and verification of the software is necessary after every update in the code in order to help with the process of checking whether the tool meats the specifications and fulfills its intended purpose. To address this challenge, test-driven development (TDD) approach was followed in the implementation of the second version of the dependency graph.

Test-Driven Development In the test-driven development process, when the developers want to write a new part of the code, first they write a test case for the code that contains assertions about the desired functionality of that part of the code, which are either true or false. After the production of the test case, they the developers produce the new part of the code, or improve an already-existing code. Then they check the code with the test cases to confirm that they pass the test and change the code if needed. In test-driven development, unit tests are often used to create the tests and automatically run them. If the tests are passed on the intended parts of the code, it confirms thet the evolved code has the correct behavior. To summarize, it the process of test-driven development, the developers usually follow a cycle of steps:

- Add a test: Before writing a new feature in the code, or modifying an existing feature, the developer first writes a test based on the definition and expected functionality of that part of the code. Needless to say, this test will initially fail since the related code has not been implemented yet.

- Run all tests and check if new test fails: To see whether the new test is indeed necessary and is working properly, all test cases are run after a new one is added. This step basically tests the test itself. If the test passes without a need to a change in the code, it shows that the test is not serving its purpose.
- Write a part of the code: In this step, the developer writes the code that causes the test case to pass. The code might not be perfect at this stage.
- Run all automated tests and check if they pass: If the code passes all tests, it shows that the new code meets the requirements.
- Refactor code: The developers can then clean up the code and remove duplications between the code used for testing and the final code that will be deployed. The programmers should run the tests again after refactoring, to confirm that refactoring has not affected the functionality of the code.
- Repeat: The cycle will be repeated for a new test. The size of the steps should be small, and if the tests fail, the changes should be reverted.

These steps are described in more details in the Test-Driven Development by Example [13].

JUnit In the implementation of dependency graph version 2, JUnit testing framework was used for supporting the unit testing. JUnit is an open-source project for the Java programming language, and is among the most common frameworks used in the process of test-driven development [3].

Chapter 4

Study

Visualizing causality is considered to be *"one of the top 10 unsolved information visualization problems"* [17]. The more recent techniques in this area get advantage of animation and motion for representing the causality. They aim to help the viewers understand the effects of a change in the graph by animating the cause and effect, and/or the path between the cause and the effect. Detecting the path of propagation of change in the graph can help the viewers understand the underlying structure of the system.
Among important methods for animating the causal paths are visual causal vectors (VCVs) [84]. VCVs use animation of an auxiliary object that moves from the cause of the change to the target, in order to help the users perceive the causal relations and paths.

The study was designed to compare a method inspired by a well-known VCV approach, called *oscillating nodes*, with two other visualization techniques, proposed by the current work. The first method is called *highlighting nodes*, and highlights the source of the change in the graph for a period of time, and then highlights the target node. If the target node acts as a source of change for other nodes, meaning that there are other nodes depending on the target, they also get highlighted when the change is propagated to them. The highlighting nodes method was proposed for CZSaw's dependency graph version one and is currently implemented and being used in CZSaw. To the best of our knowledge, this method, though simple, is not used in any other similar system. The second method, called *moving nodes*, was proposed for this study for the first time. In this approach, the node that acts as the source of the change at any time, moves from its original position towards the target of the changes. When it hits the target, the source starts getting back to its original position.

When the hit happens, the target starts moving and is the current source of change in the graph, and the process continues. The inspiration for this proposed method comes from the real world. If you have a number of balls and hit one of them, it will start moving and if it hits another ball on its way, the motion is transferred to the new ball, the target of the initial change, and it starts moving, and might hit other balls and trigger their movement as well. And the well-known oscillating nodes technique oscillates or vibrates the nodes in the graph that have received a change, whether the node is the initial source of the change, or it has been a target of another change, and can now act as the source of change for other nodes.

The goal of this study is to compare these three methods in term of performance, which is measured by the duration of task completion and the number of errors. The study also tries to find which visualization technique the users find more pleasant and easier to understand. It also aims to investigate the appropriate timing for perceiving causality based on previous research by Neufeld et al. [67], by asking the participants to rate the speed of the animation to see whether the previous results are confirmed by the new findings, and also to see if those finding stand for the new methods as well.

4.1 Methods

4.1.1 Participants

The participants were 12 undergraduate and graduate students in Simon Fraser University (SFU). Eleven of them were graduate students and one of them was an undergraduate student; nine of them were females and three of them were males. None of the participants had any familiarity with the visualization techniques being investigated. They were recruited through emails to department's mailing lists, personal contacts, and posters in common department areas. The participants were awarded $10 CAD after the completion of the study, as an incentive to take part in the study.

4.1.2 Design

The study had a within-participant design; one group of 12 graduate and undergraduate SFU students performed all three parts of the experiment. The reason behind having a

dependent group design was to reduce the effect of individual difference by reusing the same participants for all parts of the experiment. So the participants in a within-subject design perform all levels of the independent variable (visualization types).

The study was counterbalanced by the different types of visualization to avoid the carryover effects. To eliminate the effect of learning, the three types of visualization appeared in different orders for different participants. There is a combination of 6 possible orderings for three techniques, so each ordering was assigned randomly to two participants. The total of 12 participants covered all three combinations of the causality visualization techniques two times.

Participants were asked to compare the three different visualizations in term of their performance, which is measured in terms of time and accuracy of detecting the causal paths. They also ranked the visualization based on how pleasant they found them. Finally, they specified whether the speed of animation for each visualization was fast, slow, or moderate.

4.1.3 Variables

Independent variable (IV): The technique used for visualizing the causality with three levels of causality visualization techniques: *highlighting nodes*, *moving nodes*, and *oscillating nodes*.

Dependent Variables (DVs): The first group of the dependent variables were the duration of completion of one step and the number of errors. These DVs were derived from the information logged from the main body of experiment. The task duration data is continuous and the number of errors are discrete.

4.1.4 Procedure

The study consisted of three main parts. Each part implemented one of the three types of causality visualization being investigated in this study. There was a graph with around sixty nodes for each part, with paths of length one, two, or three. A path is a chain of causality, meaning that if node A is dependent upon node B, any change in node B will affect node A to change, and so the change should propagate from B to A. The paths in this experiment had a maximum of three levels. When an update or change happened on

a node, the change would then get propagated to the dependent nodes through the path, until it could not be propagated anymore, i.e. there was no other node depending on the updated node. The graph was topologically sorted based on the dependency relations to an extent, causing the directly dependent node to be one level higher than the original node in the graph layout.

Note: The directions of edges has been shown to be important on perception of causality [9], so in the design of this study the direction of propagation is always upwards. This eliminates the effect of direction in detection of causality, and helps the viewers focus on the type of the animation used in the visualization. Moreover, this layout matches the current implementation of the layout of CZSaw's dependency graph.

Each visualization contained 28 causal paths, 10 of them were of the length 1, 10 of them were of the length 2, and 8 paths had a length of 3. Firing an update event on the starting node of any of these paths caused the propagation of the update to the dependent nodes level by level. Firing an update on a node was visualized through the certain causality visualization technique being investigated in that section of the study. The participants were asked to click on the node being updated when they detected the update or change happening. They were asked to click on the updated node as soon as they noticed the animation on the node, without sacrificing the accuracy of the click. For the update to start for each one of the 28 paths, the participants had to click the next button on top of the screen, to declare their readiness for the start of this round. Then they had to click on the nodes getting updated. For going to the next round, they had to click next again. The same procedure applied to each of the 3 types of animation techniques for detecting the chain of propagation of change.

4.1.5 Hypotheses

Hypothesis 1 *The highlighting nodes causality visualization technique requires lower task completion time compared to both moving nodes and oscillating nodes methods. The reason behind this hypothesis is that the highlighting nodes method is simpler than the other two methods; it does not change the layout of the graph, fully preserves the viewer's mental map and puts less cognitive burden and the viewer.*

Hypothesis 2 *The moving nodes method requires lower task completion time in comparison with the oscillating nodes method. This assumption was made since the moving nodes method simulates the movement of objects (e.g. balls) and their collisions in real physical world. We hypothesized that this method requires less time and cognitive to be detected because of its similarity to real world movements and its familiarity for the viewer.*

Hypothesis 3 *The highlighting nodes method provides better task completion accuracy compared to moving nodes and oscillating nodes method. This hypothesis is justified according to the simplicity of the highlighting nodes method as described in hypothesis 1.*

Hypothesis 4 *The moving nodes method provides better task completion accuracy than the oscillating nodes method. The moving nodes method was inspired by real movements and propagation of kinetic energy in real world and we assumed that it would be easier for viewers to detect (refer to the description for hypothesis 2).*

4.1.6 Data Collection

For each type of visualization there was a log file kept of all important user actions and the relevant time stamps. Whenever the users pressed the next button to start the next round, a line was added to the log file declaring the start of the next round and the relevant time that it happened based on the system's clock. Also, the number of the steps that the participant had to complete complete was logged. The number of steps for each round is equal to the number of clicks the participants had to have on updated nodes in this specific round, which is equal to the number of nodes in that causal path. Then with each user click on an updated node, a line was added to the log file. This line contains the time of the click, the x and y position of the node that was being updated and the participant had to detect it and click on it, and the x and y position of the mouse on the screen when the click happened. This information was logged for every round, in each type of the three visualizations. Each visualization technique had its own log file. The log files were completely anonymous and did not contain any personal information from the participant. They were collected after the participant finished the study.

Post Questionnaire After finishing the study, the participants were asked to fill a short questionnaire, which comprised of two parts. The first part asked the participant to compare each pair of visualizations and declare the one that they preferred and found more pleasant. The second part of the questionnaire asked the users to indicate whether they found the speed of animation slow, fast, or appropriate, for each animation technique.

The goal of this study was to investigate whether the new animation techniques for visualization the causality were significantly better than a well-known and widely-used method in terms of speed and accuracy of detection of the causal path by the viewer. The participants' preferences for the methods were also be compared. In addition, the speed of animation for each type of visualization was based on previous work on the area [67]. The participants' feedback on timing was assessed to either confirm the previous findings, or reject them and propose new timings for this type of animation.

The information gathered in the post questionnaire along with the log files from the study were stored in the format of text documents. The text files were put in a password protected folder in the investigator's flash memory stick, to which only the principal investigator had access, and was locked in a secure cabinet to be retained for the next two years. After transferring the data to the memory stick, the original data was removed from the lab computer on which the study was conducted.

The risks involved in this study were minimal and were not more than as that one would expect the participants to encounter in their every day life. The confidentiality and anonymity of the participants was guaranteed through the whole study. There was no name or any sort of identifier related to each individual stored any where in the study. The log files contain generic names such as *participant1results.txt* and contain no personal information about the participant.

4.2 Prototype

A prototype was designed for the study that implements these three approaches using Java programming language in Processing development environment.

