Roma Tre University

Doctoral School in Computer Science and Automation

Ciclo XXVIII

Doctoral dissertation:

Visual Analytics of Network Routing Through Traceroute Data: Models and Techniques

Author: Marco Di Bartolomeo

Advisors: Prof. Giuseppe Di Battista
Prof. Maurizio Patrignani

Spring 2016

Visual Analytics of Network Routing Through Traceroute Data: Models and Techniques

A thesis presented by
Marco Di Bartolomeo
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Engineering

Roma Tre University
Department of Engineering

Spring 2016

COMMITTEE:
Prof. Giuseppe Di Battista, Dept. of Engineering, Roma Tre University
Prof. Maurizio Patrignani, Dept. of Engineering, Roma Tre University

REVIEWERS:
Prof. Stephen G. Kobourov, Dept. of Computer Science, University of Arizona
Emden R. Gansner, Google Inc.

"A good decision
is based on knowledge
and not on numbers."

(Plato. Laches*. 4th century B.C.)*

Contents

Chapter 1

Introduction

The Internet has become a fundamental part of our life. Born as a network for scientific purposes, it has grown in size and services to the point to become the backbone of many human daily activities. Studying, shopping, and banking, are examples of activities in which the use of the Internet is nowadays well established. The extraordinary diffusion of mobile devices (estimated in 2 billion of connected units in 2016 [Int16]) is greatly contributing in making people use online services. Multimedia services have an increasing importance in this framework, and, in fact, there is a trend in the last years to offer multimedia products over the Internet to the general public. Telephone, music, and movie streaming are examples of this trend, in which names like e.g. *Skype*, *Netflix*, *Youtube*, and *Spotify* proved to be prominent players. This phenomenon is advantageous for several stakeholders. *Content providers* can exploit a robust and world-wide distributed network for distributing their contents, easily reaching old and new customers at a fraction of the costs necessary for building and maintaining traditional, dedicated infrastructures. This has a direct impact on *customers*, who receive more complete services at lower prices. Also, these services are often more interactive than traditional ones, thanks to the digital nature of the Internet, which enables a personalized user experience. Finally, *Internet Service Providers* (ISPs) are the intermediary in this context. These organizations, traditionally, have developed and hosted the physical networks that run the Internet, and today they see new market opportunities in developing high-performance infrastructures for multimedia Internet services.

Internet Service Providers are faced with the challenging task of developing and maintaining networks that increase in size at a dramatic pace but, at the same time, must support multimedia Internet services by providing acceptable performance. In

this scenario, metrics are a fundamental tool. Measuring the performance of a network allows for a continuous monitoring, supporting the discovery of faults and the tuning of parameters. ISPs have always used some kind of local monitoring in their networks, for example embedded in *routers*, which are intermediate devices. However, given the size of modern networks, a local alert raised by a router does not necessarily represent the experience of a user, whose packets traverse long paths in the network. He could have a much worse perception of a fault, because of multiplicative effects along the path. Or, it could not notice the fault at all, because he is far from it and the effect on his connection is only negligible. *Probe systems* are a recent attempt to deal with this problem, and are gathering a growing interest. Such a system distributes small devices called *probes* across the Internet, which are always connected and continuously perform standard network measurements towards selected targets. Some common measurements that are performed are ping, traceroute, HTTP queries, etc. The results of the measurements are collected in large repositories, which are available for further analysis. The key feature of probes is that they are installed near to real users, often in their houses, hence simulating the actual user experience through the metrics they collect.

Among the measurements available in a probe system, *traceroute* is a standard networking tool that records the path followed by data in the network, from the source to the target. It also records the round-trip time between the source and each intermediate node. Like other standard measurements, it is supported by default by any IP-based network, like the Internet. Differently from other measurements, traceroute data contain intrinsic topological information, since a traceroute basically represents a path in the network. This means that they can reveal details on the structure of the network, in addition to its performance. At any instant, a protocol decides the *routing* of the network, which is a set of rules that establish what path is followed by packets to go from a given source to a given destination. In this sense, traceroute is a simple yet effective tool for sampling the status of the routing at a given instant. If several traceroute paths are merged, the result is the *traceroute graph*, or *routing graph*, which represents an approximation of the network topology and of the routing on that network at a given instant.

However, the information richness of traceroutes makes them difficult to handle and understand. In fact, most existing tools make only a partial use of traceroutes, showing single paths and the relative round-trip times, without any attempt to process and emphasize the topological information. This underuse can be explained by some challenges, listed in the following, that are encountered when processing traceroutes produced by a probe system.

Data Size A large probe system ensures a fine-grained sampling of the Internet, but

can also produce a humongous amount of data. In fact, thousands of probes can be operating at the same time, performing measurements every few minutes towards a same target for hours or days. The collected data can contain many interesting insights on the network, but transforming that large amount of data into useful information for human operators requires automatic processing methods, and suitable visualizations. While this is rather easy for numerical metrics, automatically processing and visualizing in a meaningful way topological data is complex, with the large data size making the task harder.

Dynamics Routing is a dynamic entity. It continuously changes as a reaction to network faults or to make an efficient use of the network, i.e. by distributing the load over different nodes. Probe systems execute traceroutes periodically, and for this reason produce a sampling of routing dynamics, as a sequence of snapshots. In some sense, dynamics is the most interesting aspect of routing, and probe systems give an opportunity to study its evolution. But structured data that change over time are hard to process and represent. In particular, dynamic graphs are known to be very challenging to visualize in an effective way.

Relation To Metrics Routing changes require to be correlated to metrics to be understood. Indeed, if the path between two nodes as captured by two consecutive traceroutes changed, it is hard to determine the reason for the change without knowing how the value of some metric of interest changed at the same moment. Traceroutes are naturally related to the round-trip time delay, but other metrics are possible. While visualizing metrics alone is relatively easy, there is not a consensus on how to effectively correlate numerical values with topological data in a dynamic setting.

Relation To Geography Traceroutes intrinsically contain geographical information. Each node of a traceroute is represented with an IP addresses, which can be roughly mapped to a geographical position by means of heuristics. This relationship to geography is interesting for data analysis, since the relative geographical locations of two nodes can influence the routing between them. But, at the same time, it represents a strong constraint for visualization. Indeed, the most natural and effective way to display geographical positions is superimposing them on a geographical map, which prevents further optimizations of the visualization. This becomes critical when positions are relative to nodes of a computer network, since these data are subject to scale problems. For example, a traceroute can have its endpoints in two large cities, where nodes are relatively near to each other, and then traverse a long oceanic cable with few intermediate nodes. If the path is shown in its full extent in a geographical visualization, the

areas near to the endpoints show a lot of visual clutter. On the other hand, if only an endpoint is shown so to reduce the clutter, the global view is lost.

As a result of the described problems, nowadays large amounts of traceroutes are collected without their full potential is exploited for data analysis. This thesis has the objective of supporting ISPs in making use of massive traceroute data for managing their networks. The products of the study are interactive tools, that implement novel processing algorithms and visualization metaphors for traceroutes. These tools provide human users with a simplified but flexible view of data, supporting tasks like network debugging and design. The framework is that of *visual analytics*, a research field that combines automated analysis techniques with interactive visualizations for an effective understanding, reasoning and decision making on the basis of very large and complex data sets [KAF+08]. For reaching the objective, the methodology of this study deals with the aforementioned challenges by applying several approaches.

Multiple Abstraction Levels The Internet is partitioned in large subnetworks called *Autonomous Systems*, or ASes, each belonging to a single administration. Since every node corresponds to exactly one AS, it is possible to convert a traceroute path to a shorter path of traversed ASes. The partitioning in ASes is an example that gives the intuition that traceroutes can be studied at different levels of abstraction. While abstracting data is a way to mitigate their size, in the context of network management the user still needs to see the details, i.e. single traceroute paths. The reason is that not only it is not predictable the specific set of details the user wants to see, but once he has spotted a macroscopic network behavior, he also wants to know the low level causes, to be able to intervene on the network. Therefore, tools must allow the user to look at the data at different abstraction levels, and to easily compare them.

Routing Events Often an ISP has some initial information from which to start the analysis of a network problem. A date and a time at which an issue manifested, or a set of nodes that were updated in a recent maintenance action, are examples of input for deeper data analysis. In these situations, switching between abstraction levels helps circumscribe and understand the problem. But sometimes there are no initial information available, because the ISP was not aware of the existence of a problem. The large amount of traceroutes collected by probes could have discovered an unnoticed problem, but it is too hard to find it by looking at the data in detail. For this reason, data must be automatically processed for finding macroscopic routing behaviors that could start the analysis. Such macroscopic behaviors are here introduced and studied as *routing events*.

		Methodologies		
		Multiple Abstraction Levels	Routing Events	Topology + Geography
Challenges	Data Size	✓	✓	
	Dynamics	✓	✓	
	Relation To Metrics	✓		
	Relation To Geography		✓	✓

Table 1.1: Impacted challenges regarding the analysis of traceroute data produced by a probe system, for each of the methodologies of this thesis.

Topology + Geography As discussed, geographical visualizations of topological data suffer from visual clutter, and a reduced possibility to optimize the visualization. On the other hand, correlating routing dynamics to the geographical positions of the involved nodes could outline further insights on the causes of those dynamics. For this reasons, tools must support geography, in a hybrid visualization that allows the user to compare it to the topological information.

Fig. 1.1 briefly shows which challenges are met by the methodologies described in this study.

The thesis is structured in three Parts, each discussing the application of one methodology and each divided in two chapters. Chapter 2 introduces preliminary concepts and definitions that are used through the rest of the thesis. Chapter 3 describes the application of multiple abstraction levels for visualizing the dynamics and the structure of the routing graph. Chapter 4 applies multiple abstraction levels to metrics, showing how a set of time series can be visually arranged to allow an easy comparison of each of them to their global trend. Chapter 5 introduces the concept of routing event as a model of high-level routing dynamics, and describes an algorithm for efficiently discovering routing events in a large set of traceroutes. Chapter 6 describes a visual approach for analyzing routing events, enabling the user to visually discover patterns of events that are hard to detect automatically. Chapter 7 studies a graph planarity problem related to drawing a graph whose nodes have a geographical position, in such a way that the drawing has a good readability without being too different from a pure geographical visualization. Finally, Chapter 8 describes a heuristic approach for drawing a graph whose nodes have a geographical position.

Chapter 2

Preliminary Concepts and Definitions

This chapter introduces the basic concepts and terminology used through the rest of the thesis, and related to the two fields of interest: computer networks, and graph drawing.

2.1 Computer Networks

Routing in the Internet

The Internet is a large, distributed network of computers and other devices. Among these, *routers* are intermediate devices that have the task to route traffic across the network, i.e. to decide what path data should follow to go from a given source to a given destination. This activity is also called *routing* and is executed by routers one link at the time, that is, a router chooses which is the next node (or "hop") to traverse among its closest neighbors. Routing is a dynamic entity. A *routing protocol* is continuously executed on the network to compute the best path between each pair of nodes, on the basis of their availability and performance. For example, if a given node experiences a fault, the routing protocol could replace it with an alternative node in every path that traverses it. Load balancing is another reason for which routing changes. Namely, particular nodes of the network called *load balancers* distribute traffic across several alternative devices, following some policy. *Traceroute* is a standard networking tool that reports the path followed by data to reach a destination, and it is executed by the source. The reported sequence of routers is called *traceroute path*. Network devices are labeled with an *IP address*, so a traceroute path is a sequence of IP addresses. For each node, a traceroute also reports the *round-trip time* (RTT), which is the length of time it takes for a signal to be sent plus the length of time it takes for an acknowledg-

ment of that signal to be received. Network devices can be configured to not respond to traceroutes for security reasons, in such a case the traceroute tool reports an asterisk ("*") in place of the missing node. Several traceroute paths can be merged to form a reticular structure (or *graph*) that is a partial view of the actual topology of the traversed network as seen by routing, and is called *routing graph*, or *traceroute graph*. Two consecutive traceroutes between the same pair of nodes reporting two different paths are an evidence that routing changed at some time instant between the two measurements. Traceroutes periodically performed from several observation points of the network effectively sample the evolution of the routing graph over time. If the traceroutes were collected all at the same time, the resulting graph is an approximation of the instantaneous state of routing. Otherwise, it represents several routing states, that is, it contains several paths that had been available from a source to a destination at different times.

Deciding all paths to follow in a network requires the knowledge of the entire network, which is impractical for the size of the Internet, so this activity is performed only at a local level on partitions of the network called *Autonomous System*s, or ASes. An AS belongs to a single administrative authority, like an *Internet Service Provider* (ISP), a company, a university, etc, and is globally identified with a number called AS Number (ASN). ASes are inter-connected and form a more abstract view of the Internet. Routing protocols do not cross the boundary of an AS, so to make ASes exchange their traffic a specific protocol exists, called *Border Gateway Protocol* (BGP). It is a policy-based protocol implemented by ASes by exchanging special messages called *announcements*. An announcement basically tells the world either "I own these IP addresses" or "I know a path to reach these IP addresses". Usually ASes establish links to exchange their traffic on the basis of commercial agreements, which assigns a "customer-provider" meaning to the link. Since BGP announcements are publicly available, organizations collect them to infer the links between ASes, and the AS to which an IP address belongs. With a map from an IP address to its AS, it is possible to convert a traceroute path into a shorter path of traversed ASes, and a routing graph into a more abstract graph of ASes.

Probe Systems

Network operators like Internet Service Providers, Content Delivery Networks (CDN), and cloud providers are strongly committed to offer reliable and efficient services to their customers. Besides being a way to keep a prominent position on the market, this is also a requirement encoded in Service Level Agreements (SLAs), whose violation can cause financial penalties. Achieving this requires facilities to monitor the network infrastructure for timely detection of possible problems and to help operators in

troubleshooting. Network management suites usually deal with numeric metrics (e.g., round-trip delay, error rate) and with status information (e.g., interface status). They rarely consider end-to-end information, and, if so, it is limited to devices under direct control of the operator.

A probe system is a set of interconnected network devices called *probes*, that periodically perform traceroutes (and other measurements) towards fixed destinations. The collected traceroute paths are stored in huge databases and represent a sampling of how the routing, as controlled by routing protocols, evolved over time on a portion of the Internet. Probes are often installed near to the home of users, so to perform *end-to-end* measurements of the access towards critical Internet services.

There are several organizations that distribute probes in the Internet with the aim of monitoring the status of the network and of measuring its performance. A few examples of such organizations follow. SamKnows [sam15] probes (tens of thousands) are distributed world-wide to get broadband performance data for consumers, governments, and Internet Service Providers. BISmark [SdDF+11] uses probes for measuring the performance of ISPs. RIPE Atlas [atl15] is an open project of RIPE-NCC whose probes (almost ten thousand) can be used to conduct customized measurements by anyone willing to host a RIPE Atlas probe. MisuraInternet [agc] is a probe-based project of the Italian Authority for Telecommunications that measures the quality of broadband access. Other notable examples are CAIDA Ark [ark15], and M-Lab [mla15], that use probes to continuously perform measurements towards several targets.

2.2 Graph Drawing

Concepts of Graph Theory

A *graph* $G = (V,E)$ is a mathematical structure composed of a set V of *vertices* (or *nodes*) and a set E of pairs of vertices called *edges* (or *arcs*). We assume that $(v,v) \notin E \forall v \in V$. A graph is said *directed* if each pair of vertices in E is ordered, and it is said *undirected* otherwise. Given an edge $e = (u,v) \in E$, we say that u and v are *incident* to e and that e is *incident* to u and v. Two vertices are *adjacent*, or *neighbors*, if they are incident to a common edge. Two edges are *adjacent* if they are incident to the same vertex. The *degree* of a vertex is the number of edges incident to it. A *path* in a graph G is a sequence of adjacent edges with no repeated vertices. A graph is *connected* if a path exists between every pair of vertices, and *disconnected* otherwise. It is also said *biconnected* (*triconnected*) if there exist two (three) paths between every pair of nodes, with no common vertices but the endpoints. A *cycle* is a path where the endpoint vertices are coincident. A *tree* is a maximal connected acyclic graph, and it

is *rooted* if one vertex is designated to be the *root*. A vertex with degree equal to 1 in a tree is called a *leaf*. A graph $G = (V,E)$ is *complete*, or a *clique*, if there is an edge $(u,v) \in E$ for each pair of vertices $u,v \in V$. A graph $G_0 = (V_0,E_0)$ is a *subgraph* of a graph $G = (V,E)$ if $V_0 \subseteq V$ and $E_0 \subseteq E$. A graph $G = (V,E)$ is *planar* if $|E| \leq 3|V|-6$. The *density* of a graph $G = (V,E)$ is the $|E|/|V|$ ratio. A graph with only few edges is said *sparse*, while a graph with a number of edges near to the maximum is said *dense*. The distinction between sparse and dense graphs is rather vague, and depends on the context. We assume that a graph with $|V|$ vertices is sparse if it has a number of edges near to or lower than that of a planar graph with $|V|$ vertices, and dense if it has a number of edges near to that of the complete graph with $|V|$ vertices. A *combinatorial embedding* of a graph G is a set of ordered lists of adjacent vertices, one for each vertex in G. A *clustered graph* is a graph $G = (V,E)$ together with a hierarchical grouping of the vertices in V.

Methods for Drawing Graphs

In its simplest form, a *drawing* Σ of a graph $G = (V,E)$ is a function that maps each vertex v to a distinct point $\Gamma(v)$ and each edge (u,v) to an open Jordan curve $\Gamma(u,v)$ with endpoints $\Gamma(u)$ and $\Gamma(v)$. More elaborated definitions are possible, for example vertices can be mapped to non-overlapping geometric shapes, and edges interconnect the boundaries of the shapes. However, the basic definition covers a large set of cases of interest, and it is often an intermediate step of methods for more elaborated drawing styles, so we will assume it in what follows. Edges are commonly represented as curved lines, polylines, or straight lines, and are depicted as arrows in directed graphs. Also, clusters of a clustered graph are commonly represented as nested boxes containing vertices.

Drawing a graph, in the context of this thesis, means to apply an algorithm for automatically produce a graphical representation of a graph with desired visual properties. *Aesthetic criteria* are often applied, some notable examples are the minimization of the number of edge crossings, the minimization of the drawing area, the minimization of the number of edge bends (if edges are visualized as polylines), the minimization of the variance of the lengths of the edges, the display of symmetries in the graph structure. Aesthetic criteria improve the readability of a drawing from different points of view but can be in contrast between them. For this reason, different drawing methods tend to privilege different criteria.

Constraints can also be applied to drawing algorithms, to enforce graphical properties that are specific of some domain. Differently from aesthetic criteria, which are general and apply to the entire graph, constraints operate on specific subgraphs. For example, an algorithm for drawing non-clustered graph could be modified and con-

strained to draw nodes of a clustered graph near to each other, if they belong to a same cluster. Some paths could be constrained to appear as a straight sequence of edges, avoiding bends. Some nodes could have an assigned position, that must be preserved in the layout. *Georeferenced graphs* are a notable example of this latter case, in which nodes have a geographical position. Depending on the application, algorithms can have some flexibility in applying constraints, fulfilling them only partially if this is necessary to obtain a layout.

There exist several approaches for producing a drawing of a graph. In the following we review some cases that are of interest for this thesis, more details can be found in [TDBET98].

Planarity oriented methods exploit properties of planar graphs. These graphs admit drawings without edge crossings, or *plane drawings*. A plane drawing subdivides the plane in topologically connected regions called *faces*, of which one, the *external face*, is unbounded. A combinatorial embedding of a graph is an equivalence class of its drawings, and plane drawings are grouped in *planar embeddings*. Efficient algorithms exist for finding a planar embedding of a planar graph, and a planarity oriented algorithm computes a drawing starting from a planar embedding. *Planarization* is an approach in which a non-planar graph is transformed into a planar graph, and an embedding is found for this graph. The final drawing of the original graph contains edge crossings. Planarity oriented methods work well for graphs that are planar or at least sparse. Constraints can be applied, but they can make finding an embedding and a drawing harder. For example, at the time of writing, it is not known the time complexity for computing a planar embedding of a clustered graph such that edges do not cross and avoid unnecessary crossings of clusters.

Hierarchical methods apply to acyclic directed graphs, and produce *layered* drawings. In this kind of drawing, vertices are displaced along parallel horizontal layers, and edges monotonically flow in one vertical direction. Layered drawings are sometimes referred to as *Sugiyama drawings*. This style clearly outlines hierarchies of nodes, hence the name, and it is often used for engineering diagrams (e.g. PERT, UML, Petri nets), where edges represent dependencies between nodes like precedences or requirements. Algorithms for layered drawings are composed of three phases. First, nodes are assigned to *layers* so that every edge is directed towards increasing layers. Then, each layer is ordered to reduce the number of edge crossings. With clustered graphs, the orderings must be constrained so that nodes belonging to a same cluster are consecutive. Finally, geometric positions along each layer are computed for vertices, such that the layer ordering is preserved and aesthetic properties, like spacing between nodes, are enforced. Finding layer orderings that minimize the number of edge crossings is a NP-hard problem, so often heuristics are used. Also, hierarchical methods only apply to acyclic graphs, therefore generic directed graphs

must be preprocessed for removing cycles. This usually means finding a small set of edges to remove or invert before making a drawing, and then restoring the modified edges.

Force-directed methods use a physical analogy to draw graphs. A graph is seen as a system of bodies with *forces* acting between them, and the resultant of this system of forces changes the positions of nodes. Algorithms iteratively update the positions, leaving the system converge at a configuration with locally minimal energy. A famous example of such an algorithm is the *spring embedder*. In this method, nodes are modeled as electrically charged particles, that repulse each other according to a force that decreases when their pairwise distance increases. Edges are modeled as springs, which attract pairwise connected vertices until a natural spring length is reached. Force-directed methods are among the most used for drawing graphs, for several reasons. First, they do not pose restrictions on the input and thus work on generic graphs. Second, they have a good scalability, and efficient implementations can draw graphs with thousands or even millions of vertices. Third, physical analogies are easy to understand, and algorithms are easy to implement. Fourth, it is easy to introduce constraints, by modeling them as additional forces that act on vertices and edges. Finally, these methods tend to produce aesthetically pleasant drawings, outlining symmetries in the graph structure and separating dense subgraphs from each other. Although their versatility, a drawback of force-directed methods is that a system of forces is a complex entity, and algorithms have only limited control on its evolution. The result is also very dependent on the initial configuration, for finding which there is not any general approach. These characteristics makes it hard to predict with accuracy the aesthetic properties of drawings produced with force-directed methods.

Part I

Visualizing Routing from Detail to Overview

Chapter 3

Visual Analysis of Routing Dynamics and Topology

This chapter describes an approach based on *Multiple Abstraction Levels* (see Chapter 1) for visualizing the structure and the dynamics of the routing graph produced by a selection of traceroutes. The tackled challenges are *Data Size*, *Dynamics*, and *Relation To Metrics*. The visualization approach exploits the fact that Autonomous Systems can aggregate parts of the network to furnish a more abstract view, and that consecutive snapshots of routing captured by a probe system can be used to show the evolution of routing over time with an animation. The work was based on experiments conducted on data from RIPE Atlas [atl15], and ended with the production of a prototypical tool, Radian, which was evaluated by users with an expertise in networking. A demonstrative version of Radian is available online [Rom]. A preliminary version of this chapter was published in [CDDS13].

3.1 Introduction

We present a tool, called Radian, for the visualization of traceroute data collected by a system of probes. The requirements of Radian were gathered interacting with several ISPs, within the Leone FP7 EC Project.

Radian works as follows. The user selects a set $\mathcal{S}$ of probes of a certain Internet measurement project (all the experiments described have been conducted using RIPE Atlas [atl15] probes), a target IP address τ, and a time interval $\mathcal{T}$, and obtains a visualization of how the traceroutes issued by the probes in $\mathcal{S}$ reach τ during $\mathcal{T}$.

A snapshot of Radian is in Fig. 3.1.

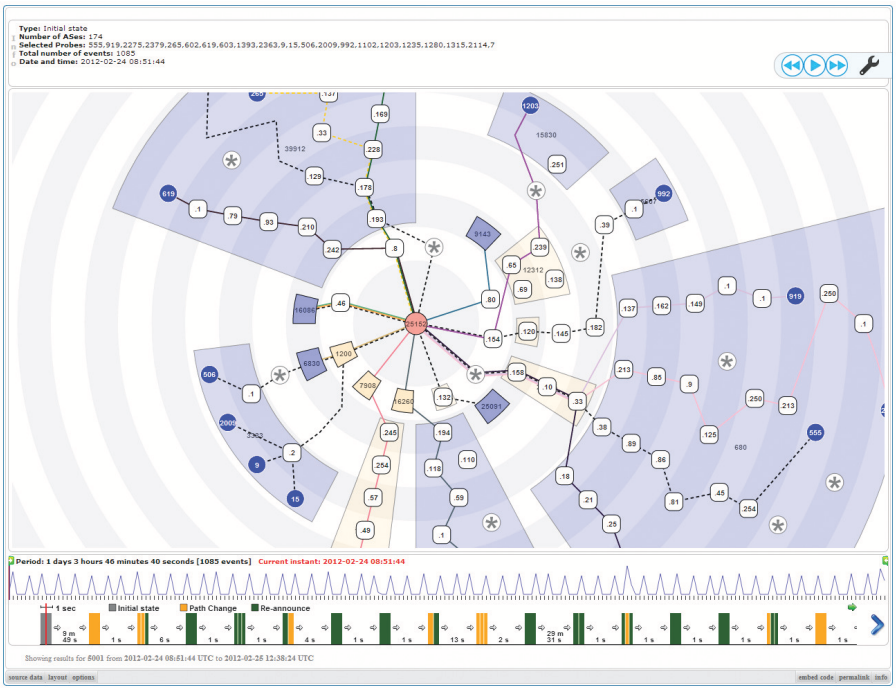


Figure 3.1: The main interface of Radian.

The chapter is organized as follows. Section 3.2 describes the scenario that originated Radian. In Section 3.3 we discuss how our contribution relates with the state of the art. Section 3.4 introduces basic terminology. Section 3.5 presents experiments that analyze the data visualized by Radian, discussing the results. Section 3.6 describes the adopted visual metaphor and the user interface of Radian. Section 3.7 illustrates the algorithmic apparatus devised for Radian. Section 3.8 describes a user study. Finally, conclusions are in Section 3.9.

3.2 Reference Scenario

There are several types of users who could be interested in monitoring the evolution of routing as seen by a system of probes. Network operators are interested to check the proper operation of their networks. Government authorities of a country need to

verify the quality of the Internet connection offered to the citizens. Cyber-security agencies look for abuses of the Internet that may implicate cyber attacks towards or from specific countries. Finally, researchers are interested to study the Internet routing because it is a complex, only partially understood system. Though this list of potential users who may benefit from a tool to monitor the evolution of routing, we focus our attention on *Internet Service Providers* (ISP), because we could outline clear user requirements by interacting with a few ISPs within the Leone FP7 EC Project. Among the activities of an ISP, we focus on three use cases that resulted from the discussion with our partner ISPs, and that could require to check the status of the routing: *troubleshooting*, *upgrade verification*, and *inter-domain consistency check*.

Troubleshooting copes with any unexpected dynamics that disrupted the normal operation of the network. For example, a router could stop forwarding packets due to an overload or a damage, making some network services unreachable or slow to respond. Or, a wrong configuration due to a human mistake could make the network be used in an inefficient way, letting many paths traverse a same node and overload it, while alternative nodes were available. Among the use cases we collected, troubleshooting is the one with less initial information, because the activity could have been started by a customer complain giving only partial and imprecise indications. The ISP may need to explore the data and look for some interesting pattern that is related to the problem, in order to find a subset of the network that was possibly the cause. Network devices are usually equipped with an alert system, which however is only capable to report the specific device where something anomalous was detected; the origin of the failure may be far from there, and understanding the full dynamics of how data flowed in the network could help better locate the cause. From this perspective, following traceroute paths link by link is a routing analogous of low-level debugging for a software application. Further, the traceroute becomes one of the very few analysis tool when a problem is originated in an external network, which is not under the control of the ISP and thus alert systems can not be of help. When troubleshooting a problem, it is often useful to compare routing changes to the variations of some metric of interest (e.g. the round-trip time), because it helps assign meaning to routing changes. For example, a sudden increment of the latency of a link could explain why many paths abandoned that link almost at the same time, right after the metric increment. Also, the path length is a parameter to consider, since longer paths tend to have higher latency.

An *upgrade* is any modification made to the network by the ISP to improve the existing services or to add new ones. When the action modifies the routing with the aim of improving the performance of one or more services, it is more precisely called *traffic engineering*. New routers and links could be added to extend the current network, or to add redundancy and make the network more resistant to failures and

high loads. Also, an upgrade can involve the configuration of a device. For example, a router could get configured to distribute the incoming traffic by following some load-balancing criteria (e.g. per-source, per-destination). After an upgrade the ISP needs to verify that the desired effect was obtained. Similarly to troubleshooting, this activity benefits from analysing the dynamics of the routing. By inspecting the paths followed by packets it is possible to check if a router was traversed by traceroutes issued by selected probes when it was expected. Also, metrics associated to the paths helps assess the outcome of the upgrade. For example, a sudden drop of the latency right after the transition from an old to a new path tells that the change was beneficial.

Inter-domain consistency check consists of verifying the relationship between the intended routing established by BGP announcements and the actual routing reported by traceroutes. The path of traversed ASes, obtained by associating the IP addresses of a traceroute path to their ASNs, tells whether the announced BGP policies were fulfilled. Checking the consistency between BGP and IP routing can be useful in a few cases. First, a change in the commercial relationships with other providers could require a change in the BGP links active with them, and the ISP must check that the modification was correctly implemented by the other parties. For example, the ISP could want to check the correct operation of a backup link towards another AS during an outage of the primary one. The ISP also needs to discover errors in the BGP policies themselves, which are usually set manually to let human operators supervise the process. Another interesting use case involves probes located outside the ISP network. The ISP could want to verify that a BGP announcement that it had sent was correctly received by the rest of the Internet, by seeing what paths the external probes followed to reach the ISP network.

3.3 Related Work

Tools for Traceroute Visualization

Because of its simplicity and effectiveness, the traceroute command attracted the interest of several researchers and practitioners that developed services for visualizing the traceroute paths discovered by executing one or more traceroutes. Broadly speaking, there are two groups of traceroute visualization systems: tools developed for local technical debugging purposes and tools that aim at reconstructing and displaying large portions of the Internet topology.

Several tools of the first group visualize a single traceroute and display the path on a map, showing the geo-location of the traversed routers. A few examples follow. Xtraceroute [Aug] displays individual routes on an interactive rotating globe as a series of yellow lines between sites, shown as small spheres of different colors.

GTrace [PN99] and VisualRoute [Vis] are traceroute and network diagnostic tools that provide a 2D geographical visualization of paths. The latter also features more abstract representations taking into account other information, e.g. the round-trip time between intermediate hops.

Tools of this kind are useful when the user wants to debug the path followed by a specific source-destination pair. In such a case, the visualization can help making sense of variations in some observed metric like the latency, by comparing it to the geographical positions of routers. However, this kind of approach does not support the exploration of data as collected from several sources at different instants, which could be exploited to find interesting routing dynamics that involved several paths. Also, as discussed in detail in Part III of this thesis, drawing geolocalized graphs is particularly challenging. Indeed, this kind of visualization gets easily cluttered as, for example, a traceroute can connect two dense metropolitan networks composed of many, relatively close network devices, through a long oceanic cable. Using the geographical positions to draw nodes poses strict restrictions on the layout, which cannot be further optimized for readability. In addition, Autonomous Systems tend to be highly distributed, so plotting hops at their geographical positions does not give a clear representation of the AS-structure of the portion of network traversed by traceroutes.

In the second group there are several tools that merge the paths generated by multiple traceroutes into directed graphs and show them in some type of drawing. The focus is on the topology of the traversed network, rather than the geographical positions of the nodes. In Argus [QoS], the length of an edge connecting two nodes reflects the one-way delay between those nodes. Also, nodes belonging to a same AS tend to be visualized close to each other. Similarly to what happens with a geographical visualization, relating the edge length to the values of a metric can produce a large amount of visual cluttering. In ThousandEys [Tho], paths are oriented from left to right, and nodes are positioned at fixed distances so that topological distances are explicit. Although the routing dynamics is not expressed in the visualization, the user can navigate time through an historical chart of a metric, and see the traceroute graphs obtained at a given instant. Several paths for the same source-target pair can be visualized at once, since the tool makes three attempts to perform traceroutes which could follow a different path each. Great emphasis is put on user interaction, with the possibility to collapse parts of the topology that are not of interest, or to outline elements that experienced a degradation in performance under a metric. The information available for a node include the AS to which the node belongs. Finally, Zenmap [Lyo] gives a radial view of the graph, with one focal node (e.g. the source of traceroutes towards several destinations, or the target of traceroutes from several sources) at the center of the drawing. Nodes are placed on concentric circles so to explicitly encode topological distances, and the thickness of an edge represents the value of latency.

Similarly to ThousandEys, the visualization can include several paths for the same source-destination pair, but the routing dynamics are not shown. The user can collapse the "children" of a node, that is, the nodes that reach the focal node through it.

Tools in this category present many desired characteristics that help users execute the use cases described in Section 3.2. First, merging traceroute paths into a graph gives users an approximated view of the network under test, which is useful to spot network behaviors that impacted several points of observation. Also, these tools include a graph layout that enhance the readability of the topology, the capability to reduce the visual complexity by hiding some parts of the visualization that are not of interest for a specific task, and a method to correlate the traceroute graph to the time trend of some metric. Some useful features, however, have been overlooked in current tools. Routing is a dynamic entity and traceroute graphs change over time, but tools present traceroute graphs more or less like static entities. Also, ASes are considered more as metadata attached to nodes, while they are a natural clustering of nodes and offer a way to look at the routing at a higher level of abstraction. Finally, the used algorithms to layout graphs are often general purpose and do not exploit the characteristics of traceroute data for improving the visualization, for example for reducing the number of edge crossings that make the topology harder to understand.

Visualization of Dynamic Graphs

Merging several traceroute paths together produces a graph. If the traceroutes are collected at different time instants, the resulting graph evolves over time and is called a *dynamic graph*. This model is particularly relevant for our work, because to the user it looks like an approximation of the real network on which the traceroutes were executed, and explicitly encoding the dynamics in the visualization helps him reason about how routing affected different parts of the network.

There exists a large amount of literature on the visualization of dynamic graphs, which is well summarized in a recent review paper [BBDW14]. According to that work, methods for visualizing dynamic graphs can be classified under these coordinates: (i) visualization metaphor; (ii) span of knowledge on data; (iii) representation of time; (iv) layout algorithm; and (v) modeling of transitions.

The two principal visual metaphors used for static graphs, *node-link* and *matrices*, are also applicable to dynamic graphs. However, we focus on the node-link style for an important domain reason: traceroutes are paths, and following paths is notoriously easier in the node-link representation. Also, this kind of diagram is intuitive for users in the computer network domain. Therefore, in the following we assume to work with this style.

Visualization methods for dynamic graphs are said *offline* if, at any time instant, future data are known and can be used to compose the current layout. Otherwise they are said *online*. The online setting is more versatile, because it can be applied to scenarios where data are known only up to the current instant (e.g. monitoring systems). On the other hand, the offline setting offers better opportunities to optimize the layout, by taking into account future data. Here we focus on the offline setting, because the use cases described in Section 3.2 only need past data.

Time is usually discrete for dynamic graphs and is represented as a sequence of instants or time slots. A different version of the graph, with different edges and nodes, is associated with each instant. *Animation* is a technique for representing the flowing of time that has been applied to dynamic graphs [EH00a], [DG02], [BS08], [BPF14], [BHJ+09]. It transforms the visualization of the graph from a given time instant (or frame) to another one, and is also called a time-to-time mapping. In its simplest form, an animation shows the evolution of a dynamic graph as a movie. The strengths of this approach are the intuitiveness, which users tend to like, and the implicit compactness of the representation, since the amount of used screen space does not depend on the number of graph updates. The main drawback is the difficulty for users to trace the history of an object across a long sequence of frames, because this requires him to rely on his memory of past frames [SLN05]. Also, the time consumed by an animation increases the time necessary to perform user tasks [APP11a]. In the *timeline* approach, time is represented through a spatial coordinate, and is thus a time-to-space mapping. Commonly this is done by juxtaposing several versions of the graph, one for each time instant, or by superimposing them along a fictitious z-axis towards the user. The juxtaposed style is also called *small multiples*. The timeline approach has been applied to the visualization of dynamic graphs in several forms [PRB08] [FAM+11] [FHQ11] [DE02]. The strength of this approach is the ease of following the evolution of an object along time, since time instants can be visually compared. On the other hand, discovering differences between two frames can be hard for users [BBDW14]. Also, the scalability of the technique is limited and depends on the number of shown frames, which is a trade-off between two parameters: the detail level of the frames, and the time granularity [BPF14].

It is still uncertain which of the two techniques, animation and timeline, is the best for representing dynamic graphs. Past studies gave contrasting results, depending on the experimental setting and the specific tasks. [SLN05] suggested to use animation for tasks involving the comparison of one or two time instants, and timeline otherwise. [FQ11] reported general better performance of timeline for both error rate and response time. [APP11a] found quicker response times for the timeline conditions, but higher accuracy of animation for tasks related to the appearance of entities. [BBL12] concluded that animation tends to reveal more findings on adjacent time steps, while

small multiples support the discovery of patterns lasting over long periods. In [BPF14] the two techniques are mixed, with small multiples used as a preview and a navigation system of an animated graph.

The layout algorithm adopted for positioning the graph elements has a big impact on the preservation of the so called user's *mental map*, which is the image that users have of the information. Preserving it requires to reduce the number of changes made to the visualization over time. The preservation of the mental map has been cited, since early stages, as a desired and fundamental property of a visualization of a dynamic graph [MELS95]. The intuition behind is that the cognitive load of the user for tracking the graph evolution is reduced by using an external visual representation, instead of relying solely on his memory. Researchers do not seem to have reached a consensus on the topic yet, since the results are mixed. While some works reported better user performance under the mental map condition [PHG07] [GEY12] [AP13b], others do not report significant differences [AP12] or even report negative performance [PS08] [SP08]. In [BBDW14], the authors conclude that the importance of preserving the mental map may have been overestimated in the past. However, a recent review paper [AP13a] suggests that preserving the mental map may actually support users in executing specific tasks. These must involve: (i) large graphs, with too many elements to be remembered; (ii) revisitation of past time frames; (iii) identification of the same elements in different frames; and (iv) navigation in the graph, for example for finding paths through specified elements. Depending on the layout strategy, the mental map can be preserved to various degrees. At one extreme, a *global optimization* strategy fixes the positions of all graph elements across time, perfectly preserving the mental map but possibly at the expense of a suboptimal space utilization within each frame. At the other extreme, a *local optimization* strategy changes the positions at every frame, optimizing the layout of single frames but possibly prejudicing the mental map.

Users can be helped in comparing different time instants of a dynamic graph by explicitly encoding transitions, i.e. by outlining the differences between two or more frames. If all time instants are aggregated in an unique image, the age of nodes and edges can be encoded by using colors or stippled lines [CKN+03]. However this greatly increases the visual complexity of the graph, and prevents the use of colors to encode other graph metrics. When only few frames are considered, this technique is also called *difference graph* and was applied in the timeline approach [AWW09] [ABHR+13] [HD12]. However, [APP11b] reports only marginal advantages in using difference graphs for outlining differences in a dynamic graph. In the animation approach, differences can be encoded with *staged transitions*, i.e. by executing changes in different groups depending on their type [FE01] [FE02] [BW04]. However, even when staged, many changes at the same time can be hard to follow [BPF14].

We conclude that many aspects of the visualization of dynamic graphs are not well understood yet, and none of the known techniques resulted significantly better than the others. Some works suggest to mix several techniques so to get their advantages and mitigate their disadvantages [RM13] [BPF14].

Layout Algorithms for Dynamic Clustered Graphs

As introduced in Section 2.1, each node in the Internet belongs to an Autonomous System (AS), and the network results partitioned into ASes. For this reason, algorithms for drawing graphs produced by traceroutes must deal with both dynamics and node clustering. There are several definitions of a graph that has a clustering on its nodes. In a *compound graph* clusters can be nested, and edges can be adjacent to both nodes and clusters. A *clustered graph* is a restriction in which edges are allowed only between nodes. Finally, a clustered graph whose clusters are not nested is called a *flat* clustered graph. Graphs produced by traceroutes fall in this last category.

Common aesthetic principles for drawing graphs like avoiding edge crossings and edge-node crossings also apply to graphs with a clustering. In addition, these graphs introduce the principle of avoiding unnecessary overlaps between clusters, avoiding unnecessary crossings between edges and clusters, and visualizing clusters as compact entities, so that the node partition is clearly represented. A drawing with such features is said *planar*, and testing the planarity of a clustered graph implies deciding if it admits a planar drawing [CD05]. The time complexity of the planarity test for clustered graph is still unknown, but several heuristic algorithms exist for producing a drawing. We focus on the case of dynamic graphs, which in addition to nodes and edges, require visual stability across changes also for clusters [BBDW14].