Processing is an open source environment and programming language that helps Java developers in graphics implementation, animation and interactions, by providing various

libraries [4].

4.2.1 Architecture

The prototype was developed separate from the main dependency graph in the CZSaw project. The goal of this prototype was to find the best causality visualization technique to be used in CZSaw's dependency graph in next generations of development. Though the results of this study are not limited to CZSaw in any way and can be used in any propagation system that focuses on causality.

The prototype modeled the three types of visualization discussed: the highlighting nodes, moving nodes, and oscillating nodes. Each visualization method was used for a separate graph. Three graphs were used for the study in total. The structures and layouts of all graphs were kept very similar. Each graph had around sixty nodes distributed among a maximum of four levels. Twenty eight causal paths were studied in each graph; ten paths with the length of one, ten with the length of two, and eight paths with the length of three. For each visualization, there were 28 rounds. In each round, one of the the 28 paths were updated and visualized in a random order on the related graph. The participants could start each round when they were ready by pressing the *next* button. Figures 4.1, 4.2 and 4.3 show screenshots of the prototype displaying the three graphs used for the three visualizations respectively.

The design of the prototype contains various classes and methods that could fit into following main categories.

- **Nodes:** The class *Node* represents the nodes in the graph that are displayed for each type of visualization. The methods that use the attributes of this class, such as drawing and moving methods, could be accessed from different threads as will be described in the next item, hence they are declared as synchronized.

- **Node animations:** The class *NodeAnimation* represents all the different types of animation for visualizing the causality for one node. The *NodeAnimation* class is extended by three classes for each type of visualization: *HighlightingNode*, *MovingNode* and *OscillatingNode*. This class *NodeAnimation* itself extends the class Thread.

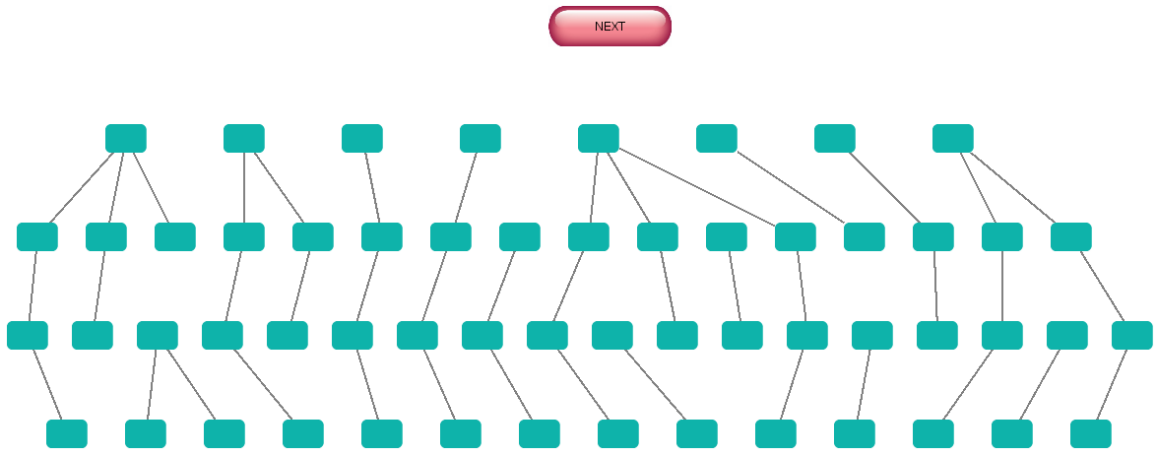


Figure 4.1: A screenshot of the prototype for visualizing causality.

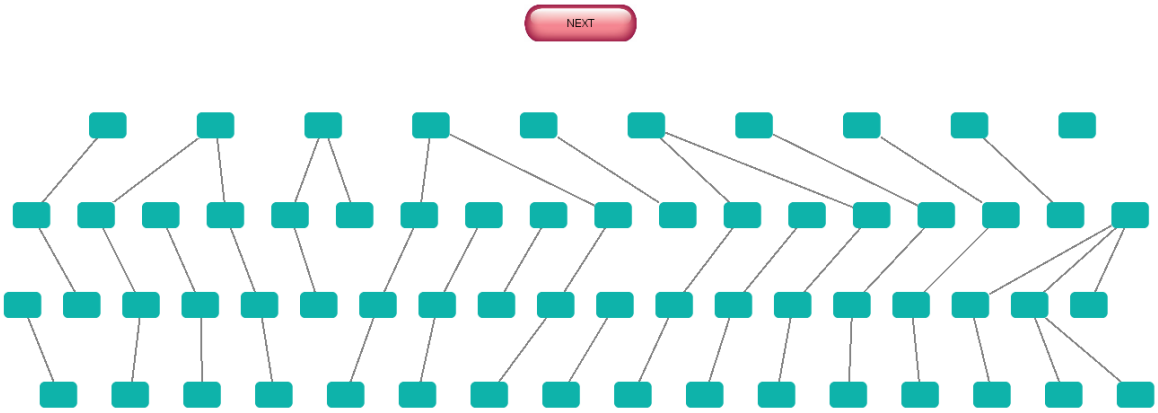


Figure 4.2: A screenshot of the prototype for visualizing causality.

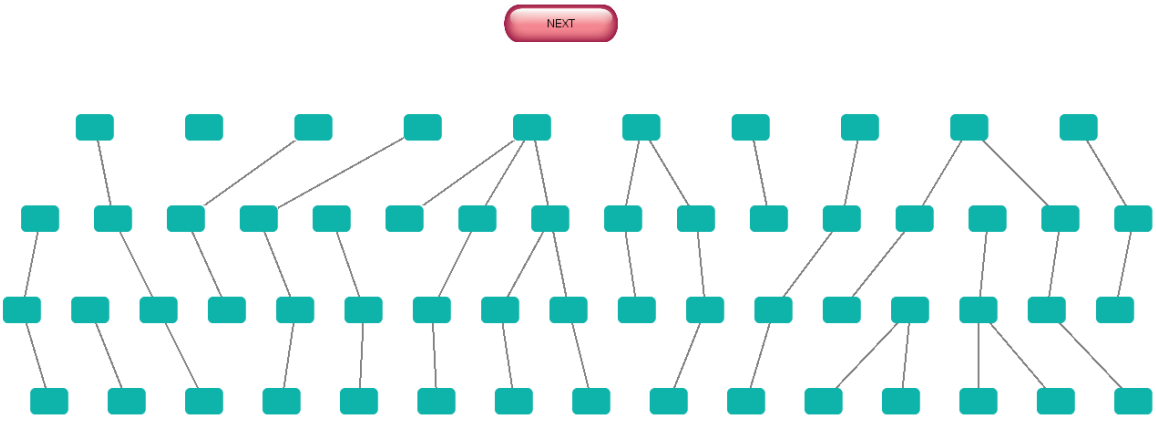


Figure 4.3: A screenshot of the prototype for visualizing causality.

Each class that extends the Thread class much implement the *run* method. The run method of each of the animation type classes implements the method that node uses for visualizing an update on the related node. Each updating node runs in its own thread and therefor concurrent updates can happen on different nodes of the graph. The fact that multiple threads could access the same node and use and change its attributes, is the reason behind having the *Node* class's methods synchronized.

- **Experiments:** In the implementation of the prototype, there is a class dedicated to each part of the study for each type of the visualization. So there are three classes for this means that all extend a class called *Experiment*, which itself extends *Thread*. The three classes that extend the *Experiment* class are *Experiment1*, *Experiment2* and *Experiment3*. Each of these classes are responsible for initializing the graph of the experiment, and running the study of all candidate causal paths that should be displayed and detected. In the run method of each experiment, the nodes of each path get updated one by one, with a delay between each two updates based on previous research in the literature by Neufeld et al. [67]. A flag in this method controls whether the next round should be executed or not, and this flag is activated when the participant presses the next button and starts the round, and is deactivated when the participant has detected the updated nodes in the path and the round is finished. Then the participant can press the next button again when he or she is ready to proceed to the next round. This approach is used for each of the three experiments for the three causality visualization methods for the experiment's propagation graph.

- **Control:** In the main thread of execution, the three experiments are initialized and kept in a list, though only one of them can be active at a time. In the main loop, the updated nodes and edges of the graph of the active graph are continuously drawn to represent the possible change in the graph. The mouse events are handled in a separate method. The valid mouse clicks are either on the next button to start the next round of update on a causal path, or to detect the updated node in the current path.

- **Logging:** All the information that needed to be analyzed later are kept in text log files. The prototype takes care of logging the information regarding user clicks that contribute to the process, which are the user clicks on the next button and the clicks on the updated nodes in each path, along with their relative timings. For each time

that the user presses the next button, a time stamp is logged, since it shows the start of a new round of update in a causal path. The next line of log file is the number of nodes involved in the current path. If there are n nodes in the current path being updated, there will be n lines in the log file for those nodes. Each line relates to a click on a certain node in the path, and contains the time stamp of the click. It also contains the x and y position of the mouse when the click happened, as well as the x and y of the clicked node at the time that the click happened, since in two of the visualizations the node are able to move and could be in a different position than their original one when the click happens. These information are logged to be analyzed later in terms of duration of task completion derived from the recorded time stamps, and in terms of the number of errors based on the positions that were stored. If a participant proceeds to the next round of the experiment without performing one or more of the required clicks, the related line(s) will not be added to the log file, which will be detected later during the analysis as an error.

Chapter 5

Results and Discussion

5.1 Analyzing Data

5.1.1 Parser

The resulting log files of the study contained the raw information as discussed in the previous chapter, to allow for different types of analysis in future. In order to be able to gain insight through this raw data and derive conclusions, a parser was written for the log files of the study.

Input Files

The input files of the parser are the log files resulting from the study. After the completion of the study by each participant, three log files are generated for each part of the experiment. These files log important user actions, i.e. clicks on the next button and on updated nodes, as well as their relevant time stamps. A sample of a few entries to a log file is shown in figure 5.1.