The two algorithmic styles that were experimented most in this domain are *force directed* and *layered* algorithms. The latter are sometimes also called algorithms for *hierarchical* layouts. Force directed algorithms produce a drawing by applying a system of attractive and repulsive forces to the graph elements, letting the system iterate until an equilibrium is reached at a point of local minimum. These algorithms are known to have good performance on large graphs, work on undirected graphs, and tend to produce drawings in which dense subgraphs are well separated. Algorithms for layered drawings work by first assigning vertices to ordered levels, or layers, so that the direction of edges is monotone with respect to the layer ordering. Then, the nodes of each layer are ordered for reducing edge crossings and putting near graph elements that are topologically related. Since the problem of finding an optimum ordering of a layer is NP-hard [TDBET98], often heuristics are applied. Finally, coordinates are produced that satisfy the assignment of nodes to layers and the ordering of each layer. These algorithm have lower scalability than force directed, but on sparse graphs pro-

duce drawings with fewer edge crossings. Also, hierarchical relationships between nodes are explicitly represented in the layering. For flat clustered graphs, [FT04] introduces a force directed algorithm that exploits virtual nodes and cohesive forces so to keep clusters compact and well separated. In [PB08], the approach of [DG02] is extended to compound graphs. In particular, the stability of the cluster structure over time is obtained by merging the cluster hierarchies corresponding to all considered time instants. This aggregated information is used to produce a super-layout of the graph through a force directed algorithm, which works as a template for drawing the single time instants. In [RPD09] a similar approach is used, but an algorithm for layered layouts is used instead of a force directed one. Algorithms for layered layouts work by assigning vertices to parallel levels, or layers, Also, during the animation from a frame to the next one, clusters that remain unchanged are collapsed so to focus on the parts of the graph subject to some dynamics.

General algorithms for dynamic compound and clustered graphs tend to be complex, because they try to ensure the visual stability of both graph elements and clusters over time. Often, these techniques exploit existing algorithms for drawing static compound and clustered graphs as underlying components to create the layout of single time instants. Several works exist on the problem of drawing a static graph with a clustering. Force directed approaches were designed for both clustered [WM96] [EH00b] [BAM07] and compound graphs [BM99], often applying the system of forces in a multi-level fashion across the hierarchy of clusters. For layered drawings, [SM91] first assigns nodes and clusters to levels, then orders elements on the layers through heuristics for reducing edges crossings, edge-cluster crossings, and maintaining clusters compact. It uses a so called *local layering*, in the sense that every cluster in the hierarchy has its own set of layers. [Rai05] presents a method for efficiently expand or contract clusters in a drawing with local layering without the need to redraw the whole graph. In [San96,San99] is described a technique for drawing compound graphs in a layered fashion that exploits a *global layering* for all nodes and clusters, producing more compact drawings than [SM91]. Finally, [FB04] studies the planarity problem for clustered graphs in a layered setting, showing that it is polynomial in specific cases.

In relation to the problem of drawing graphs produced from traceroutes, some considerations are required when choosing between force directed and layered algorithms. While the former pose few or no constraints on the input graph, they are iterative algorithms that converge at a local minimum, giving few guarantees on the quality of the produced drawing. Also, the direction of edges is usually not used in the process. On the other hand, layered drawings exploit edge directions and use them to make node hierarchies explicit, which can support user tasks. Also, layering gives an explicit visualization of topological distances in a network, making it possible, to some extent, to compare the length of different paths. But these algorithms

are also known to have lower scalability than force directed ones, and require the input graph to be acyclic. When this is not the case, the graph must be transformed to become acyclic, which may negatively influence the representation of hierarchies. In conclusion, when choosing between the two classes of drawing algorithms, the visual properties of produced drawings, the typical graph size, and the presence of cycles in graphs need to be evaluated.

3.4 Terminology

This section introduces some formal notation that will be used in the rest of the chapter. Also, the data of interest is characterized, by showing how we constructed graphs from raw traceroute paths.

Consider a time interval $\mathcal{T}$ and a set of probes $\mathcal{S}$. During $\mathcal{T}$ each probe periodically issues a traceroute towards a target IP address τ. A traceroute from probe $\sigma \in \mathcal{S}$ outputs a directed path on the Internet from σ to τ, called *traceroute path* or simply *traceroute*. Most of the times, a traceroute path is simple. However, there exist traceroute paths that are not simple and contain cycles. These correspond mostly to routing anomalies or to measurement anomalies. The origin and the incidence of those anomalies will be discussed in Section 3.5. Moreover, Section 3.7 describes an algorithm to deal with cycles in traceroute data. From now on, unless otherwise specified, we assume that traceroute paths are simple.

Each vertex of a traceroute originated from $\sigma \in \mathcal{S}$ is either a router or a computer. Vertices are identified as follows: (1) σ has an identifier assigned by the RIPE NCC; (2) vertices with a *public* IP address [rfc] are identified by such an address; (3) vertices with a *private* IP address [rfc] are identified by a pair composed of their address and the identifier of σ; (4) the remaining vertices are labeled with a "*" (i.e. an unknown IP address). For the sake of simplicity, consecutive vertices labeled with "*" are merged into one vertex. A vertex labeled with "*" is identified by the concatenation of the identifiers of its neighbors in the traceroute.

If a traceroute is available in Internet at time $t \in \mathcal{T}$, then it is *valid* at time t. A digraph G_t is defined at each instant $t \in \mathcal{T}$ as the union of all the traceroute paths valid at t produced by the traceroutes issued by the probes of $\mathcal{S}$. A digraph $G_{\mathcal{T}}$ is defined as the union of all graphs G_t with $t \in \mathcal{T}$.

Each vertex of $G_{\mathcal{T}}$ is assigned to a *cluster*, i.e. a label. The set of all clusters assigned in $G_{\mathcal{T}}$ is denoted by $\mathcal{C}(G_{\mathcal{T}})$, or simply $\mathcal{C}$ when there are no ambiguities. A vertex is assigned to a cluster as follows. (1) Each probe is assigned to the cluster that corresponds to the AS where it is hosted. (2) Each vertex identified by a public IP address is assigned to a cluster that corresponds to the AS announcing that address on

the Internet. This information is extracted from the RIPEstat [RIP] database and for some public IP addresses may occasionally be missing. (3) Each vertex v that is not assigned to a cluster after the previous steps is managed as follows. Consider the set P_v of all traceroute paths containing v. Denote by C_v a set of candidate clusters for v and initialize C_v to the empty set. For each traceroute $p \in P_v$ let μ (resp., ν) be the cluster assigned to the nearest predecessor (resp., successor) of v in p with an assigned cluster. If $\mu = \nu$ then μ is added to C_v. If after considering all $p \in P_v$ set C_v has exactly one cluster, v is assigned to it. If C_v contains more than one candidate, an inconsistency is detected and the procedure terminates prematurely. This corresponds to a nesting of ASes that is not meaningful in Internet. If C_v is empty, then v is not assigned to a cluster in this step. (4) Each remaining vertex is assigned to a corresponding *fictitious* cluster. We define V_μ as the set of vertices assigned to cluster μ.

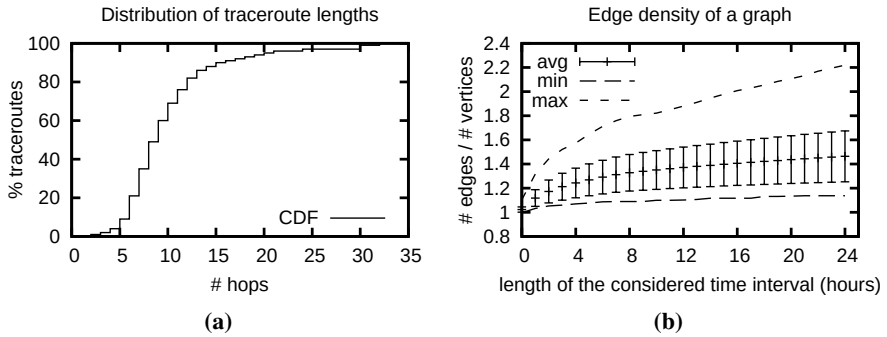
For any $t \in \mathcal{T}$ G_t can be visualized at different abstraction levels. Namely, the user can select a set $\mathcal{E}$ of clusters that are fully visualized and each cluster that is in the complement $\bar{\mathcal{E}}$ of $\mathcal{E}$ is contracted into one vertex. More formally, given the pair $G_t, \mathcal{E}$ the visualized graph $G_{t,\mathcal{E}}(V,E)$ is defined as follows. V is the union of the V_μ for all clusters $\mu \in \mathcal{E}$, plus one vertex for each cluster in $\bar{\mathcal{E}}$. E contains the following edges. Consider edge (u,v) of G_t and clusters μ and ν, with $u \in \mu$ and $v \in \nu$. If $\mu \neq \nu$, $\mu \in \mathcal{E}$, and $\nu \in \mathcal{E}$, then add edge (u,v). If both μ and ν are in $\bar{\mathcal{E}}$ then add edge (μ,ν). If $\mu \in \mathcal{E}$ ($\mu \in \bar{\mathcal{E}}$) and $\nu \in \bar{\mathcal{E}}$ ($\nu \in \mathcal{E}$) then add edge (u,ν) ((μ,v)). We define $G_{\mu,t}$ as the subgraph of G_t induced by V_μ. Analogously, we define $G_{\mu,\mathcal{T}}$ as the subgraph of $G_\mathcal{T}$ induced by V_μ. We define $G_{\mathcal{T},\mathcal{E}}$ as the union of the $G_{t,\mathcal{E}}$ for each $t \in \mathcal{T}$. Note that $G_{\mathcal{T},\emptyset}$ denotes the graph in which all clusters are contracted, while $G_{\mathcal{T},\mathcal{C}}$ denotes the graph in which all clusters are expanded and it is equivalent to $G_\mathcal{T}$.

3.5 Analysis of Data

This section describes the type of data that Radian visualizes, and some preliminary experiments that we conducted on such data. The results of these experiments influenced the design of our tool.

We selected traceroutes executed in nine months (from March 20th, 2012 to December 20th, 2012) by about 200 world-wide distributed RIPE Atlas probes towards 5 public targets. The probes performed measurements for the whole period, with variable frequencies for the execution of traceroutes ranging between one every minute and one every 30 minutes. Short-time disconnections were ignored. To perform our experiments, we preprocessed the data and generated $12,500$ random visualization scenarios of the kind to be displayed by Radian. Each visualization scenario is identified by a tuple $\langle \tau_j, \mathcal{S}_i, t_i, d_k \rangle$, where τ_j is a target, $\mathcal{S}_i$ is a set of 50 probes, t_i is a

time instant, and d_k is a time duration expressed as an integer number of hours. Such a tuple identifies a set of traceroute paths collected by probes $\mathcal{S}_i$ towards target τ_j, starting from instant t_i and for a period of d_k hours. We constructed the visualization scenarios as follows. We selected uniformly at random: 100 sets $\mathcal{S}_i$ $i = 1 \dots 100$ each consising of 50 probes, and 100 time instants t_i $i = 1 \dots 100$ in the above nine months interval. By repeating these objects for the five targets τ_j and for durations $d_k = k-1, k = 1 \dots 25$, we obtained the $12,500$ visualization scenarios. A visualization scenario of this type is an overestimation of real use cases, in terms of number of probes and length of the analyzed time interval. Often an ISP owns sufficient information collected from other sources that allow to restrict the length of the analyzed time interval. Also, at the time of writing, the number of probes deployed by the RIPE Atlas project in Italy is about 100. For each scenario we also computed graphs $G_{\mathcal{T},\mathcal{C}}$ and $G_{\mathcal{T},\emptyset}$ (see Section 3.4). In this analysis we will refer to $G_{\mathcal{T},\mathcal{C}}$ simply as the *graph*, and to $G_{\mathcal{T},\emptyset}$ as the *contracted graph*. Note that graphs with a duration equal to 0 have special semantics, since they approximate the state of the routing at a fixed time instant. On the other hand, graphs with a duration greater than 0 are composed of several states.



(a) **(b)**

Figure 3.2: (a) Cumulative distribution function (CDF) of the length of traceroute paths in our dataset. (b) Edge density of a graph in our dataset. Error bars report the standard deviation.

In Fig. 3.2a we plot a cumulative distribution function of the length of the traceroute paths of the dataset. That gives us a rough indication on the maximum distance between a probe in $\mathcal{S}$ and τ. The plot shows that traceroutes with more than 15 vertices are rare, which implies that a traceroute path can reasonably be visualized on the screen in its full extent.

Figs. 3.2b, 3.3, and 3.4 are related to the graphs computed for our visualization scenarios. On the x axis there is the time duration d_k ranging from 0 to 24 hours, while the y axis shows the value of a metric averaged over all graphs.

Fig. 3.2b shows that our graphs are quite sparse, even for large durations. In particular, graphs with duration equal to 0 are almost trees (density ~ 1), which was expected, since routing protocols forward data through a network by computing a spanning tree.

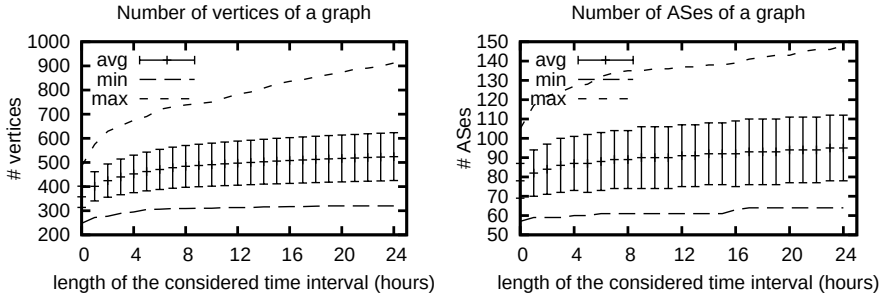


Figure 3.3: (a) Number of vertices of a graph in our dataset. (b) Number of ASes of a graph in our dataset. Error bars report the standard deviation.

Figs. 3.3a and 3.3b show, respectively, the number of vertices and the number of ASes in a graph. It is easy to see that these amounts are functions of the number of probes in our tests and the average length of a traceroute path. Visualizing a network with hundred of vertices and ASes on a screen is challenging. Even if the amount of screen space is enough to avoid clutter in the visualization, understanding such a network is a demanding cognitive task for a user. Therefore, an effective visualization requires a way to reduce the amount of details shown on the screen. From Fig. 3.3 we also draw conclusions on the impact that dynamics has on the size of a graph. We can see that the size of a graph increases with time, but such increment is not dramatic. Namely, passing from a duration of 0 hours to 24 hours increases the average number of vertices from about 400 to about 500.

When dealing with routing data one typically expects to find acyclic graphs, since cycles would prevent a correct routing in an IP network. One surprising result of our analysis is that, in graphs produced from traceroutes, cycles exist with quite a high probability. Fig. 3.4a shows that, in our dataset, 20% of graphs with $d_k = 0$

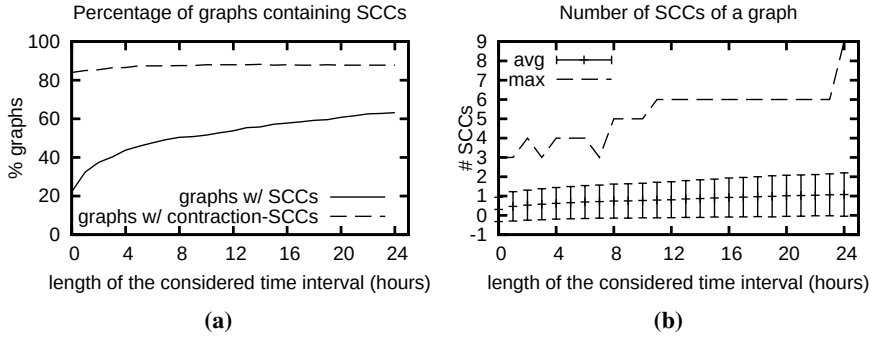


Figure 3.4: (a) Percentage of graphs that contain SCCs, and that contain contraction-SCCs when they are contracted. (b) Number of SCCs in a graph. Error bars report the standard deviation. The plot for minimum values is not shown, it is always equal to 0.

(i.e. almost static routing graphs) contain cycles. The percentage goes up to 60% for graphs with $d_k = 24$. Note that Fig. 3.4 refers to the *strongly connected components* (SCC) of a graph, which imply the existence of cycles in a graph but can computed in a more efficient way. A SCC is a subset of the vertices of a graph such that there exists a directed path between any pair of them. Even if a graph has a high probability to contain a cycle, Fig.3.4b shows that cycles in a graph are few. While cycles in a graph with some dynamics are somehow expected, since they can result from merging several routing states, finding cycles in a static graph is surprising. By analysing our dataset we discovered that such cycles are due to two factors. First, a static graph is only an approximation of the routing at a given time instant, since for each probe we select the latest measurement available and therefore the graph can actually span several time instants. Second, measurement errors can happen. The execution of the traceroute command is a process that takes a time in the order of seconds, and the routing can change right in the middle of the execution. A reported traceroute path can therefore be the fusion of two unrelated paths, which do not exist at the same time and induce the presence of a cycle. One additional property of SCCs in our dataset is that even if they are small on average, outliers can be large, up to 180 vertices. Such big numbers are due to the fact that SCCs are formed by interconnecting traceroute paths, whose length can reach 30 vertices. Interconnecting a few of them is sufficient to create large SCCs.

It is easy to see that a SCC that spans more than one AS induces a SCC also in

the contracted graph. What is less immediate is that the contracted graph can have its own SCCs. That is, there can exist a cycle in the contracted graph without that an identical cycle can be produced by contracting the ASes traversed by a cycle of the graph. We call *contraction SCCs* these SCCs of the contracted graph. BGP is a policy-based protocol that does not enforce the realization of a tree between the ASes, so loops between ASes are technically possible. However, one still expects not to find cycles, because commercially there is no point in letting data flow back and forth between two ASes, and customer-provider relationships between ASes are well known to exist in the Internet. On the other hand, Fig. 3.4a shows that contraction SCCs exist in the 90% of graphs in our dataset. Differently from the case of the graph, we do not consider the presence of contraction SCCs only a measurement error, because the policy-based nature of BGP actually allows for cycles. Also, an administrative authority (e.g. a company) can own more than one AS, for example by buying them from other companies. In that case the authority could configure its internal routing to allow traffic to flow through its ASes, since it does not have to pay anyone.

We draw several conclusions from the analysis in this section. First, the average length of a traceroute allows to visualize it on the screen in its full extent. The size of a routing graph depends mainly on the number of selected probes, and in a realistic setting it can be large enough to make cognitive tasks hard. For this reason, an effective visualization must allow to reduce the amount of details shown on the screen. On the other hand, the maximum level of detail is still required by the use cases that we considered in Section 3.2 for analyze specific parts of the network. Therefore, the user must be able to increment the amount of displayed details on specific parts of the visualization. Second, the sparsity of the graphs supports the use of graph drawing algorithms for layered layouts. With respect to the options described in Section 3.3, this kind of layout seems more adapt for visualizing traceroute graphs. We performed preliminary experiments with algorithms for layered layouts, discovering that crossing-reduction heuristics like those in [San96,San99] are quite effective. However, in our case the graph density is so low that often graphs are planar or quasi-planar, and hence planarity-based methods are more attractive. One drawback of algorithms for layered layouts is that they cannot handle cycles, which need to be treated first in order to produce a layout. Graphs produced from traceroutes have cycles with high probability, and present specific characteristics: cycles are few, large, and often look to the user like anomalies with respect to the natural direction of traceroutes. An algorithm for making traceroute graphs acyclic can exploit these properties for producing better layouts.

3.6 User Interface

This section presents the user interface of Radian, along with the motivations behind the design. Also, the supported user tasks are described.

Overview of the Interface

The user interface is shown in Fig. 3.1. It is composed of four main elements: the graph panel, the timeline panel, the controller panel, and the info panel. Through a sliding panel located in the upper right corner, the user can select the data to visualize, identified by a target τ, a set of probes $\mathcal{S}$, and a time interval $\mathcal{T}$. The interface presents at any moment the state of the data at a given time instant $t \in \mathcal{T}$, and with a set of expanded clusters $\mathcal{E}$. We detail the functionalities of the various panels in the following.

The *graph panel* displays the graph $G_{t,\mathcal{E}}$. Precisely, for each probe, the latest traceroute available before t is displayed. The graph is presented with a radial drawing centered in τ. All vertices and clusters that appear in at least one traceroute in $\mathcal{T}$ are in the drawing, including those that are not traversed by any traceroute at time t. Probes in $\mathcal{S}$ are represented as blue circles and labeled with their identifier. Vertices are represented as white rounded rectangles and labeled with the last byte of their IP address, or with a "*". Nodes are placed on concentric circles, with probes located at the periphery of the drawing and traceroutes directed towards the center. Clusters are represented as annular sectors and labeled with their AS number. Clusters in $\mathcal{E}$ enclose the nodes that are assigned to that AS, while clusters not in $\mathcal{E}$ are empty and have a fixed size. The light red cluster contains τ, clusters containing probes in $\mathcal{S}$ are light blue, and the remaining clusters are light yellow. Fictitious clusters are not displayed. Each traceroute path from a probe $\sigma \in \mathcal{S}$ to τ is represented as a colored curve from σ to τ passing through all intermediate vertices. Each curve, in fact, corresponds to a path in the graph $G_{\mathcal{T}}$. Traceroute paths are not simply merged and displayed in an aggregate fashion, since each of them has its own informative value and can change over time independently. For this reason, we explicitly represent all paths adopting a metro-line metaphor [Rob12], and draw them using different colors. Paths that change in $\mathcal{T}$ are represented with solid lines. For paths that do not change and thus represent static routes, we borrow a technique from [CDM+05a]. These paths are partitioned into sets such that each set determines a tree on the graph, and each tree is depicted with dashed lines and a distinctive color. This has the effect of reducing the number of lines in the drawing, while preserving the routing information of each probe. The user can interact with the graph in several ways. First, he can change the current time instant t that is visualized. Path differences between the two instants are shown with an

animation, which continuously morphs each path from its initial position to the final one. Morphing path are outlined with a thicker curve during the animation, so to be more visible. A new route that was previously unavailable is shown in the animation with a gradually appearing path, while a disconnection is shown with a gradually fading path. A path is also outlined when the pointer hovers on it, and, similarly, hovering on a vertex outlines all paths passing through it. Finally, the user can change the set $\mathcal{E}$ of expanded clusters. Double-clicking an expanded cluster collapses it and removes if from $\mathcal{E}$, and vice versa. Vertices in a collapsed cluster are removed from the visualization.

The *timeline panel* is in the lower part of the window and provides an overview of the trend in the number of route changes over time. It features a red cursor that points at the current time instant. The timeline panel and the graph panel are linked views, and the visualized graph corresponds to the time instant selected in the timeline. The user can move the cursor to change the current time instant. Additional timelines can be added below the main one, each regarding the trend in the value of a *metric* with respect to a probe. The user can add and remove the metric timeline of a probe by double-clicking on that probe in the graph panel. All timelines have the same width and represent the same time interval $\mathcal{T}$, allowing for direct comparison.

The *controller panel* is located in the upper right corner of the interface and is used to control the animation. It is modeled following the metaphor of a video recorder, and presents buttons that activate the typical functions play, pause, step-backward, and step-forward. During an animation the user is presented with a sequence of animation steps, each regarding a single path change and with a short pause between two changes. The cursor in the timeline panel moves accordingly, which also continuously updates the visualization in the graph panel. The duration of an animation between two time instants t_1 and t_2 is proportional to the logarithm of the elapsed time between the two instants, which gives an approximate perception of elapsed time while limiting the overhead on the total animation time.

Finally, the *info panel* is in the upper part of the window. When the cursor hovers on an element of the graph, the info panel shows all the available information about that element, i.e. an AS, a vertex, or a path.

Our choice to make a topological visualization, discarding geographical visualizations of traceroute data, comes from the conclusions drawn in Section 3.3. Our use cases require a graph visualization, which get easily cluttered when geographical positions for nodes are applied. Also, understanding the structure of the network at the AS level is more interesting for understanding routing policies, and this would be disrupted by a geographical visualization since AS are usually distributed. Finally, *anycast* addresses are assigned to more than one physical device and could not be mapped to a single location. The node-link metaphor was chosen for representing the

graph because, as explained in Section 3.3, it is intuitive to networking users, since it looks like the real network on which traceroutes were performed. The graph is visualized with a radial layered layout because it is sparse (see Section 3.5), and this style of drawing is notably effective for visualizing sparse hierarchical graphs (see, e.g., [YFDH01]). Also, in our visualization the focus is correctly put on the target, preventing the risk of giving too much importance to specific probes due to a privileged geometric position. Finally, having all traceroutes flowing in a same direction helps comparing their lengths. The choice of animations, instead, was less immediate. Although users generally like them, Section 3.3 outlined how a limit of this technique is that users needs to rely on their visual memory in order to track graph updates. On the other hand, the main alternative consisting of small multiples suffers from too low scalability to support our use cases. In a time span of a few hours hundreds of route changes can happen depending on the size of the network, which would produce too many small multiples to fit in the screen. For this reason we adopted animations, which support very long time spans. We applied several techniques to mitigate the limits of animations. First, we chose to enforce the preservation of the mental map (see Section 3.3 for a discussion). In particular, vertices preserve their relative orderings along and across layers, which is similar to the preservation of horizontal and vertical orderings of [MELS95]. Section 3.7 describes in detail the algorithms for obtaining such property. With this constraint, the user can easily identify vertices in different time frames, focusing his attention only on edge updates and how traceroute changed their paths over time. Visualizing also nodes that are not traversed by a traceroute at a given time instant (but that are traversed at some different instants) is consistent with this choice, and also furnishes to the user hints on the fact that a given part of the network is subject to dynamics at some time instant. Finally, outlining with thicker lines traceroute paths that change during an animation is a form of transition encoding. The visualization of ASes as boxes that enclose vertices satisfies the user expectation that ASes represent a partition of the network. Also, this organization offers an AS-level view of the network, which is an abstraction useful in all use cases described in Section 3.2. Finally, we exploited this representation to let the user collapse an AS and hide its content. This allows to focus the attention on the detailed dynamic of only few ASes of interest, while maintaining a high level overview of the rest. The results is a reduction of the cognitive load for understanding graphs that, how showed in Section 3.5, can be relatively large. The possibility to arbitrarily collapse and expand ASes poses challenges in the preservation of the user mental map. Indeed, if the ordering in which clusters are expanded or collapsed impacts the layout in an uncontrollable way, the layout stability is compromised. There is no practical way to check all the possible sequences of expansions and contractions, since their number is factorial in the number of clusters. For this reason, the algorithm in Section 3.7 en-

forces that $G_{t,\mathcal{E}}$ has the same layout independently from the ordering in which clusters are inserted in $\mathcal{E}$.

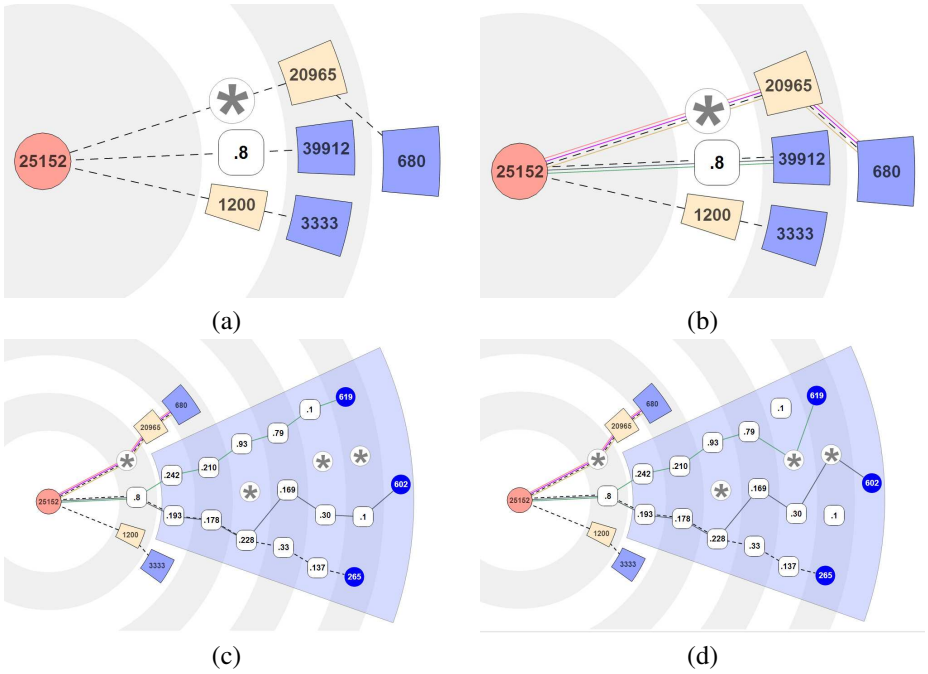


Figure 3.5: Details of the interactive features of our visualization. (a) A graph $G_{\mathcal{T}'}$ relative to a target τ, a set of probes $\mathcal{S}$, and a time interval $\mathcal{T}'$. All paths in $G_{\mathcal{T}'}$ are static and all clusters contracted. (b) A graph $G_{\mathcal{T}''}$ relative to τ, $\mathcal{S}$, and $\mathcal{T}''$ ($|\mathcal{T}''| > |\mathcal{T}'|$). Some paths are dynamic and all clusters are contracted. (c) $G_{\mathcal{T}''}$ with an expanded cluster. (d) $G_{\mathcal{T}''}$ at a different time instant.

Fig. 3.5 contains various details on how the interaction with the visualization works. A graph with static paths and no expanded clusters is presented in Fig. 3.5(a). It is related to a target τ, a set of probes $\mathcal{S}$, and a small time interval $\mathcal{T}'$. Note that some vertices are not enclosed in any cluster: they belong to fictitious clusters. From this figure we can see what ASes provide connectivity to reach the target, namely 1200 and 20965. A graph for τ, $\mathcal{S}$ and $\mathcal{T}''$ ($|\mathcal{T}''| > |\mathcal{T}'|$) is presented in Fig. 3.5(b). Some dynamic paths are visible. The same graph is presented in Fig. 3.5(c) with one expanded cluster. Note how the ordering of clusters and vertices on the radial layers

is preserved. In this figure the length and structure of the paths from each of the three probes 619, 602, and 265 is clearly visible. Fig. 3.5(d) shows the same expanded graph at a different time instant. The intermediate vertices of two paths are different, that is, we can see how the route changes affected the paths of probes 619 and 602.

User Tasks

This section describes the tasks that a user can execute through the interface of Radian. Analyzing the use cases outlined in Section 3.2, we found out that they could be supported by executing simpler user tasks, that are classified in two groups: tasks devoted to understanding the structure of the routing and of the network at a given time instant, and tasks devoted to understanding how the routing evolved over time in a given interval. The first group of tasks are executed on the routing as captured at a single time instant. The user may know the time instant in advance from other sources (e.g. customer tickets), or he may be analyzing several instants by sampling in an interval that is an approximation of the desired instant. The second group requires the user to compare different time instants and make use of animations. Also, the comparison of routing changes with network metrics is often involved. For brevity, we will refer to the two groups as *topology tasks* and *dynamics tasks*, respectively. Also, we will refer to use cases Troubleshooting, Upgrade Verification, and Inter-domain Consistency Check as UCT, UCV, UCC, respectively.

The list of the topology tasks follows.

Find what nodes are traversed by a probe This task is fundamental for all use cases, since it implies discovering the route followed by a probe at a given instant to reach the target, which is the final product of the routing. The task is accomplished by following the colored curve across the graph, from the probe to the target, and see what nodes are intersected.

Find the topological distance to the target This tasks implies to discover the number of nodes to traverse to reach the target. Intuitively, shorter paths can provide better performance, and the length can be exploited to compare alternative paths. This is useful in UCT and UCV, either to find out the reason for experienced bad performance, or to spot in advance long paths that could be future causes of issues.

Decide if a node is reachable This task supports UCT and UCV. A node is reachable from a probe if the curve from that probe intersects it. An unreachability can have different meanings. When referred to the target, the task has the fundamental objective to discover if the services of the target were accessible from

other parts of the network. Depending on how far from the target paths are interrupted, it could either mean that the routing was wrong or the target was not working properly. When referred to an intermediate node, this can either appear in the drawing or not. If it does, it means that it is reachable at least in some instant, otherwise a "*" node appears in its place. In UCV, the task is executed after a routing configuration setup to check if paths are traversing a node that was supposed to be. In UCT, an unreachability could mean that the node is experiencing a fault. Note that an unreachability does not necessarily imply a fault, since routers can be configured to not respond to traceroutes, for security reasons. An ISP usually knows, in its own AS, which routers respond and which do not.

Find what probes traverse a node This task is useful to understand what parts of the network depend on a node for reaching the target. In UCV this is to check if the designed node for connecting a subset of probes is actually doing it. In UCT, a node shared by several probes that experienced reduced efficiency is possibly the cause, and a start point for a deeper analysis. The task is executed by following all curves traversing the node, from that node back to the probes.

Find single points of failure This task is somewhat in the middle between UCT and UCV. The objective is to find in advance nodes that, in cause of a fault or an overload, would impact a large part of the network. The task is executed by finding nodes with many traversing paths, that is, with many incoming graph edges.

Find load balancers Even if load balancers are more related to dynamics tasks, their presence can be spotted from a single time instant with Radian. A load balancer induces many nodes near to each other on a same layer, with only one of them traversed by a path. This happens because the interface always shows all nodes that are traversed by some path in some time instant, which is a hint that the path may be throttling between the available alternatives during time. Finding load balancers is important in UCT and UCV for discovering if a load balancer that was known to exist is actually working.

Decide if two probes are treated equally In UCT, the ISP may receive contradicting feedbacks from its customers regarding the use of a remote service. While some could experience low performance or even an outage, others could not. By comparing the paths of different probes towards the target, the ISP may discover that intermediate nodes treat probes differently, for example depending on their estimated geographical location. If this happened in another AS, it could imply

that a service-level agreement between the two providers was violated. In UCV the difference could be desired for providing different levels of service to the customers, and the ISP need to check if premium customers are privileged by following paths to the target with higher performance.

Find the ASes that connect the target Understanding what ASes are traversed by a probe to reach the target means to reason about the routing and the network at a higher abstraction level, and it is the fundamental task behind UCC. It is accomplished by checking the AS to which each of the traversed node of a path belongs. As a result of this task, from a traceroute path, the user can induce a corresponding path of ASes. Such path can be compared to the BGP announcements to check if all the providers on the path are fulfilling them. Finding the traversed ASes has also important security implications. For example, an ISP could want to check if security critical data generated by its AS are routed by mistake through the ASes of extra-national organizations.

Find cyclic relationships between ASes As discussed in Section 3.5, cycles between ASes are unusual but possible. If there is a cycle between two ASes, the user concludes that they probably belong to a same organization. In case of performance issue in one of the ASes forming the cycle, this information helps assign the responsibility to a specific organization.

The list of the dynamics tasks follows.

Find probes with path changes As described in Section 3.6, routes that are not subject to dynamics in the selected time interval are depicted with dashed lines. That is, from a single frame, the interface of Radian tells whether a probe changed its path at some time instant. This is a general feature supporting all use cases in which it is necessary to understand the routing evolution.

Find failing nodes The paths of several probes could abandon a specific node at the same time. This is clearly outlined in the graph panel, which with an animation shows the corresponding paths morph and stop traversing that node. In UCT, this could be the sign of a hardware failure on that node. In UCV, the node could have been turned off for maintenance, and the probes redistributed on different paths for keeping the service.

Find high routing activity In UCT, it is necessary to discover when the routing changed, first or after that a known event happened. For example, this is the case when the user is working on a customer ticket that complained about degraded performance or lack of service starting from a given hour of the day. In UCV, the

same analysis is done to check if the routing changed in correspondence of a configuration change on the devices. The timeline panel gives an overview of the distribution of path changes within the selected time interval. The user, for example, may find out that nothing happened until a given instant, and therefore focus on what happened after. Also, from the chart periods with a high level of activity are easily spotted and represent a starting point for a deeper analysis.

Find repetitive phenomena Repetitive routing dynamics are shown in the timeline panel as spikes of activity happened with a certain regularity. In UCT, the user could start the analysis from a given time instant were suspect dynamics happened, and then discover that more activity is present in the hours before or after that instant with regularity. For example, the network of an ISP could be not able to respond with acceptable performance to a traffic load in the early night hours, caused by many customers connecting after work from their homes. This would be the cause of many path changes happening around that time for several days, as the result of intense load balancing.

Check if a BGP backup link is operational An ISP can be connected to the Internet through several BGP links for redundancy, either with a same partner ISP or with several ISPs. BGP policies can be set in such a way that, in case of a fault on a link, a redundant and usually unused link becomes operational. In UCC, the user can follow the evolution of paths traversing a failing BGP link and check that, in correspondence of the failure, all of them started traversing the backup link.

Decide if a path change improved the performance After the user has spotted a path change of interest in the timeline panel, this usually is not enough to decide if it was positive or not. In all use cases, the user is interested to check the value of metrics in correspondence of the change, to see if there was an improvement. This is done by selecting one or more probes in the graph panel, which makes to appear a line chart with the metric trend along time for each probe.

Decide if there is a correlation between routing and metrics Several probes could change their paths in a similar way, for example by stopping traversing a specific node in favor of an alternative. Since the involved probes do not all follow the exact same path, they could respond to the change in different ways with respect to metrics. The user can check the metric charts of the involved probes in correspondence of the routing change, and see if the impact was similar for all or most of them. For example, a sudden improvement could mean that the failing node was overloaded, and the routing decided to replace it with alternatives that have better performance.

3.7 Algorithms

This section is devoted to the algorithms that Radian uses for visualizing traceroute graphs. In particular, the first described algorithm produces a layout of a traceroute graph. This algorithm assumes the graph to be acyclic. The second described algorithm removes cycles from a traceroute graph, so that the layout algorithm can be applied.

In what follows, we make these assumptions:

- Every traceroute path contains the target vertex. When missing, it is artificially added to it.
- Paths in a graph have the reversed direction with respect to traceroutes, that is, they go from the target to a source. This complies to a convention commonly used by algorithms for layered layouts.
- A layering of a graph is a mapping from vertices to integer number, called layers. The target is assigned to the layer with the lowest number, and edges are directed towards increasing layer numbers.

Layout Algorithm

According to the conclusions of the analysis in Section 3.5, the layout algorithm of Radian produces layered layouts and is planarity oriented. For our purposes an interesting reference is [Bac07], which constructs radial drawings adapting techniques of the Sugiyama Framework. Unfortunately, it does not deal with clusters. The algorithm in [FB04], which extends the one described in [DBN88], inspired part of our work. However, it proposes a clustered planarity testing algorithm, while we rather need an algorithm for clustered graph planarization, and [FB04] is not easily extensible for this purpose (neither is the algorithm in [BDM02] that is not suitable for hierarchical drawings). For these reasons we devised a new algorithm to produce clustered hierarchical drawings, as a planarization-oriented variation of [FB04]. In [Rai05] an algorithm is proposed for the expansion/contraction of clusters of hierarchical drawings, building on [SM91]. Unfortunately it uses local layering for vertices, while global layering [San96,San99] is more suitable for our needs because it produces more compact drawings. For this reason we devised a new algorithm for expanding/contracting clusters that is based on global layering. Differently from [Rai05] it is not a local update scheme, i.e. it computes a new drawing for the whole graph at each interaction. The lower time efficiency is negligible because the graphs commonly handled by *Radian* are small, from the point of view of computation.

At a high level, our algorithm works as follows. We pre-compute a hierarchical drawing Γ_0 of $G_{\mathcal{T}}$ that integrates all the traceroutes in $\mathcal{T}$. In that drawing all clusters are expanded. The layout is computed in such a way to have few crossings involving connections between clusters. The quality of the layout inside the clusters is considered with lower priority. Moreover, the quality of the drawing of edges that are part of many traceroutes in $\mathcal{T}$ is privileged among the edges of $G_{\mathcal{T}}$. The drawing computed for each cluster is stored and reused in any drawing where that cluster is expanded. The hierarchical drawing is mapped to a radial drawing with a suitable coordinate transformation. Changes in the drawing due to an expansion or contraction of a cluster or a change in traceroutes are visualized with an animation. At any instant $t \in \mathcal{T}$ only the traceroutes that are valid in t are displayed.

What follows gives more details on our the algorithmic framework.

In a preprocessing step several information are computed on $G_{\mathcal{T}}$ that will be used for actual drawings. Given any $G_{\mu,\mathcal{T}}$, a vertex is a *source* (*sink*) of $G_{\mu,\mathcal{T}}$ if it is the last (first) vertex of $G_{\mu,\mathcal{T}}$ encountered in some traceroute path. Each graph $G_{\mu,\mathcal{T}}$ is augmented with extra vertices and edges so that all the longest paths from a source to a sink have the same length. The added vertices are called *fictitious vertices* of μ and ensure that, given an edge $(u,v) \in G_{\mathcal{T}}$, $u \in \mu$, $v \in \nu$, $\mu \neq \nu$, clusters μ and ν do not share a layer in any drawing of $G_{t,\mathcal{E}}$. Moreover, they force edges that leave a cluster by spanning several layers to be routed inside that cluster. A *μ-drawing* is pre-computed for each $G_{\mu,\mathcal{T}}$. It consists of (i) assigning vertices to layers so that all edges are between consecutive layers and (ii) computing a total order for the vertices of each layer. A partial order $\prec$ is computed for clusters, such that for any two clusters μ and ν with $\mu \prec \nu$, the vertices of μ appear to the left of the vertices of ν for any drawing Γ where μ and ν share one or more layers. This helps preserve the mental map during expansions/contractions. The preprocessing step requires to compute a drawing Γ_0 of $G_{\mathcal{T}}$ with all clusters expanded. Γ_0 gives the information needed to compute a μ-drawing for each cluster and a partial order $\prec$ for clusters. The algorithm to compute Γ_0 is similar to that in [FB04], where a PQ-tree [BL76] is used to order vertices along the layers of the drawing. Our PQ-tree is initialized with a spanning tree of $G_{\mathcal{T}}$ and incrementally updated with the remaining edges that induce ordering constraints. An edge is added only if it does not produce a crossing (i.e. the PQ-tree does not return the null tree). A rejected edge will produce crossings in Γ_0. Edges are added with priority given by their aesthetic importance: namely, they are weighted by the number of traceroutes that traverse them in $\mathcal{T}$. As an implementation detail, we actually compute a total order for clusters to represent a partial order $\prec$. Such order is produced by a DFS visit of the embedded spanning tree of $G_{\mathcal{T}}$. The tree has an embedding induced by the layer orders produced by the PQ-tree algorithm, and the children of a vertex are visited in clockwise order. Intuitively, we preserve the

geometric left-to-right order for clusters from Γ_0, and reuse it to produce a drawing of any $G_{t,\mathcal{E}}$.