Output File

The parser reads and parses each of the log files, and generates an output file containing the relevant information derived from the raw data. The log files will mainly include the items for timing and errors as as discussed below.

```
next @: 20172
Step: 1 , numOfNodes: 3
Node num: 0 in level, time: 22914, mouseX: 70, nodeX: 50.0, mouseY: 366, nodeY: 350.0
Node num: 1 in level, time: 24215, mouseX: 73, nodeX: 60.0, mouseY: 262, nodeY: 250.0
Node num: 2 in level, time: 25176, mouseX: 162, nodeX: 150.0, mouseY: 164, nodeY: 150.0
--------------
next @: 28197
Step: 2 , numOfNodes: 4
Node num: 0 in level, time: 30444, mouseX: 124, nodeX: 90.0, mouseY: 454, nodeY: 450.0
Node num: 1 in level, time: 31218, mouseX: 64, nodeX: 50.0, mouseY: 356, nodeY: 350.0
Node num: 2 in level, time: 31885, mouseX: 91, nodeX: 60.0, mouseY: 254, nodeY: 250.0
Node num: 3 in level, time: 32799, mouseX: 193, nodeX: 150.0, mouseY: 155, nodeY: 150.0
--------------
next @: 35324
Step: 3 , numOfNodes: 2
Node num: 0 in level, time: 37757, mouseX: 747, nodeX: 730.0, mouseY: 464, nodeY: 450.0
Node num: 1 in level, time: 39151, mouseX: 663, nodeX: 644.0, mouseY: 358, nodeY: 350.0
--------------
next @: 40406
Step: 4 , numOfNodes: 3
Node num: 0 in level, time: 43210, mouseX: 734, nodeX: 710.0, mouseY: 364, nodeY: 350.0
Node num: 1 in level, time: 44419, mouseX: 714, nodeX: 690.0, mouseY: 263, nodeY: 250.0
Node num: 2 in level, time: 45348, mouseX: 649, nodeX: 630.0, mouseY: 156, nodeY: 150.0
--------------
next @: 46154
Step: 5 , numOfNodes: 2
Node num: 0 in level, time: 48571, mouseX: 266, nodeX: 250.0, mouseY: 462, nodeY: 450.0
Node num: 1 in level, time: 49671, mouseX: 201, nodeX: 182.0, mouseY: 356, nodeY: 350.0
--------------
next @: 51096
Step: 6 , numOfNodes: 4
Node num: 0 in level, time: 54226, mouseX: 510, nodeX: 490.0, mouseY: 465, nodeY: 450.0
Node num: 1 in level, time: 54954, mouseX: 461, nodeX: 446.0, mouseY: 352, nodeY: 350.0
Node num: 2 in level, time: 56457, mouseX: 508, nodeX: 480.0, mouseY: 263, nodeY: 250.0
Node num: 3 in level, time: 57293, mouseX: 523, nodeX: 510.0, mouseY: 156, nodeY: 150.0
--------------
```

Figure 5.1: A sample of a few entries to a log file in the study

Timing The timings for each step of each round of the three parts of the experiment was calculated. According to time stamps collected in the log files, time duration of completion of each step in each round is the time of the completion of the step (clicking the current updated node) minus the time of starting the step (clicking the previous updated node in the round).

$$d(r_i, s_j) = t(r_i, s_j) - t(r_i, s_{j-1}) \tag{5.1}$$

In the above equation, $d(ri, sj)$ is the duration of step j in round i of each experiment. $t(r_i, s_i)$ is the time stamp in the log file recorded for clicking the updated node at the jth step of round i, while $t(r_i, s_{j-1})$ is the time stamp of the click on the previous updated node in the round.

In the case that the current updated node is the first node in the current round and there is no previously updated node in the round, the time will be calculated according to this formula:

$$d(r_i, s_j) = t(r_i, s_j) - t(r_i) - 500ms \tag{5.2}$$

Where $d(r_i, s_i)$ is the duration of step j in round i of each experiment. $t(r_i, s_i)$ is the time stamp in the log file recorded for clicking the updated node at the jth step of round i. $t(r_i)$ is the time of starting the ith round of each experiment, which is recorded in the log file when the participants press the *next* button in the study and start the next round. 500 milliseconds is deducted from the time duration of the first step of each round, since there is a gap of 500 ms between the time that the participant presses the *next* button and the time that the first update fires. This wait time is enforced by the prototype at the beginning of each round, to let the users focus on the graph again after having to press the *next* button, which is outside the layout of the graph.

Errors Another important factor that needs to be investigated and is the number of errors. When the participants do not click on the area of the latest updated node an error occurs. It is considered an error when participants perceive the update wrong and click on another area of the screen, or miss the update and go the next round without clicking on the updated node.

If the participant has performed a click for the updated that happened in a node, the log file will contain a line dedicated to that click, with the time stamp of the click. This line will contain the number of the click in the current round. It will also contain the x and y position of the mouse, as well as the x and y position of the node when the clicked happened, since in two of the visualization techniques the nodes are able to move and might not be in their original position when the participant detects the update and clicks. In the case that the user has performed a click, the position of the click is compared with the position of the node at the time of the click. If the click was not on the area of the node, an error has happened. In order to help the participants in the case that the nodes are moving, a horizontal boundary of 10 pixels around the node was considered a valid click.

It will also be considered an error if there is no record of a click occurring for an updated node in a round. Since the number of steps in each round are also kept in the initial log files, the parser is able to detect whether the required number of clicks have been registered for each round or not.

5.2 Results

5.2.1 Detection Time

An important point when analyzing the results for measured timings, is the difference in the positions and distances between the items that get clicked by the participants. One goal of this study is to investigate how different types of causality visualization affect the detection of change in a dependency graph in terms of task duration. But in order to be able to investigate the time needed for perceiving a change and clicking the changed item, we need to eliminate the time needed for moving the mouse from the source element to the target element, especially when the positions of the elements vary. That is why we used the Fitts' law to predict the movement time and eliminate it from the total task completion time.

Fitts' Law

Fitts' law aims at accurately modelling human movement time based on a highly successful relation [27]. It allows the prediction of movement time in human computer interaction. A movement model built based on Fitts' law is an equation that predicts movement time (MT) from the index of difficulty (ID) of a task (equation 5.3). In general [60] [75]:

$$MT = a + b \times ID \tag{5.3}$$

Equations 5.3, 5.4 and 5.6 are taken from [60] and [75].

There are a few proposed formulations for calculating ID. Shannon's formulation is used in this work because:

- It fits the observations slightly better
- It exactly follows Fitts' law's underlying information theorem
- It always gives a positive index of difficulty

The Shannon formulation for index of difficulty (ID) is:

$$ID = log_2(\frac{A}{W} + 1) \tag{5.4}$$

where A is the distance or amplitude of the movement and W is the width or tolerance of the destination area of the movement. Since task difficulty represents the amount of information, the speed of completion of movement is related to the processing rate of the information by the user performing the task. Card et al. [77] were the first ones to apply Fitts' law to the mouse in a study. The law has been applied to many selection tasks since.

The movement time (MT) predicted by Fitts' law can be deducted from the total task duration time (TT) measured in the study in order to give the time spent for detecting the change in the graph (detection time (DT)).

$$DT = TT - MT \tag{5.5}$$

Detection time can then be used to compare different types of causality visualization in the study.

Application of Fitts' Law to the Study In order to predict the necessary movement time for each step of each level in each of the three visualization types, first the index of difficulty was calculated. The width (W) of all nodes were the same since all nodes had the same size. The distance or amplitude (A) was calculated based on the positions of the start of the movement and the target. For the first node in each level, the distance between the center of the "next" button and the click were calculated. Each level starts by clicking the "next" button and then the participant proceeds to clicking on the updated nodes. For the rest of the updated nodes in the same level, the distance between the previous click (the click on the previous node on the chain of causality) and the current click were calculated as A. Having W and A, ID could be computed based on the Shannon formulation.

Computation of Movement Time by Fitts' Law For building a Fitts' law model for the current study, equation 5.6 was used.

$$MT = 230 + 166 \times ID \tag{5.6}$$

This equation defines the a and b constants for equation 5.3 for selection tasks and is shown to have a good fit for pointing task based on the work of Mackenzie et. al in 1991 [62]. ID was defined in equation 5.4. A is the amplitude or the distance between the starting point and the target. For each click on an updated node $n1$, A is calculated based on the distance of the click on $n1$ from the previous click. If the previous click was on another node $n2$, the distance between the two clicks on $n1$ and $n2$ is calculated as A. If the previous click was on the *next* button, meaning that $n1$ is the first node on the causal chain of this level, A is defined as the distance of the user's click on $n1$ from the center of the *next* button.
Another factor in computation of ID is W, or the width of target element. In the case of this study, all nodes have the same size, and thus W is the same for all nodes. If the nodes were circle or square shaped, there would only be one possible option for W. But since the nodes in our prototype are rectangles and the approach angle varies, a different model for interpreting the target width can be considered for choosing the proper W.
1. STATUS QUO. This model considers the horizontal extend of the target as W.
2. W'. This model better represents the 2D nature of the task. It calculates the width based on the degree of the movement path.
3. SMALLER-OF. The third model considers the smaller of width or height. The smaller

of two dimensions appears to be more consistent with the difficulty of the selection task. MacKenzie and Buxton [61] compare these models along with two other models using the index of difficulty formulation by Shannon. The SMALLER-OF model was shown to perform better. However, the W' model's results were close to those for the SMALLER-OF model. The SMALLER-OF model was selected in the computation of ID in the current study.

5.2.2 Timing

The independent variable was the animation type, which is a categorical variable that can take three values. Hence the statistical method for analyzing the gathered data was ANOVA. ANOVA is a statistical test that determines whether the means of two or more groups are statistically different. The independent variable is the animation type, and the dependent variable is the task duration time needed for detecting the change based on above discussions. The task duration is a numeric continuous variable, which is calculated by decreasing the movement time by Fitts' law from the total measured time (equation 5.5). The tool used for all statistical test was JMP [1].

Test Results and Discussions

The mean of detection time for highlighting nodes was 749.70 and the standard deviation was 36.45. For oscillating nodes, the mean was 1019.98 and the standard deviation was 37.74. For moving nodes method, the mean and standard deviation were 951.34 and 36.30 (Table 5.1). A one-way ANOVA test was used on the results of timing for each step of each animation type. The results were signifiant among the three types of causality visualization ($F(2, 2539) = 14.528, P < .0001$). The results are shown in Figure 5.2. These results confirm the hypothesis 1 that highlighting nodes method performs better than both other methods in terms of timing. The reason is that highlighting nodes uses the color and the width of the borders of the nodes to display the change and completely preserves the viewers' mental map by not changing the layout of the graph. Hence, this method requires less cognitive effort from the participant. Moreover, the results showed that the detection time for moving nodes is less than oscillating nodes, which also confirms our hypothesis. We had predicted that among these two methods that change the positions of the nodes, moving nodes simulates the real world movement and collision of objects (e.g. balls), and thus is easier to detect.