The result of the preprocessing is used to compute a drawing $\Gamma_{\mathcal{T},\mathcal{E}}$ of $G_{\mathcal{T},\mathcal{E}}$, as detailed in the following. First, note that once $\Gamma_{\mathcal{T},\mathcal{E}}$ is computed, we display, for any $t \in \mathcal{T}$ all the vertices of $G_{\mathcal{T},\mathcal{E}}$ but only the edges of $G_{t,\mathcal{E}}$. This is done to preserve the mental map of the user, using $\Gamma_{\mathcal{T},\mathcal{E}}$ as a "framework" that "hosts" the drawings of each instant. First, a layering of $G_{\mathcal{T},\mathcal{E}}$ is computed such that for each vertex the distance from τ is minimized. Also, dummy vertices, called *fictitious vertices* of $G_{\mathcal{T},\mathcal{E}}$, are added so that each edge spans two consecutive layers. Vertices are horizontally ordered on each layer such that: (i) $\prec$ is enforced; (ii) for each cluster μ of $\mathcal{E}$, the orders on the layers of its μ-drawing are enforced; (iii) the fictitious vertices of $G_{\mathcal{T},\mathcal{E}}$ are placed in such a way to have few crossings. In particular, they must not be interleaved with the vertices of any cluster, that is, the vertices of each cluster must be consecutive on every layer. For this reason, each fictitious vertex is assigned to a new fictitious cluster, which is inserted in the partial order $\prec$ in an intermediate position between the endpoints of the edge it belongs to. Finally, the ordered layers are used to assign geometric coordinates to vertices. The width of each cluster μ is computed as follows. Consider the layer containing the largest number of vertices assigned to μ. The cluster is assigned a width proportional to this number. Vertices of μ are assigned horizontal coordinates such that they can be enclosed by a rectangle with height proportional to the number of layers assigned to the vertices of μ and width equal to the width of μ. We avoid intersections between enclosing rectangles by means of an auxiliary directed acyclic graph where vertices are clusters of $G_{\mathcal{T},\mathcal{E}}$ and edges are selected from $\prec$ depending on which pairs of clusters share a layer in the current layering of $G_{\mathcal{T},\mathcal{E}}$. Edges are weighted based on the widths of the clusters they are incident to. The total width of the drawing is given by the longest path in this graph. The above is applied recursively to compute the horizontal spacing among all clusters. The vertical coordinate of a vertex is equal to the one assigned to its layer, which is proportional to the index of that layer in the total order of layers.

Going back to the state-of-the-art, concerning restrictions $R1$, $R2$ and $R3$, described in [FB04], that a planar clustered hierarchical drawing must obey, drawings produced by our algorithm satisfy $R1$ and $R2$, while we consider $R3$ too restrictive for our application. Restriction $R1$ is satisfied in the preprocessing step by merging, for each cluster, all sources into one vertex. At the end of the algorithm, the PQ-tree contains a spanning tree of the graph with merged sources, which has the effect to keep the vertices of each cluster consecutive on any layer. However. although effective, a drawback of this technique is that it can create edge crossings even if the instance admits a planar drawing. The reason is that, for a vertex resulting from merging the sources of a cluster, the PQ-tree does not preserve the consistency between the orders

of its incoming and outgoing edges. When the vertex is split again in the embedded graph to reconstruct the original sources, crossings can result in the layer that precedes or follows the one of the sources. We mitigate this effect by post-processing the embedding produced by the PQ-tree. Namely, the position in each layer of the vertices that represent probes is preserved, while the position of any other vertex is computed by a single bottom-up sweep of the layers that uses a barycentric positioning. The technique is inspired by those commonly used in algorithms for Sugiyama-like layouts [TDBET98]. Restriction *R*2, as shown in [FB04], is automatically satisfied for the initial drawing Γ_0, and is satisfied for any drawing of $G_{t,\mathcal{E}}$ by exploiting the partial order $\prec$.

To obtain a radial drawing, the geometric coordinates of vertices so computed are transformed as follows. Each vertex is placed on the perimeter of a circle centered in an arbitrary fixed point and having radius equal to the vertical coordinate of the vertex. Then the horizontal coordinate of the vertex is mapped to a circular coordinate on the perimeter of that circle. The perimeters of clusters are mapped with a similar radial transformation. An edge (u,v) is drawn either as a straight segment or a curved arc, depending on the angle it must sweep to connect vertices u and v. Note that in our setting each edge connects only vertices in two consecutive layers, hence a curved edge can be drawn only in the space between these layers.

Handling Cyclic Graphs

As pointed out in Section 3.5, the contraction of clusters in a graph can create cycles. Namely, the graph $G_{\mathcal{T},\mathcal{E}_2}$ that is produced by contracting a cluster in graph $G_{\mathcal{T},\mathcal{E}_1}$ can contain a cycle that does not exist in $G_{\mathcal{T},\mathcal{E}_1}$. For this reason, given a graph $G_{\mathcal{T}}$ with clusters $\mathcal{C}$, an algorithm for removing cycles must ensure that $G_{\mathcal{T},\mathcal{E}}$ is acyclic for any set of expanded clusters $\mathcal{E} \subseteq \mathcal{C}$. A naive approach like checking all possible sets of expansions is unsuitable, since it requires to check a number of graphs that is exponential in the number of clusters. However, Theorem 1 states that it is sufficient to check a limited number of graphs.

Theorem 1. *There exists a cycle in $G_{\mathcal{T},X}$ for some $X \subseteq \mathcal{C}$ if and only if there exists a cycle either in $G_{\mathcal{T},\emptyset}$ or $G_{\mathcal{T},\mathcal{C}}$.*

Proof: The proof in one direction is trivial: $\emptyset$ and $\mathcal{C}$ are sets of expanded clusters, therefore if $G_{\mathcal{T},\emptyset}$ or $G_{\mathcal{T},\mathcal{C}}$ contains a cycle then some $G_{\mathcal{T},X}$ does. For the other direction, assume that $G_{\mathcal{T},X}$, with $X \neq \emptyset$ and $X \neq \mathcal{C}$, contains a cycle. If all vertices of the cycle belong to a same cluster then that cycle also exists in $G_{\mathcal{T},\mathcal{C}}$ and the theorem holds, so assume that it spans more than one cluster. We proceed by induction on the number of expanded clusters (i.e. the number of clusters $\mu \in X$) that are traversed by

the cycle. If every vertex μ of the cycle represents a contracted cluster then that cycle also exists in $G_{\mathcal{T},\emptyset}$ and the theorem holds. Then, assume that the cycle traverses n expanded clusters, while the other traversed clusters are contracted. Let μ be one of the traversed expanded clusters. The directed cycle intersects the boundary of μ an even number of times, half of the times entering it and half leaving it. Given a pair of entering and leaving intersections that are consecutive along the cycle, let path π be the part of the cycle that goes from the entering intersection to the leaving one and that traverses only vertices of μ. We identify the entering and the leaving intersections respectively with the first vertex i_e and the last vertex i_l of π. There exists a path $\hat{\pi}$ from i_l to i_e that is part of the cycle and traverses only vertices not in μ, or the cycle would not exist. Contracting μ replaces it with a single vertex μ, which has an incoming edge and an outgoing edge for each pair of entering and leaving intersections (i_e, i_l), that is, the number of intersections of the cycle with μ is preserved after the contraction. Together with the aforementioned paths $\hat{\pi}$, these edges form a cycle in the graph after the contraction of μ. Such cycle traverses $n-1$ expanded clusters, therefore the theorem holds by the inductive hypothesis. □

The result of Theorem 1 is that, to make a graph $G_{\mathcal{T}}$ acyclic, it is sufficient to make acyclic the corresponding graph with all clusters expanded and the corresponding graph with all clusters contracted, called in the following respectively the *graph* and the *contracted graph*.

Two techniques exist to remove cycles from a directed graph: deleting edges from the graph or reversing their directions. Reversing edges is more correct for our application, because all edges must be visualized by Radian but removed edges would not influence the layout. Also, it is desiderable to reverse few edges, so to preserve the original graph as much as possible. Finding the minimum number of edges to reverse to make a directed graph acyclic is equivalent to the well known *feedback arc set problem*, which is NP-complete, and several heuristics exist to find a small set of edges to reverse [TDBET98]. However, existing heuristics for cycle removal give little or no control on which edges are reversed and this can lead to undesiderable effects on the layouts produced by Radian, which are based on layering. In this style of drawing edges have monotone directions, e.g. they go from higher layers to lower layers, and a choice of edges to reverse can significantly impact the drawing. For example, Fig. 3.6(a) shows a graph with a cycle. Vertex 1 is the source, vertex 8 is the target, and the edges that close the cycle are dashed. Figure 3.6(b) shows a layered layouts resulting from reversing edge $(5,6)$. The drawing is slightly distorted, and vertex 6 is on a higher level than the source, which does not correctly represent their natural hierarchy. Figure 3.6(c) shows a different layout, where edge $(3,4)$ is reversed. The layout is very distorted, and the source is even on a lower layer than

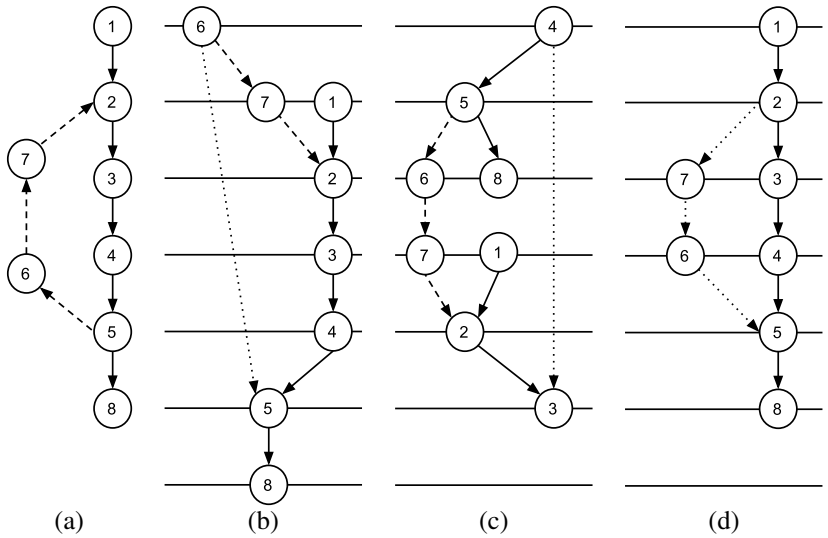


Figure 3.6: Layouts induced by different choices of edges reversed for removing cycles. The edges that close the cycles are dashed, the reversed edges are dotted. (a) A cyclic graph. (b) Layered layout induced by reversing edge $(5,6)$. (c) Layered layout induced by reversing edge $(3,4)$. (d) Layered layout induced by reversing the path from vertex 5 to vertex 2.

the target. Intuition suggests that the correct path in the graph goes from vertex 1 to vertex 8, while the entire path that goes from 5 to 2 is "going back" and is the cause of the cycle. Figure 3.6(d) shows a drawing that follows this intuition, by reversing the entire subpath from 5 to 2.

In the following we describe an algorithm to remove cycles from a directed graph that builds on the fact that the graph is produced from traceroute paths. At a very high level, the algorithm first finds a maximal subset of traceroutes that induce an acyclic graph, and computes a layering for it. Then the remaining paths are added, reversing those that violate the existing hierarchy. Edges are weighted by the number of traceroutes that traverse them, and paths that traverse light edges are preferred for reversing.

Given a graph $G_{\mathcal{T}}$ with clusters $\mathcal{C}$, let T be the set of traceroutes that induce it. Assume that, when referred to $G_{\mathcal{T},\emptyset}$, traceroute paths are contracted, that is, each path is a sequence of clusters. The algorithm starts by assigning a weight to each edge of $G_{\mathcal{T},\emptyset}$ that is equal to the number of traceroutes in T that traverse it. Each occurrence

of a path is counted as one, that is, traceroutes paths repeated over time increase the weight of the edges they traverse. Then the strongly connected components (SCCs) of $G_{\mathcal{T},\emptyset}$ are computed. For each SCC, its edges are iteratively removed from the graph in increasing order of weight, until the vertices of the SCC induce an acyclic subgraph. Each time an edge is removed, the traceroutes that traverse it are put in a set T_1. At the end of the process there are no cycles in the graph. The algorithm merges the traceroutes in set $T_2 = T \setminus T_1$, obtaining an acyclic graph $G'_{\mathcal{T},\emptyset} \subseteq G_{\mathcal{T},\emptyset}$. If it is the empty graph, then the target vertex is added to it. A layering is computed for this graph. All traceroutes in T_1 are added to $G'_{\mathcal{T},\emptyset}$ as follows. Traceroute paths that do not contain cycles are processed first. A path is split in maximal subpaths such that the two extremal vertices of a subpath either have a layer assigned or are a source vertex. Each subpath is added to $G'_{\mathcal{T},\emptyset}$, reversing its direction if it goes towards decreasing layer numbers (i.e. it violates the current layering). If an edge with a given direction already exists then it is skipped, since multi-edges are not allowed. The layering is updated accordingly every time a subpath is added. Then cyclic traceroutes in T_1 are processed. Assume that they follow the same convention as the graph, going from the target vertex τ to a source vertex σ. For each traceroute $\pi \in T_1$:

(*i*) The graph G_π that represents π is computed.

(*ii*) Let $a = \tau$ and $b = \sigma$.

(*iii*) The following steps are executed iteratively, until all edges of π have been added to $G'_{\mathcal{T},\emptyset}$.

(*iv*) An acylic path π_1 from a to b is selected in G_π by a DFS visit. If $a = b$, then the last edge traversed by the visit is returned as a distinct acyclic path.

(*v*) π_1 is added to $G'_{\mathcal{T},\emptyset}$ following the rules for acyclic paths.

(*vi*) The edges of π_1 are removed from π and from G_π. Note that π can contain several copies of an edge, which are all removed.

(*vii*) Find a maximal path in π, let a and b be its first and last vertices.

The procedure described so far computes a graph $G'_{\mathcal{T},\emptyset}$ which is an acyclic version of $G_{\mathcal{T},\emptyset}$. Namely, it contains the same vertices and the same edges, with some edges possibly reversed. The same procedure is applied also to $G_{\mathcal{T},\mathcal{C}}$, with the following constraint: inter-cluster edges must have the same direction established in $G'_{\mathcal{T},\emptyset}$. Namely, let $e = (v_1, v_2)$ be an edge of $G_{\mathcal{T},\mathcal{C}}$ such that $v_1 \in C_1$ and $v_2 \in C_2$. The direction of e is (v_1, v_2) if $(C_1, C_2) \in G'_{\mathcal{T},\emptyset}$, otherwise it is (v_2, v_1). The constraint keeps

$G'_{\mathcal{T},\emptyset}$ and $G'_{\mathcal{T},\mathcal{C}}$ consistent, however it can create source vertices (i.e. vertices with no incoming edges) other than the target. Note that this does not happen for $G'_{\mathcal{T},\emptyset}$. The layout algorithm requires exactly one source vertex, so an incoming edge must be added to each source vertex v_s in $G'_{\mathcal{T},\mathcal{C}}$ that is not the target. Let $v_s \in C_s$, then in cluster C_s there exists a vertex v_c that is not a source (otherwise cluster C would be a source also in $G'_{\mathcal{T},\emptyset}$), and it is used to add a new edge (v_c, v_s) in $G'_{\mathcal{T},\mathcal{C}}$. Finally, $G'_{\mathcal{T},\mathcal{C}}$ is the acyclic graph to be processed by the layout algorithm.

3.8 User Study

We conducted an informal user study at the end of the development of our tool, in order to gather expert opinions on its soundness and effectiveness. We interviewed 6 selected employees of the R&D division of a prominent Italian ISP. Their areas of expertise covered IP edge innovation, cyber security threat evolution, security solutions analysis, and video & multimedia platforms. Such heterogeneous set of expertise covers a wide spectrum of the actual needs and challenges of the ISP. The interviews were held in the context of the Leone FP7 EC research project. We had only one chance to do a user study with them before the end of the research project. The general objective of the study was to receive feedbacks about the motivations of our work, and to assess whether the functionalities offered by Radian supported the typical needs of an ISP. Namely, we wanted expert opinions on how useful the tasks described in Section 3.6 were for solving the use cases described in Section 3.2, and how effective Radian was in supporting those tasks. The users were informed on the supported tasks and their classification into two classes: tasks aimed at understanding the topology of the network as seen by the routing, and tasks aimed at understanding the routing evolution over time. We also wanted opinions on the perceived utility of simplifying the visualization by contracting ASes. A the time the study was conducted, Radian did not support the visualization of metrics.

The interview had three parts. 1) A presentation of the tool, to explain the motivations, the input data, and the functionalities. 2) A supervised session of usage of the tool, in which we proposed a real-life scenario and they were asked to argue on the dynamics of the routing. The focus was not to actually give complete explanations, but to get an initial sense of the logic and interactions of the system. 3) A questionnaire that the users were asked to anonymously fill out with their opinions.

The questionnaire contained several statements, each representing a hypothesis we made that a given feature of Radian was useful or effective. The users were requested to rank each statement between 1 (completely disagree) and 5 (completely agree), and to write a short comment with the motivation for each given rank. Table 3.1 contains

Table 3.1: Results from the questionnaire presented in the user study For each statement, the minimum, the average, and the maximum ranking given by the users are shown.

	Statement	min	avg	max
S1	Traceroute data produced by a probe system, because of the magnitude, are hard to exploit without a visualization tool.	4	4.83	5
S2a	The topology of a part of a network as deduced from traceroute data, both at router and AS level, provides reasonable information to understand the state of the routing in that part of the network and in a given instant.	3	3.33	4
S2b	Radian represents the topology of a network in a comprehensible way.	3	3.83	4
S3a	Traceroutes performed periodically in a part of a network provide sufficient information to understand the dynamics of routing in that part of the network.	2	3.17	5
S3b	Radian represents routing changes in a comprehensible way.	3	4.2	5
S4a	There exist cases in which it is necessary to focus on the routing of a specific AS, keeping at the same time an overview of the routing at AS level.	3	3.6	4
S4b	Radian supports focusing the attention on a specific AS, maintaining at the same time an overview of the neighbour ASes.	3	3.83	5
S5a	Understanding the topology of a network and the dynamics of the routing supports typical activities of network administration and monitoring, including the study of complex scenarios otherwise difficult to analyze.	3	4.5	5
S5b	Analyzing traceroute data with Radian supports the study of complex scenarios.	3	4.33	5

the statements of the questionnaire, together with the minimum, the average, and the maximum ranking given by the users. The statements after the first can be considered in pairs: each statement ending with an "a" is about a motivational matter, and has a corresponding statement ending with a "b" which is about how good Radian was

considered with respect with that motivation.

Statement S1 is about the original motivation of our work, and asked the users whether visualization is actually useful in this domain. It received very high rankings, the highest in the study. The users were already familiar with the analysis of traceroutes, and in the comments they confirmed that understanding large amount of traceroutes, like those produced by a system of probes, is challenging. Visualization was considered a fundamental tool for this kind of task. Some keywords that frequently appeared in the comments as desiderable properties of a visualization were "dynamics", "aggregation", and "overview". Radian has interface elements to support all of these. Namely, the dynamics of routing is represented with animations. Data are aggregated from two different points of view: first, several traceroutes collected over time are merged into a graph, which highly reduces the amount of data to watch. Also, the user can expand and collapse ASes, to further reduce the amount of displayed data and focus only on a portion of interest. A temporal overview of routing changes is furnished by the event timeline. Finally, the graph itself is a form of overview, since all vertices traversed by a traceroute in some time instant are always part of the visualization.

Statements S2a and S2b regard the capability of the traceroute graph to reconstruct a reasonable and comprehensible snapshot of the routing in a given instant. The opinions were mixed. The users were not completely convinced of S2a, because of some known limitations of the traceroute itself. 1) Some routers of the Internet could be configured, for security reasons, not to respond to measurements like traceroutes. 2) Even if configured to respond, an overloaded node could decide to discard some requests. 3) A reported traceroute path could be a simplified version of the real path, because packets transit inside tunnels that are invisible to traceroutes. 4) Finally, a traceroute path could report non-existing links between some nodes, because of the presence of load balancers in the traversed network. This kind of problem is tackled by a special version of the traceroute tool (see [ACO+06a]), which however could be not installed on the probes used for the measurements. For these reasons, many of the users suggested to integrate traceroute data with the decisions taken by routing protocols, which are known to an ISP for its own network. On the other hand, all users admitted that these limitations of traceroutes are hardly avoidable, and that when measuring a network belonging to someone else, the traceroute (with its limitations) is one of the very few, if not the only, tools available. The average ranking, even if not that bad (it is greater than 3), is relatively low. We believe that this was due to the strong statement we made: it would have been probably higher if we stressed more the fact that the graph reconstructed from traceroutes is only an approximation of the topology of the traversed network and of its routing. Surprisingly, Statement S2b received higher rankings: this means that, apart from the reported limitations of the traceroute

tool, the graph metaphor implemented in Radian was considered clear and effective. This was also confirmed in the comments.