Animation Technique	Mean	Std Error
Highlighting nodes	749.70	36.45
Oscillating nodes	1019.98	37.74
Moving nodes	951.34	36.30

Table 5.1: Means and standard deviations for change detection time in each type of causality visualization technique

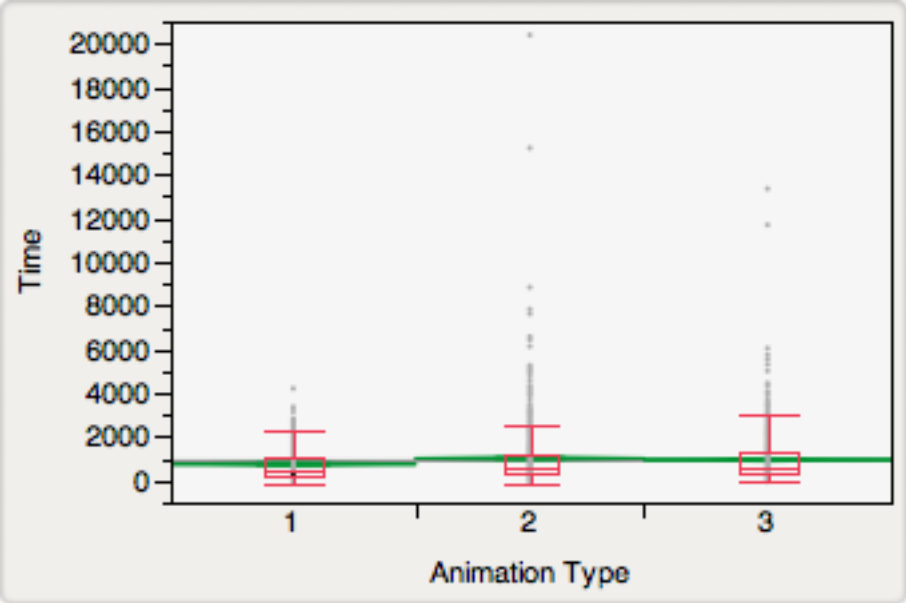


Figure 5.2: One-way ANOVA analysis of timing by animation type. 1: highlighting nodes, 2: oscillating nodes, 3: moving nodes

Animation Technique	Mean	Std Error
Highlighting nodes	1382.28	71.29
Oscillating nodes	1975.55	71.51
Moving nodes	1734.88	71.40

Table 5.2: Means and standard deviations for change detection time of first click in each level in each type of causality visualization technique

In order to further investigate the data, another one-way ANOVA was used on only the first steps of each level. The reason behind this decision is that the static links between the nodes in each causal chain, represent the causal relations between the nodes. The animated and dynamic techniques provide a stronger representation of causality compared with the static technique (graph's dependency edges). Once the viewer has detected the change in the first node of a causal chain, he or she can follow the static links and detect all the dependent nodes that get updated because of the first node. Hence, the role of detection of change in the first node of each causal path is crucial. It is the first node of each path that represents the whole path and all the consequent updates in the path can be derived having the first updated node. The dependent nodes can then be detected much easier and quicker, because of the existence of the static links as well as their spatial proximity to the previous node. This is the reason behind the second ANOVA test, that only considers the duration of first step of each level (causal path). This test investigates the effects of causality visualization technique on the completion time of detecting the first node of each causal path. The means and standard deviations for the records are summarized in Table 5.2. The results of the ANOVA test were significant ($F(2, 969) = 17.47, P < .0001$). The findings confirm our hypotheses 1 and 2 that the required time for change detection using highlighting nodes < moving nodes < oscillating nodes. The significance of the results for first clicks is stronger, since the differences between the means is much higher than previous test. The results are shown in Figure 5.3.

In addition to the task duration, the accuracy of task completion equally contributes to the user performance. Comparisons based solely on task completion time can be challenging if error rate behaves oddly. For example, if a condition is faster than another one, but has more errors, it is not always clear which condition is better. The error rates in different animation types are discussed in details in the next section. However, in the current section, to further improve our analysis, some filtering was applied to the data used for the previous

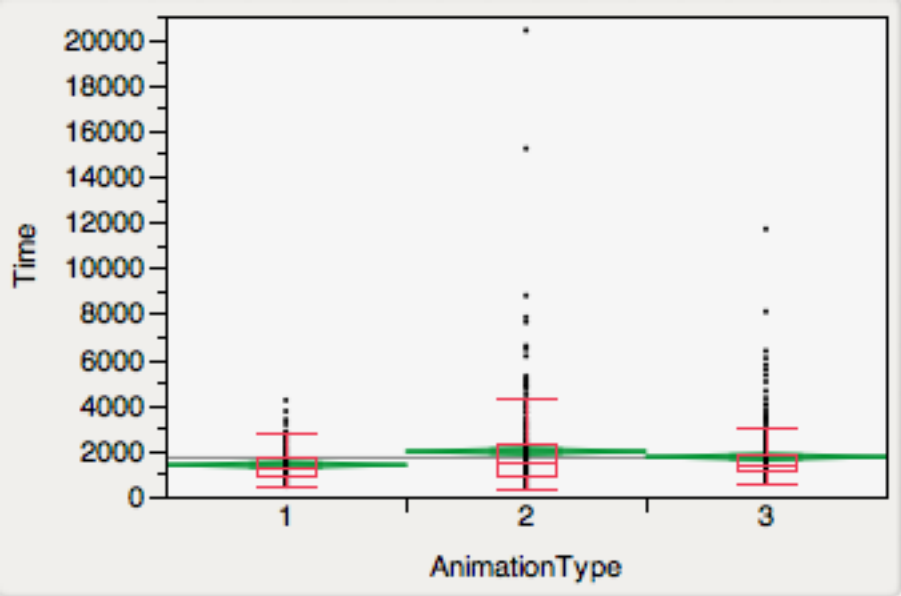


Figure 5.3: One-way ANOVA analysis of timing by animation type on first nodes of each level in each visualization type. 1: highlighting nodes, 2: oscillating nodes, 3: moving nodes

test. For the results of each animation type for each participant, if the number of errors made by the participant exceeded a threshold, the data was eliminated as outlier. Making errors more than a limit shows a lack of understanding of the visualization technique or a problem in performing the experiment and hence the data is potentially invalid. Seven records of different animation type timings were eliminated this way. These records belonged to five different participants. The excluded records had thirteen or more errors.

The means and standard deviations of the filtered data is summarized in Table 5.3. The results of the ANOVA test on update data are still significant ($F(2, 781) = 18.525, P < .0001$). But the difference is, after removing the records for cases with high error rates, the detection time for moving nodes gets significantly higher than oscillating nodes. A way to describe this difference with previous test's results is to assume that oscillating nodes technique puts more cognitive burden on the viewer. Thus the more complex tasks that are more error prone, take more time to complete as well. So by removing the complex tasks, the advantage of moving nodes method is taken away. The results of the ANOVA test of the clicks on the first nodes in level including the filtering is shown in figure 5.4.

Animation Technique	Mean	Std Error
Highlighting nodes	1284.32	59.57
Oscillating nodes	1610.90	65.97
Moving nodes	1816.43	65.97

Table 5.3: Means and standard deviations for change detection time of first click in each level in each type of causality visualization technique, after removing records with high error rate

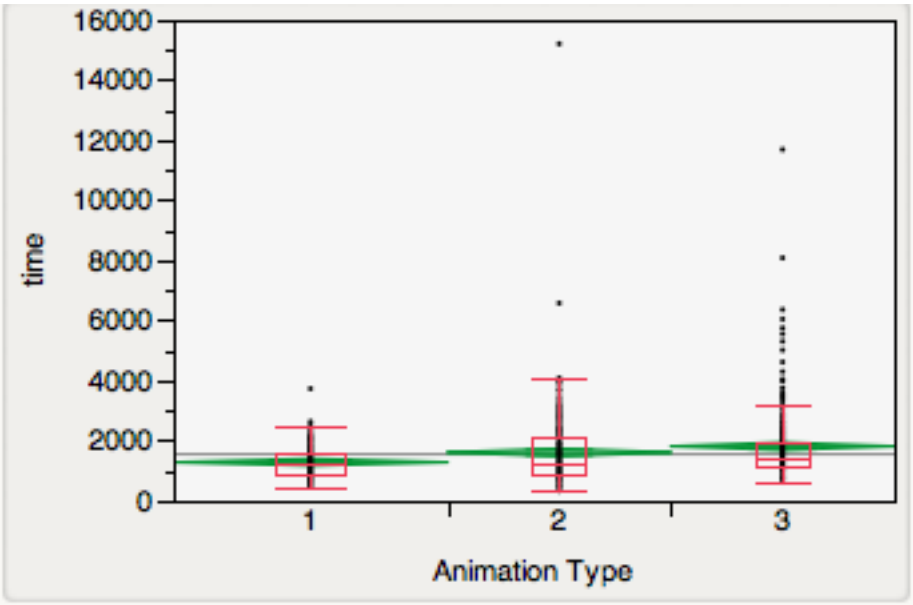


Figure 5.4: One-way ANOVA analysis of timing by animation type on first nodes of each level in each visualization type, after removing records with high error rate. 1: highlighting nodes, 2: oscillating nodes, 3: moving nodes

Animation Technique	Mean	Std Error
Highlighting nodes	7.09	0.62
Oscillating nodes	8.00	0.69
Moving nodes	7.44	0.69

Table 5.4: Means and standard deviations for number of errors made in each type of causality visualization technique

5.2.3 Accuracy

The tradeoff between the speed of actions and the accuracy of responses has always been a challenge. Many researchers focus mainly on task completion time measurements for analyzing the performance. However, the accuracy of task completion also contributes equally to the performance. In the case of this study, the number of errors that each user made was also logged for each animation type. An ANOVA test was ran to investigate the effects of animation on the accuracy of detecting the causal chains based on the number of errors. Before conducting the test, outliers were excluded from the data. A total of two records for different animation types from five participants were eliminated. Outliers were defined as the logs for each animation type where the participant had made 13 or more errors, coveting that the participant did not fully understand the visualization technique or the study procedure. These issues can be improved with more pre-study training in future studies.

Test Results and Discussions

The mean of measured errors for participants using the highlighting nodes method for visualizing causality was 7.09. The mean error for the oscillating nodes method was 8.00, and the mean for moving nodes method was 7.44. The results are summarized in Table 5.4. The results of running ANOVA on the data is shown in figure 5.5. There was no significant difference among three types of causality visualization, $F(2, 26) = 0.472, P = 0.628$. Thus, hypotheses 3 and 4 are rejected. However, by looking at the data and the means, it seems that highlighting nodes method had the least errors. This could be because of the simplicity of this method that requires the least cognitive load and does not change the position of any element in the original graph. Also, the number of errors for the moving nodes method appears to be slightly better than the oscillating nodes method. As mentioned above, the results were not significant, but according to the minor differences in error rates, perhaps

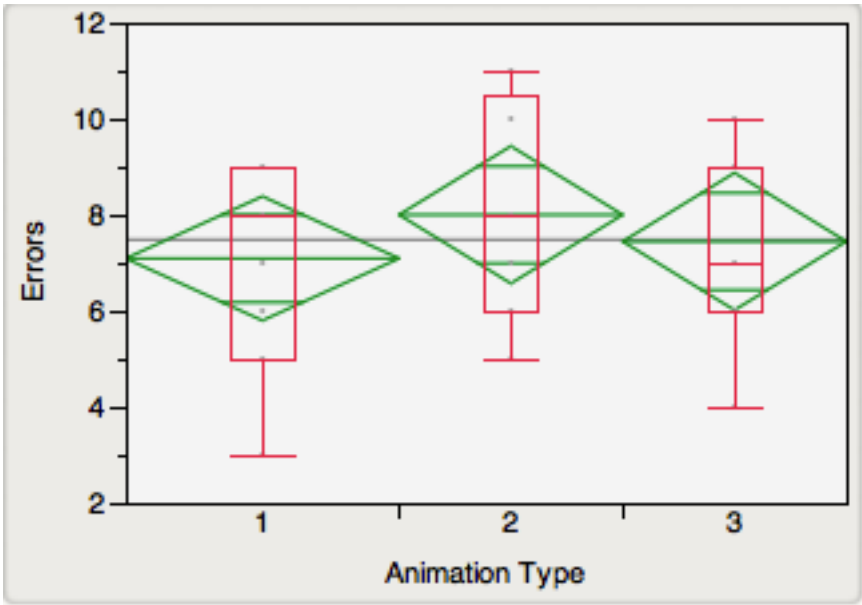


Figure 5.5: One-way ANOVA analysis of errors by animation type. 1: highlighting nodes, 2: oscillating nodes, 3: moving nodes

better results can be acquired by designing more complex graphs and more difficult tasks.