Statements S3a and S3b regard the effectiveness of periodically performed traceroutes to sample the dynamics of the routing, and the effectiveness of the animations implemented in Radian to represent such dynamics. Statement S3a received a particularly low average ranking and was subject to criticisms similar to S2a: traceroutes were considered too poor in information to let the user to fully understand the routing dynamics. The statement, in our intentions, expressed the capability of periodic traceroutes to report a sampling of the routing changes happened in the network, so that the user becomes aware of them and has the possibility to make deductions from their comparison. However, similarly to S2a, we believe that the statement was too strong and that was interpreted by the users. Their written comments were fundamental to understand the reasons behind the answers: they complained that simply seeing the routing changes is not enough to understand the reasons behind them, which requires the usage of additional data sources like information on the routing protocols. Two of them were very specific in these terms, saying that traceroutes do not allow a "root-cause analysis" of the routing events, and that they do not help "understand why". We are obviously aware of these limitations, and never intended to present Radian as a tool for root-cause analysis, which is a challenging task that requires specific tools. All that Radian is capable of is to report the sequence of routing changes happened over time, so that the user can precisely tell what changed and when, and then draw some conclusions. The reported events are possibly starting points for a deeper analysis. And, indeed, the very high ranking of Statement S3b confirmed that Radian is effective in this task. The written comments pointed out that the users appreciated the usage of animations, and considered them an effective an intuitive way to represent changes in a traceroute path. Some very interesting comments to Statement S3a and S3b regarded the possibility of comparing routing changes to other metrics, like the round-trip time. The user considered useful, to understand the reason of a routing change, to know if some metric of interest changed at the same time. For example, a sudden improvement of the round-trip time in correspondence of a routing change may imply that some node was overloaded, and the routing protocol changed the routing to avoid that node and restore acceptable performance. A feature of this kind was indeed missing in Radian, and its conceiving is an important outcome of this user study. Hence, we labelled traceroute paths with round-trip time information. However, integrating networked data with more network metrics into one visualization is an interesting challenge, also, and we will work on it in the future.

Statements S4a and S4b regard the usefulness of looking at the network at different abstraction levels, exploiting the clustering of nodes into the ASes they belong to, and the effectiveness of Radian in supporting this feature. Despite the good rankings,

these were the only Statements for which no significant written comments were given by the users. We believe that, more than for the previous statements, S4a was too abstract and was not understood completely. Therefore, the users may have given rankings similar to those of the previous statements. However, we observed them during the usage session of Radian, to spot interesting patterns of use, and noticed that all of them made use of the possibility of keeping collapsed the ASes that were not involved in any dynamics. This makes perfectly sense to us, since some ASes with static routing were very large and caused cluttering on the screen, while the interesting part of that instance was the very particular inter-AS routing dynamics. This observation indirectly confirms our expectation that, if available, users gladly use a feature to simplify the current visualization by abstracting those parts that are not of interest for a given task.

Finally, Statements S5a regards the general functionalities offered by Radian, consisting in supporting the comprehension of the topology of a network and the dynamics of its routing, as inferred from periodic traceroute data. Statement S5b asks how good Radian is at implementing these features. The average ranking is quite high for both, which is a strong confirmation of the quality of our work. In the written comments, the users considered Radian very effective for the tasks it was designed for, and said that it could help "debugging problems" and "refine future strategies", referring to the administration of a network. Differently from before, the Statement regarding the implementation received slightly lower rankings than the motivations behind it. In the comments the users explained that the possibility of comparing routing changes to other kind of metrics is an important, missing feature of Radian, which would enable a much deeper analysis of the routing. This influenced their general opinion on the tool. See the comments to Statements S3a and S3b.

In conclusion, the users considered Radian a very useful tool for supporting their everyday work in the administration of a network. The kind of functionalities and our implementation were well appreciated. The main complaints were for the intrinsic limits of traceroutes. On the other hand, the users admitted that these limitations are unavoidable and that traceroutes are one of the few sources of data available when analysing a network administrated by someone else, on which there is no control nor information available on the routing protocols. The possibility of visually comparing routing changes to other kind of metrics is considered a fundamental, missing feature of Radian, which motivates the changes discussed above.

3.9 Conclusions and Future Work

We presented a metaphor for the visualization of traceroute measurements towards specific targets on the Internet. It consists of a radial drawing of a clustered graph where vertices are routers or computers and clusters are administrative authorities that control them. Our metaphor allows the user to interact with the visualization, both exploring the content of individual clusters and animating the graph to see how traceroute paths change over a time interval of interest. The visual metaphor and the relative algorithms were implemented as a tool, Radian.

In the future we will take into account the *DNS resolution* of selected targets in the visualization. That means that some targets may be represented by more than one vertex, giving rise to an anycast behavior of the target, depending on the policies implemented at the DNS level. We will also explore the possibility to process streams of incoming data, adding or removing elements in the visualization incrementally. Finally, we will improve the visualization of network metrics presented by Radian. The current solution supports the user tasks that we considered, but it could be improved in two ways. First, metrics should be integrated with the graph visualization, besides be presented with temporal charts. Second, the system should allow an easy comparison between metrics of different probes, and of single probes with respect to the global trend.

Chapter 4

Visualization of Network Metrics as Stacked Charts

This chapter describes an approach based on *Multiple Abstraction Levels* (see Chapter 1) for visualizing network metrics, like the round-trip delay reported by the traceroute tool. The approach deals with the *Relation To Metrics* challenge. Metrics are time series, that is, sequences of numeric values that depend on time. Visualizing this kind of data is a problem of general interest, so the approach in this chapter is presented in the more general context of time series visualization. *Streamgraph* is a technique for representing time series as stacked charts. It allows one to visually compare the trend of single time series to the global trend, that is, the trend of the sum, which results in a comparison of two different abstraction levels of this kind of data. The state of the art on this technique, when applied to network metrics, can produce drawings with noticeable distortions that reduce the readability. The solution described in this chapter deals with this limit, and also introduces an efficient algorithm for automatically labeling time series in a streamgraph. Experiments were conducted on a number of real data sets. A preliminary version of this chapter was published in [DH].

4.1 Introduction

A time series is a sequence of numeric values each labeled with a temporal reference, and ordered by time. A standard way to visualize time series is to plot them on a cartesian chart that has time on the x-axis and the numeric values on the y-axis. This chart clearly shows how the series evolve over time. Plotting multiple time series in the same chart allow easy comparison among them. However it is not effective at

showing the evolution of their sum.

Stacked graphs, or stacked charts, are a representation that mitigate this limit. In a stacked graph, time series are shown as colored stripes (or *layers*) that flow in the direction of the x-axis, whose thickness represents at each time instant the numeric value. Layers are stacked one on top of another without gaps. The results is a diagram in which it is easy to compare single layers with the total, at the expense of harder comparisons between pairs of layers.

Given a stacked graph, the *baseline* is its bottommost curve. The simplest way to make a stacked graph is to stack layers on a straight line that corresponds to the x-axis. However, different baselines are possible, and may be preferred because it offers a way to reduce the amount of fluctuation in the layers. First introduced by ThemeRiver [HHWN02], stacked graphs with curved baselines were popularized in 2008 by an article on The New York Times, which used them to visualize box office revenues for 7500 movies over 21 years. The visualization become immediately popular and controversial, gathering comments ranging from "fantastic" to "unsavory". Later on, a paper by Byron and Wattenberg [BW08] described the technique used in The New York Times visualization, calling it *streamgraphs*. The article outlined how the aesthetic of a streamgraph is controlled by three components: the ordering of layers, the shape of the baseline, and the labeling of the layers. The solutions used by the Times were described. In addition, the authors explained how stacked graphs with a flat baseline, ThemeRiver-like graphs and their streamgraphs all fitted in a general mathematical framework, and that each of these had different aesthetic properties. For this reason, in this chapter we will refer to any kind of stacked graph as a streamgraph.

At the time of writing this thesis, the paper of Byron and Wattenberg [BW08] is still the most authoritative work on streamgraphs, and other works are mostly implementations or minor variations of the original concepts. The algorithmic pipeline of Byron and Wattenberg [BW08], and that of this thesis, consists of three distinct steps: layers are first ordered; a baseline is then computed that minimizes fluctuation of layers; finally, labels are computed and superimposed to the streamgraph. However, in each of these steps, the solutions in [BW08] have limitations that have not been documented in the literature. First, the layer ordering algorithm exploited statistical properties of the data of interest, box office revenues, that may not hold in other data. Second, the algorithm to compute a baseline is based on a 2-norm measurement of the fluctuation of layers, called *wiggle*, which works well for relatively smooth time series, but suffers from unpleasant distortions when a layer has sudden changes in thickness (or *jumps*, see Fig. 4.1). Third, in [BW08] a brute-force layer labeling algorithm was used that does not scale well with the size of the data.

We resolve each of these limitations by proposing better ordering, baseline calculation and layer labeling algorithms, and verified the effectiveness of our solutions by

performing experiments that compared our algorithm with the current state of the art. Our contributions are

- a new ordering algorithm that works effectively on general time series data.
- an alternative definition of wiggle based on 1-norm that gives visually calmer layer arrangements even for time series with sudden jumps, and a solution process for the 1-norm minimization problem.
- an efficient layer labeling algorithm that scales linearly to the size of the time series data.

Together these represent a new algorithmic pipeline that significantly advance the state of the art in the creation of streamgraphs.

4.2 Related Work

The first work to introduce streamgraphs was ThemeRiver [HHWN02], a tool to visualize the time evolution of topics extracted from large collections of text documents. The thickness of a layer represented the popularity of a topic between the selected documents. The baseline was computed so that the drawing was symmetric with respect to the x-axis. No specific layer ordering algorithm were described, however the authors discussed about the possibility of letting the user decide the ordering or putting related layers close.

The paper from Byron and Wattenberg [BW08] formally introduced streamgraphs. It described the mathematical technique and design considerations behind the stream graphs in a 2008 New York Times article showing box office revenues for 7500 movies over 21 years. The authors introduced the concept of *wiggle* as a measure of distortion of all the layers. Minimizing this measure (see also Section 4.3 for a description) aimed at both making layers more readable, and at avoiding the artificial look of ThemeRiver drawings which are symmetric with respect to the x-axis, in favor of a more natural, river-alike flowing. For ordering of layers, the authors outlined how layers with high wiggle values should not be at the center of the drawing to avoid distorting the other layers around them. Layers produced by box office revenues tend to have a sudden increase in the first weeks out, then they rapidly decrease as the interest in the movies declines. For this reason, layers were ordered by their "on-set" time, with older movies at the center of the drawing and newer movies at the edges. It was left as future work to study how layers can be ordered by using the wiggle as a measure of quality. Other aspects were also discussed, like layer coloring and labeling.

Several other works used streamgraphs, but most of them are applications of the concepts of [HHWN02] and [BW08], which were extended to work on specific applicative contexts.

TIARA [LZP+12] is a system that visualizes topics extracted from text documents. Layers represent topics and are enhanced by adding keyword clouds inside them. The ordering of layers is chosen on the basis of several, contrasting, criteria, which include the on-set approach of [BW08] and a new measure of the volatility of a layer, to estimate its wiggle. Although the paper does not completely describe how to combine these criteria, their solution is one of the few attempts to order layers with an approach different from [BW08].

TextFlow [CLT+11] is a system for visualizing the evolution patterns of topics extracted from text documents. The visualization has some similarity with streamgraphs, since topics are shown as stripes (or "rivers") flowing in one direction, and whose thickness represents a measure of magnitude. Moreover, TextFlow computes its layout starting from a stacked graph. However, unlike streamgraphs, topics can merge and split over time, and rivers in the visualization split and merge into branches accordingly.

TripVista [GWY+11] is a system that visualizes traffic trajectory data. Layers represent traffic volume information, while the layout is first computed with the ThemeRiver algorithm and then enhanced by adding to the layers glyphs for representing trajectory directions. In this way, directions and volume statistics on traffic are combined.

Another work dealing with the visualization of topics is the system described in [DGWC10]. The system supports the visualization of dynamic data, which the user can navigate by panning the current time interval. Layers are totally ordered by a global measure of "newness", that it the time of first appearance of a topic. A topic never changes its position in the ordering. The on-set approach of [BW08] resulted ineffective in a dynamic setting.

TouchWave [BLC12] is a system that exploits multi-touch interfaces for interacting with streamgraphs. With standard interactions, layers can be scaled and extracted from a stack to improve their legibility. If the user touches a point on the x-axis, layers are dynamically reordered in increasing ordering of the values they have at that x coordinate. Additionally, the user can manually rearrange the ordering. Hierarchies of layers are supported, by expanding and collapsing layers.

Other areas of visualization studied problems similar to that of streamgraphs, namely, optimal ordering and sloping of lines to minimize the wiggle. In Storyline visualization [TM12,LWW+13], this is achieved by genetic algorithms [TM12], or by quadratic programming [LWW+13]. In directed graph visualization [GKNpV93],

polyline edges across multiple layers are kept as parallel to the vertical direction as possible by linear programming.

4.3 Finding a Baseline via Wiggle Optimization

In this section we define the concept of wiggle for a streamgraph, and show how to compute a baseline for a streamgraph such that the wiggle is minimized. After discussing the limits of the existing techniques, we describe our solution, which encompasses a new definition of wiggle and an optimization method.

Basic Concepts

We assume to work with discrete data, that is, a time series is a sequence of m numbers. Given an ordered list of time series, we assume that the layer ordering of a streamgraph of them follow the ordering of the list. We denote f_i as the i-th time series, where $i = 1,2,\ldots,n$. Given a streamgraphs of series f_i, we denote g_i as the sequence of y-coordinates corresponding to the points of series f_i. As a particular case, g_0 denotes the baseline of the streamgraph. Also, note that $g_i = g_0 + \sum_{j=1}^{i} f_j$ for $i = 1,2,\ldots,n$.

Wiggle is a metric introduced in [BW08] to measure the aesthetics of a streamgraph. Intuitively, the wiggle of a streamgraph can be thought of as a indication of how much the top and bottom boundaries of its layers fluctuate. For example, a streamgraph with only flat layers has wiggle equal to 0. Wiggle is a measure of the visual complexity of a streamgraph, since distorted layers are harder to understand, e.g., an increasing trend in the thickness of a layer could be hidden by a visual distortion that visualizes the layer as a steep descent. In [BW08] two different definitions of wiggle are given.
The "weighted wiggle", which gives more importance to layers with higher thickness and is thus consider better, is defined as

$$\mathrm{ww}_2(g_0) = \sum_{i=1}^{n} f_i \left(\frac{g_i^{'} + g_{i-1}^{'}}{2} \right)^2 \tag{4.1}$$

By definition, wiggle is a function which returns a number (*wiggle value*) for each x-coordinate of a streamgraph. However, with a slight abuse of notation, we also define the *wiggle value of a streamgraph* as the sum of the wiggle values of the streamgraph over all x-coordinates. Further, the wiggle value of a layer is the wiggle value of a streamgraph composed only of that layer and having a flat baseline. Finally,

the wiggle value of a list of ordered layers is the wiggle value of the streamgraph that has that layer ordering and a baseline that minimizes the wiggle.

With a fixed layer ordering, the wiggle of a streamgraph depends on the baseline. A method to find a baseline g_0 that minimizes the wiggle is described in [BW08], and consists of taking the derivatives of (4.1) with regard to g_0', and set them to zero. The optimization problem can be solved with a per-value approach, restricting (4.1) to a single x-coordinate of the streamgraph. Derivatives in the equations can be computed with the backward finite differences.

Limits of Existing Techniques

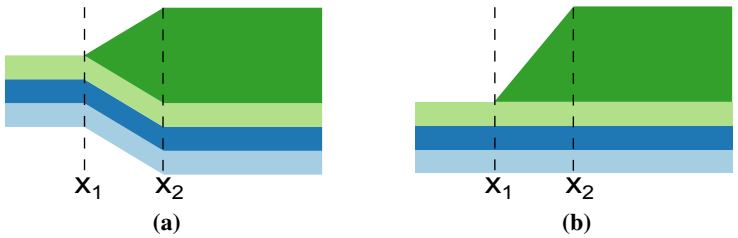
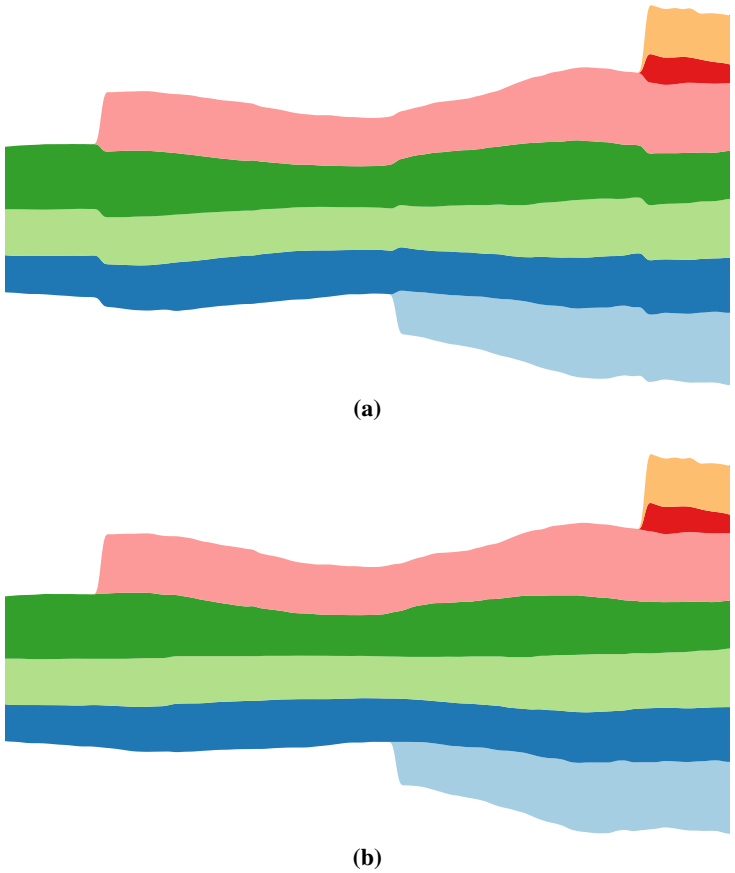


Figure 4.1: Different baselines: (a) with weighted 2-norm minimization, notice the wiggle on all layers; (b) with 1-norm minimization: only the green layer has a wiggle.

Equation 4.1 was used to produce aesthetically pleasant streamgraphs of box office revenues in The New York Times article. It is effective in visualizing relatively smooth time series, but suffers from unpleasant distortions when a layer has sudden changes in thickness, or *jumps*. In Fig. 4.1 two possible drawings of four ordered layers are depicted, where one layer is shorter than the others. It is easy to see that the streamgraph in Fig.4.1a is affected by distortions at x_1 and x_2, that is, in correspondence of the first non-zero value of the short layer at x_2. Much of these are avoidable, as shown in Fig. 4.1b. Equation 4.1 gives a lower wiggle value for Fig. 4.1a than Fig. 4.1b. For example, assume that the short layer has thickness equal to 4 and the others have thickness equal to 1. Also, assume $x_2 - x_1 = 1$. In Fig. 4.1a, from bottom to top, values g_i', with $i = 0 \ldots 4$, at x_2 are -2, -2, -2, -2, and 2, respectively, therefore $\text{ww}_2 = 12$. In Fig. 4.1b, with a similar reasoning, we obtain $\text{ww}_2 = 16$. This means that, when minimizing (4.1), a baseline like Fig. 4.1a is preferred over that of Fig. 4.1b. In real data, the presence of many layers can amortize the distortion due to a jumping layer, however it is still noticeable and unpleasant, see Fig.4.2a.



(a)

(b)

Figure 4.2: Different baselines: (a) with 2-norm, distortions are present; (b) with 1-norm, all layers are smooth.

There are two reasons for the unappealing distortions made by a baseline computed by minimizing (4.1). First, consecutive wiggle pairs $(g_i^{'}, g_{i-1}^{'})$ are cancelled if the two terms have equal absolute value and opposite sign. Such pairs are ignored in the minimization process, independently of the amount of distortion they cause to other layers. In Fig. 4.1a, this happens with $g_3^{'}$ (light green) and $g_4^{'}$ (dark green). Also, the wiggle is defined based on 2-norm, which tends to favour many small "wiggles" to a large one, as seen in Fig. 4.1a and Fig. 4.2a.

Optimal Weighted Wiggle Under 1-norm

Mathematically, the quadratic function involved in the 2-norm based definition of wiggle is smooth and easier to optimize, which may be part of the reason why Byron and Wattenberg choose to define wiggle that way. However in the follow we show that 1-norm based definition, though non-smooth, can also be solved easily, but does not suffer from the issue with distortion due to sudden jumps.