5.2.4 Post Questionnaire

After completing the experiment, the participants were asked to complete a post-questionnaire form. This questionnaire was consisted of two parts.

1) The first question asked the participants to compare each two types of visualization based on their preference of the technique and how pleasant they found each method.

2) The second question asked the participants to rate the speed of each animation and indicate whether the speeds were slow, fast or appropriate.

Only one of the participants failed to fill the questionnaire. The results of the answers of the eleven other participants to these questions are mentioned in the following sections.

User Preference

Highlighting nodes vs. oscillating nodes. Almost all of the participants preferred the highlighting nodes method over oscillating nodes (ten participants preferred highlighting nodes and only one preferred oscillating nodes). This is consistent with our original assumption that the highlighting nodes method will be easier to use because of its simplicity and low cognitive burden. This allows the viewers to preserve their mental maps of the graph and follow the changes more easily.

Highlighting nodes vs. moving nodes. Again, almost all of the participants preferred the highlighting nodes method. Nine participants chose highlighting nodes as their preferred method, one participant selected moving nodes, and one participant favoured both methods equally. The results are again consistent with our initial theories as mentioned in the previous item.

Oscillating nodes vs. moving nodes. We had initially predicted that the moving nodes method would be preferred by most since it is similar to real world physics of the movement and collision of objects such as balls. The results of the questionnaire confirmed our prediction: most of the participants (seven people) preferred moving nodes method which only four of them preferred the oscillating nodes method.

Animation Speed

This part of the questionnaire was designed to investigated the timing of the animations, which was based on previous research. The results confirm that the timings were suitable for the animations overall. For the highlighting nodes method, nine participants indicated that the speed was appropriate, and only two of them thought that the animation speed was fast.

For the oscillating nodes method, five of the participants thought of the speed as appropriate, four of them thought that the animation was slow, and only one participant claimed that it was fast. One other participant mentioned that the animation was subtle. The oscillating nodes animation method consists of smaller animation units, which are the small steps in which the node moves horizontally and a combination of these steps forms the animation for the node. One solution could be to increase the number of the small steps,

which leads to an increase in the domain of the movement. As a result, the whole animation seems to happen faster since there are more steps happening in the same amount of time. Also, extending the movement domain helps the concern of animation being "subtle" for some viewers.

For the moving nodes method, most of the participants were satisfied with the animation speed. 9 participants declared that the speed was appropriate, while only one participant decided that the animation was slow and one other stated that it was fast. So overall, the estimated speed for the animations seems appropriate for all three investigated visualization techniques and can be used in other causality visualization methods as well.

5.3 Summary and Conclusion

In this document, first an introduction of the general fields and concepts used throughout the document was given. The field of visual analytics was briefly described and the CZSaw visual analytics tool was introduced. Then the concepts and functionalities of a dependency graph was described, a main structure within CZSaw, developed during the course of this project. In the area of visualizing the dependency graph, one major challenge is the visualization of causality when an update happens in the graph and gets propagated to all directly or indirectly dependent nodes. Causality visualization was also briefly described in the section.

In chapter 2, a review was provided on the literature around the core concepts of the project, such as visual analytics, graph visualization and causality visualization. Dependency graphs were also discussed from the approach of parametric modeling.

In chapter 3, CZSaw's dependency graph was described in details in terms of development. The architecture was discussed and the design and the implementation were reviewed. The dependency graph version 2 was then proposed to overcome the shortcomings of the first version.

As mentioned before, perhaps the most important aspect of dependency graphs is the propagation of change. Various methods and techniques have been previously investigated for better visualizing the change propagation or the causality in the graph. A study was

designed and conducted to compare two novel causality visualization methods introduced for the first time in this work, as well as another customized method that is based on a well-know method. The study was described in chapter 4, along with the details about the prototype that was implemented merely for this purpose.

In chapter 5, the results of the study were described. The log files of the study were analyzed using a parser built for the study to derive the necessary information for investigating the timing and accuracy of task completion. The detection time for a causality visualization technique was introduced by decreasing the time needed for movement of the mouse from the total recorded task completion time. The movement time was calculated for each step using the model build with Fitts' law. The results confirmed our hypotheses in terms of task completion duration. There results did not confirm our hypotheses for the task completion accuracy. However, they suggested that there is room for more investigation in this area. Having a follow up study with more complicated tasks that better differentiate the 3 methods may improve the results.

5.4 Limitations and Future Work

Due to the limited amount of time, dependency graph version 2 was not deployed into CZSaw. Using this structure in CZSaw and adapting the visualization would benefit the tool with regards to performance and usability.
The dependency graph visualization leaves much room for improvement. The visualization needs to support more advanced interactivity. One desirable feature is to provide the means to allow the analysts to perform all the functionalities of the graph directly from visualization.
Also, providing a customizable and configurable visualization and making the analysts capable of choosing their own settings can be beneficial to support different types of users.
Finally, a future user experiment with more realistic settings and more complex tasks could better depict the effects of the used causality visualization technique.

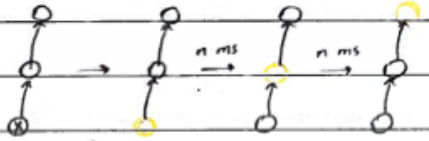
Appendix A

Sketches for Causality Visualization Techniques

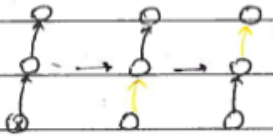
(I) drawing characteristics (static) of nodes & edges (lines & borders)

a) stroke color

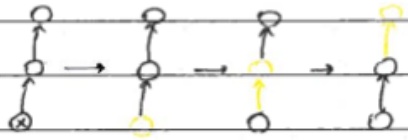
1. highlighting nodes



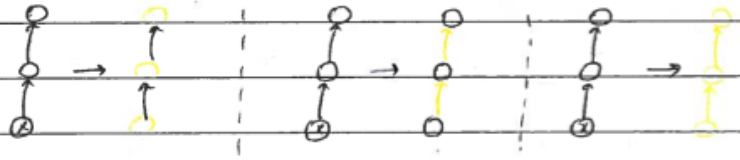
2. highlighting edges



3. highlighting nodes + edges

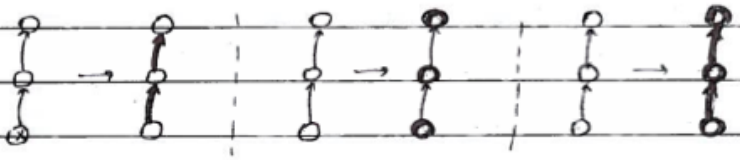


4. highlighting the whole path — nodes / edges / nodes + edges



b) stroke weight

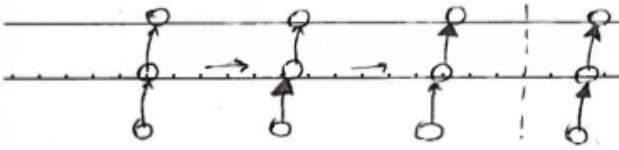
1. increase the width of the lines → edges & nodes



c) size

1. increas the size of arrows.

some methods are shown level-by-level, some are shown all at the same time

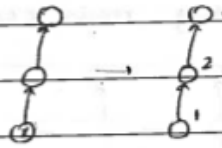


d) static characteristics of the content of the nodes

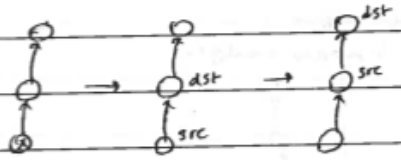
size of the text + style of the text | bold
italic
underscored
etc.

II) External Clues

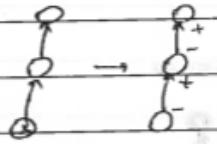
1) numbers beside nodes. (based on relative level to the source of update, or dfs/bfs traverse or —)



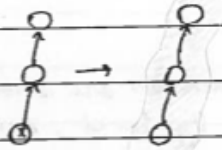
2) text beside nodes (src/dest)



3) signs (+/-)

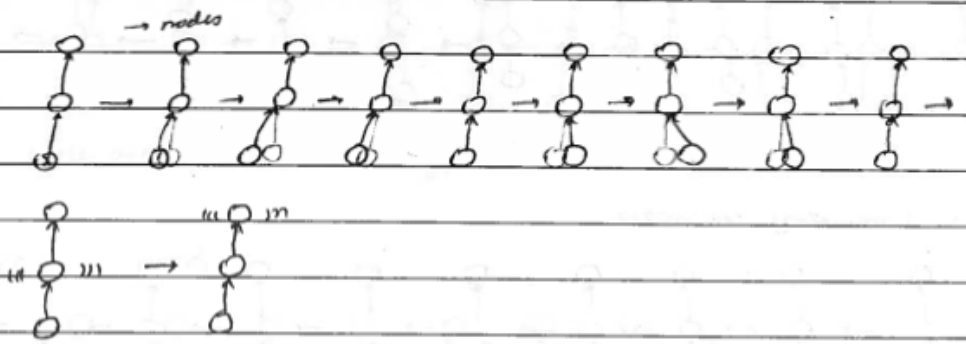


4) groups / clusters.