We define the 1-norm based weighted wiggle as follows:

$$\begin{aligned} \mathrm{ww}_1^a(g_0) &= \sum_{i=1}^{n} f_i \left(\frac{\left|g_i^{'}\right| + \left|g_{i-1}^{'}\right|}{2} \right) = \sum_{i=0}^{n} w_i \left|g_i^{'}\right| \\ &= \sum_{i=0}^{n} w_i \left| g_0^{'} + \sum_{j=1}^{i} f_i^{'} \right| = \sum_{i=0}^{n} w_i \left| g_0^{'} - p_i \right|. \end{aligned} \tag{4.2}$$

Similarly to (4.1), we assume to work on a per-value basis, that is, at a fixed x-coordinate of the streamgraph. We denote $w_i = \frac{1}{2}(f_i + f_{i+1})$ (we assume $f_0 = f_{n+1} = 0$), and $p_i = -\sum_{j=1}^{i} f_i^{'}$ (we assume $p_0 = 0$). The wiggle is based on 1-norm, which avoids uniform small jumps across the layers, a problem discussed in Section 4.3 that affects the 2-norm wiggle, in favour of a few larger jumps. This sparsifying effect is similar to the use of 1-norm for regularization in regression, known as *lasso* [Tib94]. Also, the norm is computed for each $g_i^{'}$ term, which avoids the cancelling of terms with equal absolute value and opposite sign.

To minimize (4.2), we first observe some properties of its derivative with respect to $g_0^{'}$, which is $d(g_0^{'}) = \sum_{i=0}^{n} w_i \ \mathrm{sgn}\left(g_0^{'} - p_i\right)$. With a slight abuse of notation, we reorder the $p's$ from small to large and still denote them as p, in such a way that $p_0 \leq p_1 \ldots \leq p_n$. When looking at the $n+2$ intervals defined by this sequence from left to right, the derivative of the wiggle, when $g_0^{'} \in (-\infty, p_0)$, is $d_{-1} = -\sum_{i=0}^{n} w_i$. The derivative when $g_0^{'} \in (p_0, p_1)$ is $d_0 = d_{-1} + 2w_0$, the derivative when $g_0^{'} \in (p_1, p_2)$ is $d_1 = d_0 + 2w_1$, etc. That is, the derivative of the wiggle is negative for $g_0^{'} < p_0$, it increases as $g_0^{'}$ increases, and it is positive for $g_0^{'} > p_n$. Therefore, the minimum wiggle is achieved at the point where the derivative changes from negative to non-negative. Fig. 4.3 shows an algorithm for finding the optimal $g_0^{'}$. The returned value can be numerically integrated to obtain g_0, that is, a baseline value that minimizes (4.2).

Fig. 4.2 compares the effect of computing a baseline with 2-norm minimization, and with this technique. The baseline in Fig. 4.2a is computed by minimizing (4.1),

```
1: function OPTIMALBASELINEDERIVATIVE(w, p)
2:     d ← −∑_{i=0}^{n} w_i
3:     for i = 0, 1, ..., n do
4:         d ← d + 2w_i
5:         If d ≥ 0 return p_i
6:     end for
7: end function
```

Figure 4.3: The optimal baseline algorithm

and it is affected by distortion near the boundaries of short layers. The baseline in Fig. 4.2b is computed by minimizing (4.2), resulting in a smoother drawing.

4.4 Layer Ordering

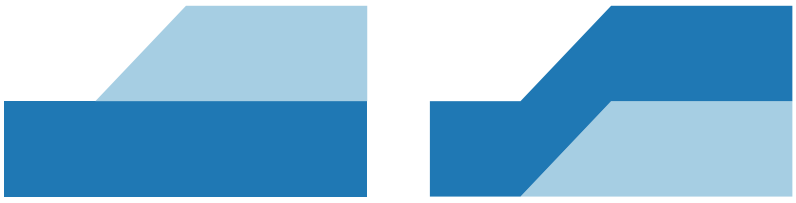
In [BW08] an algorithm is proposed for ordering layers. Layers produced by box office revenues tend to have a sudden increase in the first weeks out, then they rapidly decrease as the interest in the movies declines. For this reason, layers were sorted by their "on-set" time, defined as the x-coordinate of the first non-zero value. Ordering was done with an "inside-out" approach, which stacks the sorted layers by alternatively putting them above or below the two edges of the drawing. As a result older movies appear at the center of the drawing and newer movies at the outskirt, and the overall effect is visually appealing. Different implementations are possible, for example the D3 library [Bos] implements streamgraphs ordering layers by the x-coordinate of their maximum value. These approaches work well for movie data but are not appropriate for general data. For example, Fig. 4.5a is an ordering produced with the on-set method of [BW08]. Although layer 10 has an early on-set time, it cannot be placed at the center of the drawing without distorting other layers. As an alternative, more general approach, [BW08] suggested that layers with high wiggle values should not be at the center of the drawing to avoid distorting the other layers around them, but did not implement this idea.

This section describes an algorithm for ordering the layers of a streamgraph that is effective for general time series. This involved generating an initial good ordering, and subsequent iterative refinement of the ordering.

The initial ordering is based on the intuition, introduced in [BW08], that layers with a large amount of wiggle must be as far from the center of the drawing as possible, so not to perturb the other layers around them. On the other hand, putting calm layers at the center gives a good support for stacking other layers on top of them. Our

`BestFirst` initial ordering algorithm exploits this idea. The algorithm starts from an empty ordering and iteratively adds layers, choosing the one with the best wiggle at each iteration. More specifically, the algorithm starts with a straight line, whose two sides are labeled as *current top baseline* and *current bottom baseline* and are defined as two sequences of 0's. Then, all layers still to insert are tested. Each of them is stacked on both the current top and bottom baseline, computing the wiggle produced by each choice. Finally, the layer-baseline pair with the lowest wiggle is selected for insertion. The chosen current baseline is then updated by adding the thickness of the inserted layer. An iteration is performed for each layer to insert.

`BestFirst` is effective at keeping "calm" layers inside the drawing and putting the others outside. However, on some instances it can produce aesthetically unpleasant orderings, such as in Fig. 4.5b, where it is easy to see that layer 24 and layer 6 should be switched (see Fig. 4.4), and that layer 30 should be on top of them. The reason for the wrong decisions is that `BestFirst` computes the wiggle of a layer as an integral along the layer. Parts of a layer that have thickness equal to zero do not contribute to the wiggle, and this could mitigate the sudden increment in value (despite the high derivative) that is present at the boundaries of the zero intervals. As a result, a short layer tends to have much smaller wiggle that a long layer, and is more likely to appear closer to the center of the streamgraph, distorting any other layers on top of it. This is a critical limit of `BestFirst`. Given that data with jumps is common in the real world (e.g., demographic data could have been collected starting from different dates for different countries), we propose a refinement algorithm, `TwoOpt`, to further improve the initial ordering.



(a) **(b)**

Figure 4.4: Two orderings of two layers on a flat baseline. Case (b) produces a distortion, case (a) is preferred.

In `TwoOpt`, the wiggle of a layer is evaluated also with respect to the impact it has on the layer above. For example, Fig. 4.4a is a better ordering than Fig. 4.4b, since in the latter the bottom layer does not properly "support" the top one, and distorts

it. Starting from an initial ordering, the algorithm iteratively compares two neighboring layers and decides which of the two possible orderings gives the lowest wiggle. Specifically, the two orderings are stacked on a flat baseline, then their respective wiggles are computed and compared. Note that the reason we compare the orderings of two layers on a flat baseline, instead of evaluate using the total wiggle on the best baseline each time, is that the former is computationally cheaper, and that a good ordering tends to give relatively flat middle layers, hence using a flat baseline is reasonable.

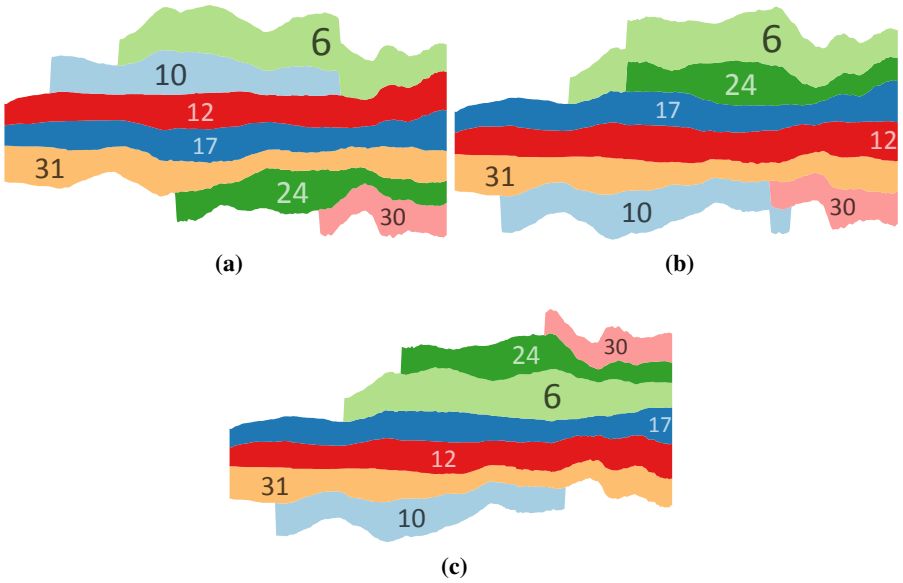


Figure 4.5: Layer orderings produced by different algorithms: (a) on-set; (b) BestFirst; (c) TwoOpt. The first two have layer distortions. Labels identify layers and were positioned with the algorithm described in Section 4.5.

Fig. 4.6 gives the pseudo code for `TwoOpt`. It starts with the initial ordering from `BestFirst`. Then, from the center of the ordering, it performs several inside-out scans towards the top and the bottom. Each scan uses the wiggle heuristic to compare every pair of adjacent layers, and decides whether swapping them decreases their wiggle. The scans are repeated the given number of times and each repetition produces a new ordering, which is evaluated through the wiggle heuristic. To do this, the ordered layers are stacked with a baseline such that the center of the drawing, i.e. the start

```
1: function TwoOpt(init, m, nr, ns, wig)
2:     ▷ init: initial ordering; m: the center of the drawing; nr: the number of repeats; ns: the
   number of scans; wig: the wiggle heuristic.
3:     ordering ← init, bestWiggle ← ∞, bestOrdering ← nil
4:     ▷ Do random executions and take the best result
5:     for r ← 1..nr do
6:         ▷ Do not discard the initial ordering
7:         If r > 1 shuffle the ordering
8:         ▷ Execute several inside-out scans
9:         for s ← 1..ns do
10:            for j ← m..lenght(ordering)−1 do
11:                ▷ Compare the wiggle of j, j+1 and j+1, j
12:                res ← cmp(j, j+1, ordering, wig)
13:                if res > 0 then
14:                    swap(j, j+1, ordering)
15:                end if
16:            end for
17:            Scan in the other direction, with j ← m−1..2
18:        end for
19:        ▷ Evaluate the produced ordering
20:        graph ← midlineGraph(ordering, m−1, m)
21:        currWiggle ← wig(graph)
22:        if currWiggle < bestWiggle then
23:            bestWiggle ← currWiggle
24:            bestOrdering ← ordering
25:        end if
26:    end for
27:    return bestOrdering
28: end function
```

Figure 4.6: The algorithm to order layers

point of the inside-out scan, is a straight line. Then the wiggle of this streamgraph is computed. After every repetition, the current ordering is shuffled for escaping local minima. Finally, the ordering with the lowest wiggle is returned as result. Shuffling avoids situations like Fig. 4.5b, in which layer 10 and layer 30 are at the same side of the central reference line, and swapping them does not improve their wiggle. Fig. 4.5c is a drawing with an ordering produced by `TwoOpt`, which does not have the distortions of the on-set and `BestFirst` versions (see respectively Figures 4.5a and 4.5a).

4.5 Labeling of Layers

An important part of the design of a streamgraph is the placement of labels for the layers. Ideally a label is visually associated with the data it represents.

Byron and Wattenberg [BW08] placed labels within the layers themselves, rather than using a call-out line. The font size of the labels is adjusted to fit each layer. The labels are located to maximize the font size. They adopted a brute-force algorithm, but noted that "the online interactive piece does not use this proposed label placement strategy because of the poor real-time performance of the brute-force algorithm." Based on their description, we suspect that the computational complexity of their algorithm may be quadratic to the data size. In this section we propose a faster label placement algorithm that scales linear to the data size, and logarithmic to the ratio between the largest and smallest desirable font size.

We assume that the top and bottom layers for which we need to place the label is defined by the sequence $\{t_i | i = 0, \ldots, m-1\}$ and $\{b_i | i = 0, \ldots, m-1\}$, respectively, with $t_i > b_i$.

Suppose we have a label with width w and aspect ratio σ (defined as the ratio between width and height of the label) to be placed in this layer such that it is centered at $x = i$. Then the lower boundary of this label much be greater than b_j for all $j = i - w/2, \ldots, i + w/2$. Thus the lowest the label could reach along the y-direction is the top most point of the bottom sequence within the x-window size of w centered at i, namely $B_i = \max_{i-w/2 \le j \le i+w/2} b_j$. Similarly the highest it can reach along the y-direction is the bottom most point of the top sequence within the x-window of size w centered at i, or $T_i = \min_{i-w/2 \le j \le i+w/2} t_j$. Fig. 4.7 shows a layer with $m = 10$ points and $w = 4$. On the left side, the shaded area shows the case with $i = 2$. Here we have $B_2 = 1$ and $T_2 = 2$. On the right side, when $i = 7$, we have $B_7 = 1$ and $T_7 = 3$.

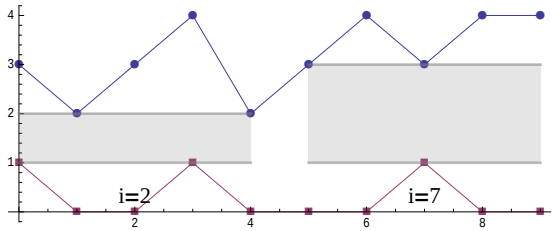


Figure 4.7: Lower and upper bounds for a label with width $w = 4$. Left:streamgraphs- $i = 2$. Lower/upper bounds $B_2 = 1$ and $T_2 = 2$. Right: $i = 7$. Lower/upper bounds $B_7 = 1$ and $T_7 = 3$

Therefore to place a label with the largest possible font size, we can start from a

large label width w based on the largest font size to use and the number of characters, then vary i from $w/2$ to $n-1-w/2$, and check whether the achievable height of the label, $H_i = T_i - B_i$, satisfied the aspect ratio requirement $w/H_i \leq \sigma$. If it is true, we can define the center of the label as $(i,\ 0.5*(B_i+T_i))$. If not, we shrink w by a constant factor (e.g., 10%), and repeat the process until a feasible solution is found. We know this process will finish in the number of steps logarithmic to the the ratio between the largest font size, and largest feasible font size, because after that many step, w will be smaller enough to allow a feasible solution.

To compute B_i (or T_i) for $w/2 \leq i \leq m-1-w/2$, a naive way would be to compute the maximum (or minimum) of a sliding x-window of width w over the sequences b (or t). However this would take time $O(wm)$. Instead we can use a faster sliding window min/max algorithm that compute all mins or maxes in time $O(m)$. This fast labeling algorithm is given in Fig. 4.8. In the algorithm, the *slidingWindowMin* and *slidingWindowMax* functions implement the faster sliding window min/max algorithm (see, e.g., http://articles.leetcode.com/2011/01/sliding-window-maximum.html).

```
1: function LABELING(t, b, minFontWidth, maxFontWidth,
   label, sigma)
2:     w ← round(maxFontWidth * length(label))
3:     w_min ← round(minFontWidth * length(label))
4:     while w > w_min do
5:         T = slidingWindowMin(t, w)
6:         B = slidingWindowMax(b, w)
7:         imax ← argmax_i{B_i − T_i | i = 0, 1, ..., m − w − 1}
8:         hmax ← B_imax − T_imax
9:         center = (½w + imax, ½(T_imax + B_imax))
10:        if w ≤ σ * hmax then
11:            render label at center with width w and height w/σ
12:            return
13:        end if
14:        w ← w − max(1, round(0.1 * w))
15:    end while
16: end function
```

Figure 4.8: The algorithm for layer labeling

4.6 Time Complexity of the Algorithms

Like the 2-norm baseline algorithm [BW08], the proposed weighted 1-norm baseline algorithm works on a per-x-value basis. It takes $O(n)$ operations to find the baseline

derivative using Algorithm 4.3 per x-value. Thus overall complexity is $O(nm)$, i.e., linear to the data size.

We analyse the complexity of the proposed algorithm in the following. Recall that we assume that each time series is a sequence of m numbers, and there are n time series.

Our ordering algorithms execute as subprocedures the operations of constructing a streamgraph by stacking a set of layers on a baseline, and computing its wiggle value. Both these operations can be executed in $O(nm)$ time.

`BestFirst` executes one iteration for each time series to add to the streamgraph, so there are $O(n)$ iterations. In turn, each iteration tests every time series that has not be added to the streamgraph, requiring another $O(n)$ factor. A time series is tested by stacking it on each of the two current baselines, and evaluating its wiggle. These two operations, executed on a single time series, takes $O(m)$ time. Therefore, the total time complexity of `BestFirst` is $O(n^2m)$.

`TwoOpt` depends on two parameters: the number of repeats r and the number of scans s. Refer to Fig. 4.6. First, the initial layer ordering is shuffled, which can be done in $O(n)$ time with the Fisher-Yates method. Then s inside-out scans are executed. For each of them, time series are pairwise compared and $O(n)$ comparisons are performed. Comparing two time series means stacking them on a flat baseline in one of the two possible orderings. Constructing these two streamgraphs and computing their wiggles takes $O(nm)$ time. After the scans, a baseline for the ordered series is computed, such that the center of the resulting streamgraph, i.e. the start point of the inside-out scan, is a straight line. This requires to sum at each time point all series that appear in the ordering before than the position representing the center of the drawing, which is given in input. So, computing the baseline takes $O(nm)$ time. Finally, stacking the ordered layers on this baseline and computing its wiggle takes $O(nm)$ time. Since the entire procedure is repeated r times, the time complexity of `TwoOpt` is $O(r(n+snm+nm)) = O(rsnm)$.

The layer labeling algorithm finds the largest possible height for a fixed width of the label in time linear to m per layer. The overall complexity is $O(nmlog(r))$, where r is the ratio between the largest font size to try out and the largest feasible font size.

4.7 Experiments

We performed numerical evaluation to compare the effectiveness and the efficiency of our layer ordering algorithm to the state of the art. The quality of an ordering is measured using the 1-norm (4.2) and the 2-norm (4.1) wiggle values. The hypothesis

tested in these experiments is that ordering algorithms designed specifically to reduce the wiggle produce drawings with a lower wiggle than other algorithms.

The experiments were conducted on a number of datasets, which we collected from real applications. These are:

- Unemployment: unemployment statistics [datb] for 31 European countries, from 1983 to 2013. 371 time points.
- Movies: box office revenues in the U.S. [data] of 161 movies which were on screen in the first six months of 2015. 28 time points.
- Sandy: number of calls to the NYC 311 [date] in the days before and after hurricane Sandy hit New York City, i.e. from 10/14/12 to 11/17/12, for 158 topics. 35 time points.
- Stocks: stock prices of 662 companies listed on the NASDAQ [datf] from 2005 to 2015. 121 time points.
- Marketcap: market capitalization of 662 companies listed on the NASDAQ [datg] from 1995 to 2015. 241 time points.
- Google: relative variations in volume of Google search traffic in the U.S. [datc] across 27 sectors of the economy from 2014 to 2015. 365 time points.
- Linux: volume of commits on GitHub for the Linux kernel project [datd] for 786 contributors from 2013 to 2015. 24 time points.

The algorithms evaluated in the experiments were: Optimum, TwoOpt, TwoOptR, BestFirst, Onset, D3 and Random. TwoOpt and TwoOptR both implements the algorithm in Fig. 4.6, with the difference that TwoOpt applies BestFirst to produce an initial layer permutation, while TwoOptR starts from a random permutation. Optimum is a brute-force ordering algorithm that explores all layer permutations, and selects the one that minimizes the wiggle. Finally, algorithm Random orders the layers randomly. All algorithms were implemented in Javascript and the experiments were executed on a machine with this setting: (a) Mac OSX 10.9.5; (b) 2.3 GHZ Core i7; (c) 16 GB RAM; (d) Node.js 0.12.2.