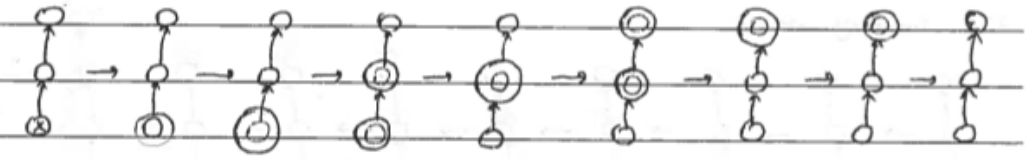


III) animation

1) vibration / oscillation → nodes

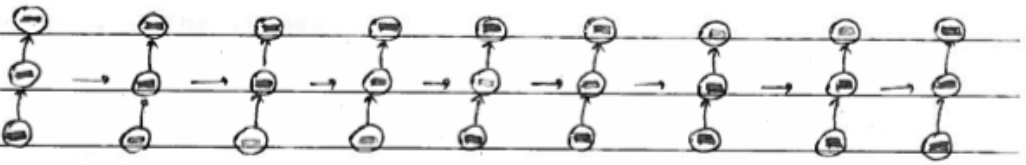


2) growing / shrinking → nodes

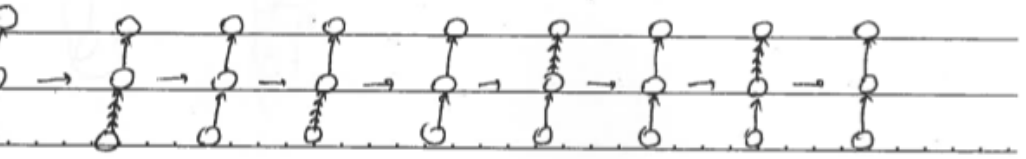


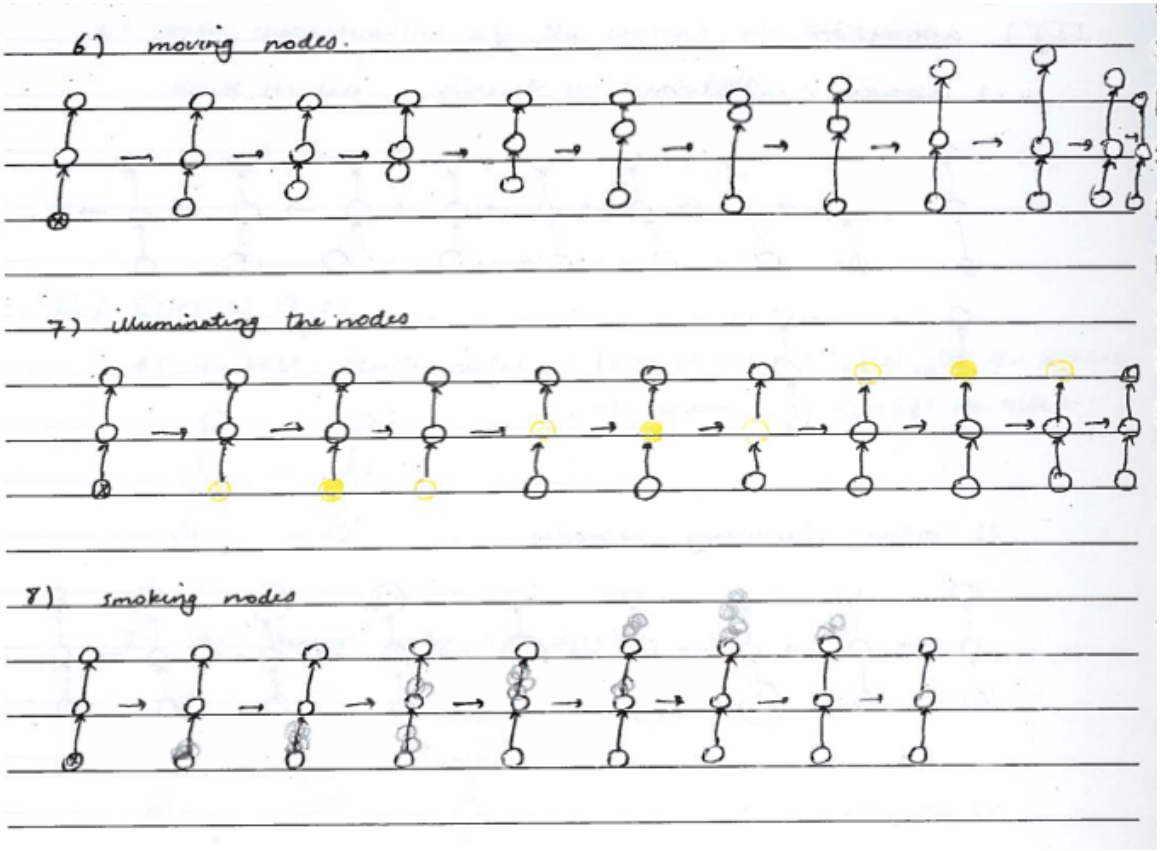
3) polygons (?)

4) blinking text → nodes (combined w/ making bold, etc.)



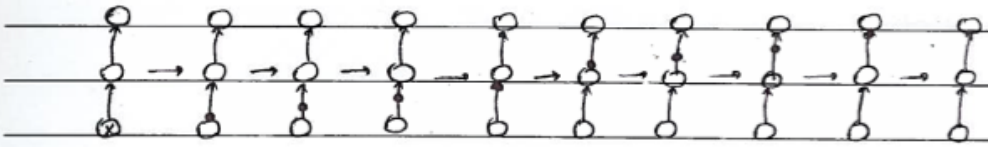
5) blinking arrows (or static)




6) moving nodes.
7) illuminating the nodes
8) smoking nodes

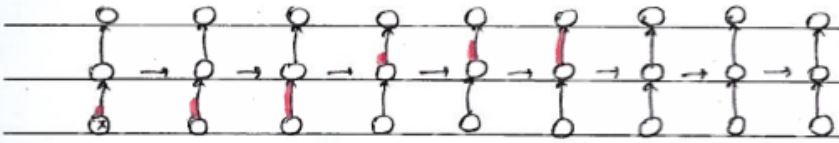
III. 2) VCV Date. No.

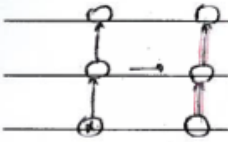
1) VCV → pinball



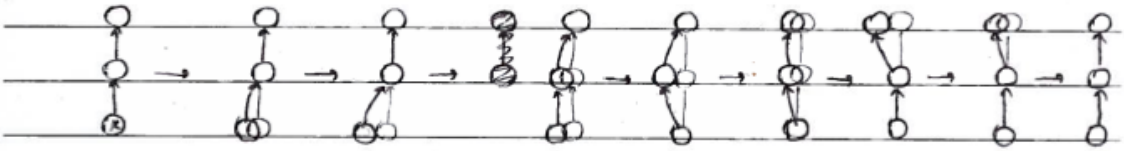
(or little arrows)

2) VCV → prod





3) VCV → wave



4) VCV → light comets.

Appendix B

Sample Output Files

This section includes snapshots of sample output files resulting from different stages of the experiment.

B.1 Experiment Initial Logs

Figure B.1 represents a sample page of one of the logs collected during the participation of one participant in the experiment. The information about hitting the "next" button, clicks and their positions, number of the level and number of nodes in each level, and appropriate timestamps are gathered in this stage.

B.2 Parser Output Files

After gathering the raw information, the log files are used as inputs to the parser. The parser then generates output files containing processed information about the distances between the clicks, the differences between the timestamps, and whether the participant has detected the target correctly or not (figure B.2).

B.3 First Clicks

Due to importance of first clicks in each level in detection of the causal path, the information about the durations of first clicks was extracted from the log files and investigated separately (figure B.3).

```
next @: 20172
Step: 1 , numOfNodes: 3
Node num: 0 in level, time: 22914, mouseX: 70, nodeX: 50.0, mouseY: 366, nodeY: 350.0
Node num: 1 in level, time: 24215, mouseX: 73, nodeX: 60.0, mouseY: 262, nodeY: 250.0
Node num: 2 in level, time: 25176, mouseX: 162, nodeX: 150.0, mouseY: 164, nodeY: 150.0
--------------
next @: 28197
Step: 2 , numOfNodes: 4
Node num: 0 in level, time: 30444, mouseX: 124, nodeX: 90.0, mouseY: 454, nodeY: 450.0
Node num: 1 in level, time: 31218, mouseX: 64, nodeX: 50.0, mouseY: 356, nodeY: 350.0
Node num: 2 in level, time: 31885, mouseX: 91, nodeX: 60.0, mouseY: 254, nodeY: 250.0
Node num: 3 in level, time: 32799, mouseX: 193, nodeX: 150.0, mouseY: 155, nodeY: 150.0
--------------
next @: 35324
Step: 3 , numOfNodes: 2
Node num: 0 in level, time: 37757, mouseX: 747, nodeX: 730.0, mouseY: 464, nodeY: 450.0
Node num: 1 in level, time: 39151, mouseX: 663, nodeX: 644.0, mouseY: 358, nodeY: 350.0
--------------
next @: 40406
Step: 4 , numOfNodes: 3
Node num: 0 in level, time: 43210, mouseX: 734, nodeX: 710.0, mouseY: 364, nodeY: 350.0
Node num: 1 in level, time: 44419, mouseX: 714, nodeX: 690.0, mouseY: 263, nodeY: 250.0
Node num: 2 in level, time: 45348, mouseX: 649, nodeX: 630.0, mouseY: 156, nodeY: 150.0
--------------
next @: 46154
Step: 5 , numOfNodes: 2
Node num: 0 in level, time: 48571, mouseX: 266, nodeX: 250.0, mouseY: 462, nodeY: 450.0
Node num: 1 in level, time: 49671, mouseX: 201, nodeX: 182.0, mouseY: 356, nodeY: 350.0
--------------
next @: 51096
Step: 6 , numOfNodes: 4
Node num: 0 in level, time: 54226, mouseX: 510, nodeX: 490.0, mouseY: 465, nodeY: 450.0
Node num: 1 in level, time: 54954, mouseX: 461, nodeX: 446.0, mouseY: 352, nodeY: 350.0
Node num: 2 in level, time: 56457, mouseX: 508, nodeX: 480.0, mouseY: 263, nodeY: 250.0
Node num: 3 in level, time: 57293, mouseX: 523, nodeX: 510.0, mouseY: 156, nodeY: 150.0
--------------
next @: 58115
Step: 7 , numOfNodes: 3
Node num: 0 in level, time: 60516, mouseX: 331, nodeX: 314.0, mouseY: 366, nodeY: 350.0
Node num: 1 in level, time: 61275, mouseX: 364, nodeX: 340.0, mouseY: 265, nodeY: 250.0
Node num: 2 in level, time: 62344, mouseX: 288, nodeX: 270.0, mouseY: 163, nodeY: 150.0
--------------
next @: 63739
Step: 8 , numOfNodes: 4
Node num: 0 in level, time: 66326, mouseX: 427, nodeX: 410.0, mouseY: 465, nodeY: 450.0
Node num: 1 in level, time: 67178, mouseX: 402, nodeX: 380.0, mouseY: 359, nodeY: 350.0
Node num: 2 in level, time: 68464, mouseX: 427, nodeX: 410.0, mouseY: 262, nodeY: 250.0
Node num: 3 in level, time: 69642, mouseX: 408, nodeX: 390.0, mouseY: 169, nodeY: 150.0
--------------
next @: 70556
Step: 9 , numOfNodes: 3
Node num: 0 in level, time: 73189, mouseX: 386, nodeX: 380.0, mouseY: 359, nodeY: 350.0
Node num: 1 in level, time: 74351, mouseX: 430, nodeX: 410.0, mouseY: 260, nodeY: 250.0
Node num: 2 in level, time: 75142, mouseX: 405, nodeX: 390.0, mouseY: 148, nodeY: 150.0
--------------
next @: 77047
Step: 10 , numOfNodes: 2
Node num: 0 in level, time: 79247, mouseX: 509, nodeX: 480.0, mouseY: 267, nodeY: 250.0
Node num: 1 in level, time: 80208, mouseX: 534, nodeX: 510.0, mouseY: 156, nodeY: 150.0
```

Figure B.1: Sample page of logs gathered during the experiment.

```
-1, -1, -1, 620.8, 92.4, next
1, 0, 2742, 70.0, 366.0, hit
1, 1, 1301, 73.0, 262.0, hit
1, 2, 961, 162.0, 164.0, hit
-1, -1, -1, 620.8, 92.4, next
2, 0, 2247, 124.0, 454.0, hit
2, 1, 774, 64.0, 356.0, hit
2, 2, 667, 91.0, 254.0, hit
2, 3, 914, 193.0, 155.0, hit
-1, -1, -1, 620.8, 92.4, next
3, 0, 2433, 747.0, 464.0, hit
3, 1, 1394, 663.0, 358.0, hit
-1, -1, -1, 620.8, 92.4, next
4, 0, 2804, 734.0, 364.0, hit
4, 1, 1209, 714.0, 263.0, hit
4, 2, 929, 649.0, 156.0, hit
-1, -1, -1, 620.8, 92.4, next
5, 0, 2417, 266.0, 462.0, hit
5, 1, 1100, 201.0, 356.0, hit
-1, -1, -1, 620.8, 92.4, next
6, 0, 3130, 510.0, 465.0, hit
6, 1, 728, 461.0, 352.0, hit
6, 2, 1503, 508.0, 263.0, hit
6, 3, 836, 523.0, 156.0, hit
-1, -1, -1, 620.8, 92.4, next
7, 0, 2401, 331.0, 366.0, hit
7, 1, 759, 364.0, 265.0, hit
7, 2, 1069, 288.0, 163.0, hit
-1, -1, -1, 620.8, 92.4, next
8, 0, 2587, 427.0, 465.0, hit
8, 1, 852, 402.0, 359.0, hit
8, 2, 1286, 427.0, 262.0, hit
8, 3, 1178, 408.0, 169.0, hit
-1, -1, -1, 620.8, 92.4, next
9, 0, 2633, 386.0, 359.0, hit
9, 1, 1162, 430.0, 260.0, hit
9, 2, 791, 405.0, 148.0, hit
-1, -1, -1, 620.8, 92.4, next
10, 0, 2200, 509.0, 267.0, hit
10, 1, 961, 534.0, 156.0, hit
-1, -1, -1, 620.8, 92.4, next
11, 0, 2169, 192.0, 461.0, hit
11, 1, 1797, 189.0, 463.0, miss
-1, -1, -1, 620.8, 92.4, next
12, 0, 2324, 463.0, 358.0, hit
12, 1, 697, 497.0, 261.0, hit
12, 2, 806, 533.0, 151.0, hit
-1, -1, -1, 620.8, 92.4, next
13, 0, 2819, 594.0, 470.0, hit
13, 1, 853, 532.0, 360.0, hit
13, 2, 1394, 566.0, 260.0, hit
-1, -1, -1, 620.8, 92.4, next
14, 0, 2386, 990.0, 365.0, hit
14, 1, 1038, 979.0, 258.0, hit
14, 2, 1611, 902.0, 162.0, hit
-1, -1, -1, 620.8, 92.4, next
15, 0, 3378, 992.0, 464.0, hit
15, 1, 1332, 1060.0, 368.0, hit
15, 2, 1364, 1059.0, 260.0, hit
```

Figure B.2: Sample page of the output of the parser using the information gathered during the experiment as input.

```
1766.4888340369139, 1
1271.6924169503573, 1
1559.0376092788297, 1
1993.5676046013964, 1
1483.0821000944402, 1
2258.1606362696803, 1
1523.5940301396608, 1
1697.8768117631985, 1
1781.1616628527217, 1
1464.574704149657, 1
1212.6659052796776, 1
1502.9210580817212, 1
1953.0666042418388, 1
1476.95538251821, 1
2438.440354079163, 1
1980.846761992571, 1
1944.057766410801, 1
1687.241105880456, 1
1369.7549532132598, 1
2018.5997717265698, 1
1987.074605527479, 1
2080.5751890625975, 1
1009.5002662282852, 1
1917.6017543496853, 1
1764.5717113020005, 1
2421.511370469926, 1
1157.815586307825, 1
15208.863105850274, 2
6579.706982669993, 2
3932.0995646822935, 2
2654.676065463557, 2
2983.1488528915784, 2
2892.5222195925744, 2
2217.8618787377804, 2
2779.1951251990968, 2
2447.5271788026807, 2
1550.1628607774605, 2
1286.655084270204, 2
3020.904756715597, 2
1653.21367214487, 2
2321.100691083803, 2
4086.3170483662125, 2
2913.5063752203723, 2
2095.1186093085153, 2
2087.875850679061, 2
3728.0344602897603, 2
2743.688674640864, 2
3085.215740141216, 2
3641.403073309285, 2
2777.4946293757407, 2
2905.925708690572, 2
3355.5000536149446, 2
2085.6513275552484, 2
1625.3535191304263, 2
1951.786704426333, 3
2448.9291514176894, 3
1652.534275091759, 3
1580.7053517255995, 3
2613.28472660416, 3
```

Figure B.3: Timing information of the first clicks of each level of the experiment.

Appendix C

Study Files

This chapter includes a sample the information consent form signed by the participants.

Information Consent Form

In the following study, animation is used to help the viewer detect a change in the graph. The goal of the study is to investigate and compare different animation types based on the duration and accuracy of task completion, and also user preferences. In addition, the speed of the animations will be investigated.

Title of the study: Visualizing Causality in Propagation Graphs

Principal Investigator: Saba Alimadadi Jani

Supervisor: Dr. Chris Shaw

Application number: 2012s0841

The study is being done under the auspices of SFU.

You will be viewing propagation graphs. The nodes are the entities that could be changed or updated and the links between the nodes show the dependency relations between the nodes. If node A is dependent upon node B, any change in node B will affect node A, and so the update will be propagated. During this study, you are asked to detect the nodes that get updated (shown using the animation) and click on them. You must click on the changed node as soon as you detect the change, but without sacrificing the accuracy; the positions of the clicks are important as well. The prototype contains 84 rounds of update. You can go to the next round by pressing the "next" button on top of the screen. After you pressed "next", an update event will get fired on a node and will get propagated to the dependent node(s). You must detect the dependent nodes using the animation and click on the as quickly and accurately as possible. After you finished each round (clicked on all updated nodes), it is not important how much time you take to yourself, before hitting the "next" button again. You may rest in these time intervals.

Also, you can leave the experiment at any time that you wish. You may withdraw of your voluntary participation without consequences. Upon completion of the experiment, a cash value of $10 CAD will be awarded to you.

Complaints to be directed to:

Chris Shaw, Associate Professor, Simon Fraser University Surrey, B.C. Canada V3T 0A3

778 782 7506 [redacted]

Hal Weinberg, Director Office of Research Ethics Simon Fraser University Burnaby, B.C. Canada V5A 1S6

778 782 6593 [redacted]

You can obtain the research results from Saba Alimadadi Jani [redacted] after the results are processed, analyzed and documented.

After the completion of this experiment, there will be no contact with you from the investigator team. However, feel free to contact the primary investigator should you have any comments, questions or concerns.

The data collected during this experiment is completely anonymous and total confidentiality is promised.

The collected data will be in form of text documents and will not include any personal information about the participant. The gathered date will be put in a password protected folder in the investigator's flash memory stick, and will be locked in a secure cabinet for two years from the date of the study.

Participant's Name:

_____

Participant's Signature:

_____

Date:

_____

Figure C.1: The information consent form.

Bibliography

[1] Jmp. http://www.jmp.com/. Accessed: 30/10/2012.

[2] Jung. http://jung.sourceforge.net/. Accessed: 30/10/2012.

[3] Junit. www.junit.org/. Accessed: 30/10/2012.

[4] Processing. http://www.processing.org. Accessed: 30/10/2012.

[5] R. Aish and R. Woodbury. Multi-level interaction in parametric design. In *Smart Graphics*, pages 924–924. Springer, 2005.

[6] C. Alexander. *The timeless way of building*, volume 1. Oxford University Press, USA, 1979.

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

[8] L. Bartram and C. Ware. Filtering and brushing with motion. *Information Visualization*, 1(1):66–79, 2002.

[9] L. Bartram and M. Yao. Animating causal overlays. *Computer Graphics Forum*, 27(3):751–758, 2008.

[10] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom.*, 4:235–282, 1994.

[11] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

[12] E. Baumgartner. "causal mapping".

[13] K. Beck. *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[14] B. Bederson and J. Hollan. Pad++: a zooming graphical interface for exploring alternate interface physics. In *Proceedings of the 7th annual ACM symposium on User interface software and technology*, UIST '94, pages 17–26, New York, NY, USA, 1994. ACM.

[15] E. Bier, M. Stone, K. Pier, W. Buxton, and T. DeRose. Toolglass and magic lenses: the see-through interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 73–80, New York, NY, USA, 1993. ACM.

[16] U. Brandes, D. Fleischer, and T. Puppe. Dynamic spectral layout of small worlds. In *Graph Drawing*, pages 25–36, 2005.

[17] C. Chen. Top 10 unsolved information visualization problems. *IEEE Computer Graphics and Applications*, 25(4):12–16, 2005.

[18] M. P. Consens, I. F. Cruz, and A. O. Mendelzon. Visualizing queries and querying visualizations. *ACM SIGMOD Record*, 21:39–46, 1992.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2001.

[20] M. Derthick and S. F. Roth. Data exploration across temporal contexts. In *Proceedings of the 5th international conference on Intelligent user interfaces*, pages 60–67. ACM, 2000.

[21] D. K. Van Duyne, J. A. Landay, and J. I. Hong. *The design of sites: patterns, principles, and processes for crafting a customer-centered Web experience*. Addison-Wesley Professional, 2003.

[22] P. Eades and Q. Feng. Multilevel visualization of clustered graphs. In *Graph Drawing*, pages 101–112, 1996.

[23] M. R. Edward and S. Tilford. Tidier drawing of trees. *IEEE Transactions on Software Engineering*, 1981.

[24] J. Eklund, J. Sawers, and R. Zeiliger. A tool for the collaborative construction of knowledge through constructive navigation. In *Ausweb99, The Fifth Australian World Wide Web Conference*, pages 396–408, Lismore, 1999. Southern Cross University Press.

[25] N. Elmqvist and P. Tsigas. Animated visualization of causal relations through growing 2d geometry. *Information Visualization*, 3(3):154–172, 2004.

[26] P. Evitts and D. Hinchcliffe. *A UML pattern language*. Macmillan Technical Pub., 2000.

[27] P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of experimental psychology*, 47(6):381, 1954.

[28] U. Frishman and A. Tal. Dynamic drawing of clustered graphs. In *INFOVIS*, pages 191–198, 2004.

[29] Y. Frishman and A. Tal. Online dynamic graph drawing. In *EuroVis*, pages 75–82, 2007.

[30] G. W. Furnas. Generalized fisheye views. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '86, pages 16–23, New York, NY, USA, 1986. ACM.

[31] G. W. Furnas and B. Bederson. Space-scale diagrams: understanding multiscale interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 234–241, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[32] D. Gotz and M. X. Zhou. Characterizing users' visual analytic activity for insight provenance. *Information Visualization*, 8(1):42–55, 2009.

[33] C. Grg, P. Birke, M. Pohl, and S. Diehl. Dynamic graph drawing of sequences of orthogonal and hierarchical graphs. In *In Proceedings of 12th International Symposium on Graph Drawing*, pages 228–238, 2004.

[34] P. Hanrahan. Visual thinking for business intelligence. Technical report, Tableau Software, 2005.

[35] J. Heer, J. Mackinlay, C. Stolte, and M. Agrawala. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1189–1196, 2008.

[36] I. Herman, G. Melançon, and M. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, January 2000.

[37] I. Herman, G. Melanon, M.M. de Ruiter, M. Delest, Mathematisch Centrum (smc, and The Dutch Foundation. Latour - a tree visualisation system, 1999.

[38] M. L. Huang and P. Eades. A fully animated interactive system for clustering and navigating huge graphs. In *Proceedings of the 6th International Symposium on Graph Drawing*, GD '98, pages 374–383, London, UK, UK, 1998. Springer-Verlag.

[39] X. Huang, W. Lai, A. S. M. Sajeev, and J. Gao. A new algorithm for removing node overlapping in graph visualization. *Inf. Sci.*, 177(14):2821–2844, July 2007.

[40] D. E. Huber and C. G. Healey. Visualizing data with motion. In *IEEE Visualization*, page 67, 2005.

[41] M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *J. Graph Algorithms Appl.*, 1, 1997.

[42] D. Jungnickel. *Graphs, Networks and Algorithms*. Springer Publishing Company, Incorporated, 3rd edition, 2007.

[43] N. R. Kadaba, P. Irani, and J. Leboe. Visualizing causal semantics using animations. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1254–1261, 2007.

[44] N. Kadivar, V. Chen, D. Dunsmuir, E. Lee, C. Qian, J. Dill, C. Shaw, and R. Woodbury. Capturing and supporting the analysis process. In *IEEE VAST*, pages 131–138. IEEE, 2009.

[45] M. Kaufmann and D. Wagner, editors. *Drawing graphs: methods and models*. Springer-Verlag, London, UK, UK, 2001.

[46] T. A. Keahey and E. L. Robertson. Techniques for non-linear magnification transformations. In *Proceedings of the 1996 IEEE Symposium on Information Visualization (INFOVIS '96)*, INFOVIS '96, Washington, DC, USA, 1996. IEEE Computer Society.

[47] K. Korb and A. Nicholson. *Bayesian Artificial Intelligence*. Chapman and Hall, 2nd edition, 2010.

[48] E. Kraemer and J. Stasko. Creating an accurate portrayal of concurrent executions. *IEEE Concurrency*, 6(1):36–46, January 1998.

[49] M. Kreuseler, T. Nocke, and H. Schumann. A history mechanism for visual data mining. In *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium on*, pages 49–56. IEEE, 2004.

[50] G. Kumar and M. Garland. Visual exploration of complex time-varying graphs. *IEEE Transactions on Visualization and Computer Graphics*, 12:805–812, 2006.

[51] P. Kumar and K. Zhang. Visualization of clustered directed acyclic graphs with node interleaving. In *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, pages 1800–1805, New York, NY, USA, 2009. ACM.

[52] P. Kumar, K. Zhang, and Y. Wang. Visualization of clustered directed acyclic graphs without node overlapping. In *IV*, pages 38–43, 2008.

[53] M. Laguna and R. Martí. Grasp and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11(1):44–52, 1999.

[54] M. Laguna, R. Martí, and V. Valls. Arc crossing minimization in hierarchical digraphs with tabu search. *Computers & OR*, 24(12):1175–1186, 1997.

[55] W. Lai. Layout adjustment and boundary detection for a diagram. In *Proceedings of the International Conference on Computer Graphics*, CGI '01, Washington, DC, USA, 2001. IEEE Computer Society.

[56] Y. Lee, C. Lin, and H. Yen. Mental map preserving graph drawing using simulated annealing. In *Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation - Volume 60*, APVis '06, pages 179–188, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[57] Y. K. Leung and M. D. Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Trans. Comput.-Hum. Interact.*, 1(2):126–160, June 1994.

[58] F. Leymann, editor. *Web Services Flow Language (WSFL 1.0)*. IBM, May 2001.

[59] W. Li, P. Eades, and N. Nikolov. Using spring algorithms to remove node overlapping. In *proceedings of the 2005 Asia-Pacific symposium on Information visualisation - Volume 45*, APVis '05, pages 131–140, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[60] I. S. MacKenzie. Movement time prediction in human-computer interfaces. In *Proceedings of Graphics Interface*, volume 92, pages 140–150, 1992.

[61] I. S. MacKenzie and W. Buxton. Extending fitts' law to two-dimensional tasks. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 219–226. ACM, 1992.

[62] I. S. MacKenzie, A. Sellen, and W. Buxton. A comparison of input devices in elemental pointing and dragging tasks. In *Proceedings of ACM CHI 91 Conference on Human Factors in Computing Systems*, pages 161–166, 1991.

[63] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *J. Vis. Lang. Comput.*, 6(2):183–210, 1995.

[64] Y. Moses, Z. Polunsky, A. Tal, and L. Ulitsky. Algorithm visualization for distributed environments. In *INFOVIS*, pages 71–78, 1998.

[65] S. Nadkarni and P. Shenoy. A bayesian network approach to making inferences in causal maps. *European Journal of Operational Research*, 128(3):479–498, 2001.

[66] E. Neufeld, J. Solheim, and S. Kristtorn. Experiments in the perception of causality. In *Smart Graphics*, pages 36–49, 2006.

[67] E. Neufeld, J. Solheim, and S. Kristtorn. Experiments in the perception of causality. In *Smart Graphics*, pages 36–49. Springer, 2006.

[68] S. C. North. Incremental layout in dynadag. In *In Proceedings of the 4th Symposium on Graph Drawing (GD*, pages 409–418. Springer-Verlag, 1995.

[69] P. Pirolli and S. Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of International Conference on Intelligence Analysis*, volume 5, pages 2–4, 2005.

[70] C. Plaisant, A. Rose, G. Rubloff, R. Salter, and B. Shneiderman. The design of history mechanisms and their use in collaborative educational simulations. In *Proceedings of the 1999 conference on Computer support for collaborative learning*, page 44. International Society of the Learning Sciences, 1999.

[71] H. Purchase. Which aesthetic has the greatest effect on human understanding? In *Proceedings of the 5th International Symposium on Graph Drawing*, GD '97, pages 248–261, London, UK, 1997. Springer-Verlag.

[72] Z. Qian, Y. Chen, and R. Woodbury. Participant observation can discover design patterns in parametric modeling. In *Proceedings of the 27th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA 2007)*, pages 230–241, 2007.

[73] A. C. Robinson and C. Weaver. Re-visualization: Interactive visualization of the process of visual analysis. In *Workshop on Visualization, Analytics & Spatial Decision Support at the GIScience conference*, 2006.

[74] D. Schaffer, Z. Zuo, S. Greenberg, L. Bartram, J. Dill, and M. Roseman. Navigating hierarchically clustered networks through fisheye and full-zoom methods. *ACM Transactions on Computer-Human Interaction*, 3:162–188, 1998.

[75] R. W. Soukoreff and I. S. MacKenzie. Generalized fitts' law model builder. In *Conference companion on Human factors in computing systems*, pages 113–114. ACM, 1995.

[76] J. Stasko, C. Görg, and Z. Liu. Jigsaw: supporting investigative analysis through interactive visualization. *Information Visualization*, 7(2):118–132, April 2008.

[77] K. Stuart, K. William, and J. Betty. Evaluation of mouse, rate-controlled isometric joystick, step keys, and text keys for text selection on a crt. *Ergonomics*, 21(8):601–613, 1978.

[78] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man & Cybernetics*, 11(2):109–125, 1981.

[79] J. J. Thomas and K. A. Cook, editors. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. National Visualization and Analytics Ctr, August 2005.

[80] J. Tidwell. *Designing interfaces*. O'Reilly Media, Incorporated, 2010.

[81] C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[82] C. Ware, J. Bonner, R. Cater, and W. Knight. Simple animation as a human interrupt. *International Journal of Human-Computer Interaction*, 4(4):3411–348, 1992.

[83] C. Ware and M. Lewis. The dragmag image magnifier. In *Conference Companion on Human Factors in Computing Systems*, CHI '95, pages 407–408, New York, NY, USA, 1995. ACM.

[84] C. Ware, E. Neufeld, and L. Bartram. Visualizing causal relations. In *IEEE Symposium on Information Visualization*, 1999.

[85] J. H. Wigmore. *The Problem of Proof.* llinois Law Review, 1913.

[86] G. J. Wills. Nicheworks - interactive visualization of very large graphs. In *Graph Drawing*, pages 403–414, 1997.

[87] N. Wong, S. Carpendale, and S. Greenberg. Edgelens: an interactive method for managing edge congestion in graphs. In *Proceedings of the Ninth annual IEEE conference on Information visualization*, INFOVIS'03, pages 51–58. IEEE Computer Society, 2003.

[88] R. Woodbury. Elements of parametric design. 2010.

[89] R. Woodbury, A. Kilian, and R. Aish. Some patterns for parametric modeling. In *Expanding Bodies: Art • Cities • Environment: Proceedings of the 27th Annual Conference of the Association for Computer Aided Design in Architecture*, pages 222–229, Halifax (Nova Scotia), 1-7 October 2007. Riverside Architectural Press and Tuns Press.

[90] S. Wright. *Correlation and Causation.* Washington, DC, 1921.

[91] J. Yi, Y. Kang, J. Stasko, and J. Jacko. Toward a deeper understanding of the role of interaction in information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1224–1231, November 2007.

[92] R. Zeiliger, C. Belisle, and T. Cerratto. Implementing a constructivist approach to web navigation support. In *ED-MEDIA'99 Conference*, pages 19–24, Seattle,Wa., USA, June 1999.