To allow comparison of the effect of the number of layers on the result of the algorithms, we executed two families of experiments. In the first family, for each dataset we randomly selected 8 layers, and ordered them using each algorithm. Then, for each ordering we computed a streamgraph with a baseline that minimizes the wiggle (either 1-norm or 2-norm) and computed its wiggle value. This process is repeated

dataset	TwoOpt	TwoOptR	BestFirst	OnSet	D3	Random
unemp.	0.16/0.05	0.17/0.01	0.22/0.23	0.26/0.25	1.00/1.00	0.56/0.42
movies	0.06/0.15	0.06/0.19	0.72/0.91	0.27/0.97	0.34/1.00	1.00/0.88
sandy	0.21/0.44	0.14/0.28	0.15/0.62	0.11/0.50	0.25/0.77	1.00/1.00
stocks	0.17/0.07	0.18/0.12	0.64/0.55	0.64/0.53	0.90/0.88	1.00/1.00
marketcap	0.03/0.02	0.02/0.02	0.17/0.07	0.31/0.40	0.70/0.55	1.00/1.00
google	0.13/0.17	0.14/0.20	0.59/0.44	0.82/0.84	1.00/1.00	0.71/0.74
linux	0.26/0.16	0.32/0.24	0.33/0.34	0.57/0.71	0.52/0.56	1.00/1.00

Table 4.1: Normalized wiggles of algorithms on 8 randomly sampled layers, averaged over 20 iterations. Each entry shows 1-norm/2-norm wiggles. Lower values are better.

dataset	TwoOpt	TwoOptR	BestFirst	OnSet	D3	Random
unemp.	0.00/0.00	0.00/0.02	0.34/0.25	0.14/0.10	1.00/1.00	0.71/0.60
movies	0.00/0.00	0.39/0.07	0.10/0.08	0.10/0.42	0.27/0.46	1.00/1.00
sandy	0.00/0.00	0.32/0.22	0.04/0.06	0.59/0.60	0.30/0.33	1.00/1.00
stocks	0.00/0.00	0.00/0.00	0.76/0.46	0.38/0.50	1.00/0.96	0.91/1.00
marketcap	0.00/0.00	0.19/0.17	0.15/0.32	0.12/0.30	0.58/1.00	1.00/0.99
google	0.00/0.00	0.02/0.05	0.74/0.23	0.93/1.00	1.00/0.88	0.72/0.75
linux	0.00/0.00	0.43/0.33	0.05/0.02	0.83/0.81	0.66/0.57	1.00/1.00

Table 4.2: Normalized wiggles of algorithms on 50 randomly sampled layers, averaged over 20 iterations. Each entry shows 1-norm/2-norm wiggles. Lower values are better.

20 times and the average wiggle value is taken. The second family of experiments is similar to the first, but at every repetition we randomly selected 50 layers. The only exceptions are dataset Unemployment and Google, for which we selected 20 layers because the datasets contained fewer than 50 layers. Algorithm Optimum was not executed, because it would have taken too much time to execute on that number of layers.

Tables 4.1 and 4.2 show the results of the first and the second family of experiments, respectively. Values are normalized by replacing each value x with $x' = \frac{x-min}{max-min}$, where *min* and *max* are the minimum and the maximum wiggle value in the same table row as x. In Table 4.1, *min* was always the wiggle value of algorithm Optimum. This algorithm is not shown, since its normalized wiggle was always 0.

Both tables outline that TwoOpt had the best performance, with the exception of dataset Sandy in Table 4.1. This confirms our hypothesis that an ordering algorithm specifically designed to reduce the wiggle produces drawings with a lower wiggle than other algorithms. TwoOptR had variable performance depending on the dataset,

dataset	TwoOpt	TwoOptR	BestFirst	OnSet	D3	random	ww1a	ww2	labeling
unempl.	205.04	190.28	18.25	0.91	3.89	0.02	5.34	2.03	15.86
movies	38.05	32.24	8.57	0.26	1.03	0.02	0.92	0.49	23.05
sandy	64.93	58.85	9.91	0.28	1.23	0.02	1.11	0.67	28.54
stocks	378.25	365.74	30.78	0.69	2.83	0.02	3.64	1.34	48.10
marketcap	688.67	682.19	63.68	1.29	5.48	0.02	7.47	3.20	112.41
google	174.98	165.79	15.71	0.78	2.52	0.02	4.28	0.89	24.35
linux	46.81	36.93	8.04	0.23	0.96	0.02	0.86	0.44	30.99

Table 4.3: Running times (in milliseconds) of algorithms on 50 randomly sampled layers, averaged over 20 iterations.

which suggests that executing TwoOpt starting from a BestFirst layer ordering give better results with respect to a random ordering.

While Table 4.1 shows that the wiggle of TwoOpt is close to the optimum, the results in Table 4.2 are more representative of a realistic scenario and we further discuss them. Refer also to the figures in the additional material, which represent the drawings produced by the various algorithms on one instance for each dataset in Table 4.2. BestFirst had very poor performance on datasets Stocks and Google. In these examples, some thick layers were put at the two edges of the drawing because they presented relatively high level of wiggle on specific parts due to their thickness. However, they were impacted by the cumulated wiggle of the underlying thinner layers, which caused distortions along the entire layer extent. The greedy approach of BestFirst could not foresee this type of situations, which were adjusted by TwoOpt by moving the thick layers slightly towards the center of the ordering. Dataset Stocks also contained some jumps, for which BestFirst was not designed. OnSet had excellent performance on dataset Movies, which was expected. Good results were obtained also for datasets Unemployment and Marketcap, where many layers contained on-set points. Correctly handling these allowed OnSet to avoid orderings with high values of wiggle. On the other hand, in absence of on-set points, OnSet had poor performance (Sandy, Stocks) which at times are close to that of randomly ordering the layers (Google, Linux).

We illustrate the effect of ordering and baseline algorithms in Fig. 4.9. Clearly D3 (top) has a drifting medium line due to the 2-norm baseline algorithm. In addition the peaks seen in "SAP" caused significant distortion in other layers. Likewise, a drift of medium line is seen in Fig. 4.10. Fig. 4.11 shows the top 20 companies from the same dataset Marketcap, but from Jun 2001 to Jun 2011. In this figure, the effect of the ordering is more visible, because the lightgreen ("SAP") layer in Fig. 4.11a is placed by D3 at the center of the drawing, distorting the other layers.

Table 4.3 shows the running time of the various algorithms, averaged over all instances of Table 4.2. Our ordering algorithms, by nature of trying to optimize, are

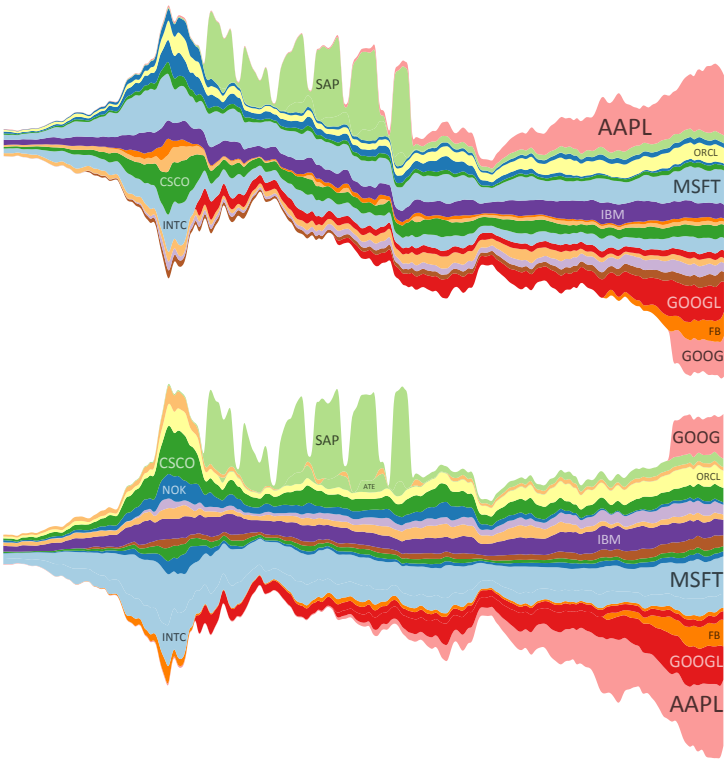


Figure 4.9: Marketcap data for top 50 companies from 1995-2015. Top: D3's implementation of Byron and Wattenberg [BW08]. Bottom: with `TwoOpt` ordering and a baseline algorithm that minimizes weighted 1-norm wiggle.

much slower than the state of the art, namely TwoOpt and BestFirst are slower than OnSet, D3, and Random. TwoOpt had is the slowest. Because of its iterative nature, it scanned all layers several times. Also, we configured it to do a number of layer scans equal to the number of layers, and a constant number of repeats. The number of time points in data greatly contributed to the running time of our algorithms. This is noticeable for datasets Unemployment, Stocks, Marketcap, and Google, which contained many time points. When the number of layers and the number of time points are comparable, our algorithms resulted cubic in the size of the input (see Section 4.6 for a description of the time complexity). The good performance of OnSet

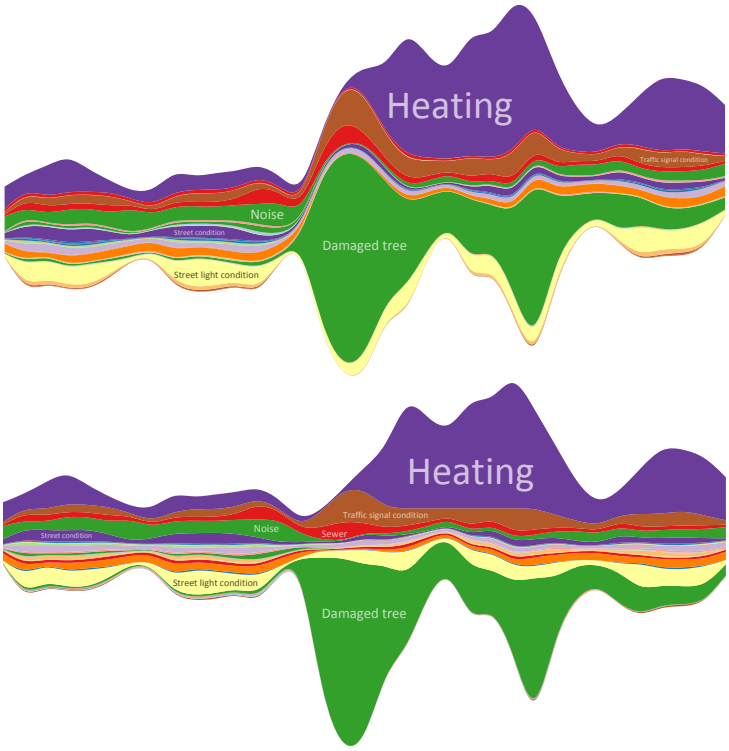
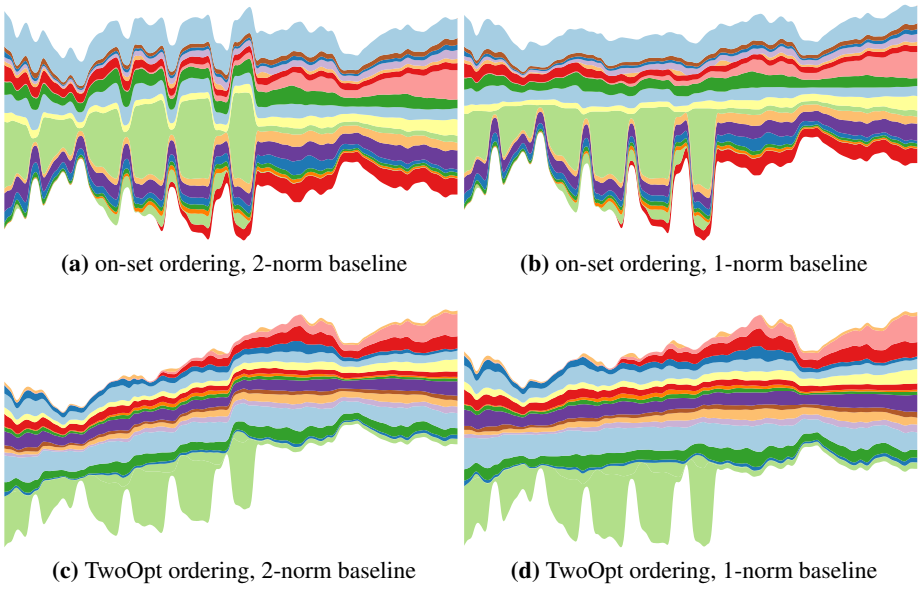


Figure 4.10: Number of calls on 50 topics to the NYC 311 for the days before and after hurricane Sandy hit New York City. Top: D3's implementation of Byron and Wattenberg [BW08]. Bottom: with `TwoOpt` ordering and the 1-norm baseline algorithm. Notice that the top streamgraph has an up-drift of the median line, which is remedied on the bottom with the new baseline algorithm.

and D3 were mainly due to the simplicity of their logic, which allowed for very efficient implementations. OnSet performed better than D3 because it stops scanning a layer at the first non-zero value, while D3 performs a full scan. We conclude that considering the wiggle for ordering the layer requires complex algorithms with non negligible running times. However, it is worth pointing out that running time in each of the ordering algorithm is below 1 second, which is acceptable for real uses. The 1-norm and 2-norm baseline algorithms (ww1a and ww2) both take very little time. The labeling algorithm is also very fast. We did not compare with the labeling algo-



(a) on-set ordering, 2-norm baseline

(b) on-set ordering, 1-norm baseline

(c) TwoOpt ordering, 2-norm baseline

(d) TwoOpt ordering, 1-norm baseline

Figure 4.11: Top 20 companies from dataset Marketcap, from Jun 2001 to Jun 2011, with different layer orderings and baselines. (a) The light-green layer is at the center of the drawing, distorting the other layers. (b) Because of the 1-norm baseline, the distortion caused by a wrong ordering of the light-green layer is limited to the lower half of the drawing. (c) The light-green layer "pushes" up the other layers because of the 2-norm baseline, creating an upward-going drawing. (d) None of the previous issues are present.

rithm of [BW08] since the latter was described as a brute-force algorithm with "poor real-time performance", too brief a description to allow a proper implementation.

4.8 Discussions

We proposed a 1-norm based baseline algorithm which overcomes the distortion associated with 2-norm based baseline algorithm. We note that both definitions of wiggle are reasonable to use when there are not sudden jumps. Compared with the 2-norm baseline algorithm, the 1-norm based baseline algorithm tend to give relatively flat middle layers. It is not clear to us which definition, or an alternative definition, correlates best with our visual perception of wiggle. For example, in the current definitions,

wiggles are additive over time. A time series $\{y, y+1, y+2\}$ is considered to have the same wiggle as $\{y, y+1, y\}$, but perceptually the former is smoother, while the latter represent a spike. Thus a wiggle definition based on both first- and second-order derivatives may be more expressive.

In the presentation of our fast layer labeling algorithm, for simplicity we assumes that labels have integer width. This is reasonable for time series with hundreds or more points, for which integer width is fine grain enough. For time series with few points, if a feasible position for a label is not found, our algorithm adaptively interpolate the time series by doubling the number of points, and apply the layer labeling procedure to the new time series, until a feasible position for the label is found. This effectively make the label width a floating point number.

4.9 Conclusions and Future Work

The aesthetics of a streamgraph is affected by the ordering of the layers, the shape of the baseline of the drawing, and the labeling for the layers. This chapter advances the state of the art for streamgraphs by proposing an ordering algorithm that works well regardless of the properties of the input data, a 1-norm baseline procedure that overcomes the distortion associated with the existing baseline algorithm, particularly when there are sharp changes in the time series, and an efficient layer labeling algorithm that scales linearly to the data size. We demonstrate both qualitatively and quantitatively the advantage of our algorithms over existing techniques on a number of real world data sets.

For future work, we like to conduct a user study to measure how the user's perception of the smoothness of a streamgraph correlates with the various measure of wiggles. In addition we are investigating possible ways to use streamgraphs to visualize time series with both positive and negative values.

Part II

Abstract Representation of Routing

Chapter 5

Automatic Discovery of High-Impact Routing Events

This chapter describes an approach based on *Routing Events* (see Chapter 1) for automatically inferring high-level routing dynamics from a large set of traceroute data. The tackled challenges are *Data Size* and *Dynamics*. Events are an abstraction of the dynamics of routing. This work first gives a precise definition of what a routing event is, and then describes an algorithm for finding events with high-impact, that is, whose effects on routing impacted many probes of a probe system. Experiments were conducted on real data related to the ASes of two European ISPs. A preliminary version of this chapter was published in [DDP+15b].

5.1 Introduction

One of the primary goals of a network operator is to ensure its network works as expected. Since misbehaviors can happen for a variety of reasons, constant monitoring is performed by operators to timely detect problems and limit users complaints. Directly monitoring the health of each network element works in many situations, but may fall short when the element itself lacks the necessary agent support or is not under the operator's control. Also, there are cases in which network elements are reported as working despite end-to-end communication being impaired by misconfigurations or subtle hardware failures (the *silent failures* in [KYGS]).

A large corpus of research works has focused on methodologies to detect and locate faults using information collected by hardware or software elements (called *monitors* or *probes*) deployed throughout the network, possibly far away from the

problem. Indeed, large probing networks are already running to ease the assessment of service levels by regulators (e.g., [sam15]) or for management and scientific purposes (e.g., [atl15,ark15,mla15]). A widely discussed approach to localize the faults on a network consists in correlating end-to-end measurements from a large number of probes using a technique called *binary tomography* [Duf06,DTDD07,KYGS]. However, several problems hinder the application of this approach in a production environment [CTFD09,HFT08]: false positives in the detection phase, the failure to accommodate network dynamics, and the need for a complete knowledge of the topology and for a synchronization among the probes. Other approaches take advantage of control plane information [JCC+13,FMM+04], which may require a complex collecting infrastructure or may just not be available to the operator for the part of the network that is not under his control.

We introduce a novel methodology and an algorithm that enable the analysis of large collections of traceroute measurements in search of significant changes, thus easing management and troubleshooting. Our methodology takes as input only a set of traceroutes, identifies paths that evolve similarly over time, and reports them aggregated into inferred events (e.g., routing changes, loss of connectivity), augmented with an impact estimate and a restricted set of IP addresses that are likely close to the cause of the event (a piece of information similar to those provided by tomography-based techniques). The methodology, as well as its correctness, are founded on the notion of *empathy*, a relation that binds similarly behaving traceroutes, which are therefore a good evidence of the same network event. Our approach does not need a-priori knowledge of the network topology, does not assume a stable routing state, and does not impose restrictions on the schedule of traceroutes, which may be collected asynchronously and at arbitrary intervals. Instead, it takes advantage of asynchronous measurements to improve the timeliness and precision of event detection, and is almost unaffected by measurement errors (e.g., due to software errors or routing anomalies), which in most cases only generate fictitious events with a small impact.

We provide experimental evidence of the effectiveness of our approach by running our algorithm on data collected by large-scale measurement infrastructures such as RIPE Atlas, and by comparing the inferred events with ground truth derived from induced routing changes or third-party information.

The rest of the chapter is organized as follows. In Section 5.2 we review related contributions. In Section 5.3 we describe our network model and the fundamental properties of empathy. In Section 5.4 we introduce a methodology and an algorithm, based on empathy, to infer events and report relevant data about them. In Section 5.5 we analyze the results of the application of our methodology to real-world data. In Section 5.6 we draw conclusions and present ideas for future work.

5.2 Related Work

There are many large-scale platforms that collect traceroute measurements (e.g., [ark15, atl15,sam15]), and a standardization effort is also ongoing [BEB+13]. As a consequence, there is a growing interest in finding patterns in such measurements, as confirmed, for example, in [Bro14]: in this paper the authors search for events by clustering data from [atl15] according to distance metrics that determine the amount of difference between subsequent traceroutes. While we also aim at grouping traceroute changes into events, our approach is based on the novel concept of empathy and is provably correct and complete.

A large number of contributions focus on identifying the location or the root cause of a fault based on data gathered by measurement networks. The binary tomography approach, firstly proposed for trees [Duf06] and then extended to general topologies [DTDD07,KYGS], has applicability problems which have been discussed in [CTFD09,HFT08,MHS+14]. Most notably, the authors assume at least a partial knowledge of the network topology (which must often be inferred from input data). Similar approaches to root cause analysis have also been described for interdomain routing data [FMM+04,JCC+13]. A number of systems combine control plane information with data plane measurements: for example, Hubble [KBMJ+08], LIFEGUARD [KBSC+12], NetDiagnoser [DTDD07], as well as [KYGS]. Our approach relies on traceroutes only, does not assume any knowledge of the network topology, and does not impose restrictions on the schedule of traceroutes.

5.3 The Empathy Relationship

In this section we describe the model we use to analyze traceroute paths and we introduce the concept of *empathy*, which is at the basis of our event inference method.

Let $G = (V, E)$ be a graph that models an IP network: vertices in V are network devices (routers or end systems), and edges in E are links between devices. Some devices in V, called *network probes* or *sources*, periodically perform traceroutes towards a predefined set of *destinations*. We assume that each traceroute is acyclic (otherwise there is evidence of a network anomaly) and instantaneous (reasonable because in the vast majority of cases traceroutes terminate within a smaller time scale than that of routing changes).

Let $i = (s, d)$, where $s \in V$ is a source and $d \in V$ is a destination. A *traceroute path* $p_i(t)$ measured at time t by s towards d is a sequence $\langle v_1\ v_2\ \ldots\ v_n \rangle$ such that $v_1 = s$, $v_j \in V$ for $j = 1, \ldots, n$, and there is an edge in E for each pair (v_k, v_{k+1}), $k = 1, \ldots, n-1$. While we include the source in $p_i(t)$, the destination may not appear because a

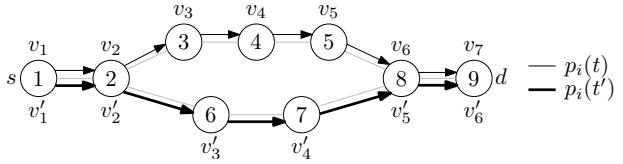


Figure 5.1: An example of two traceroute paths from s to d collected at different time instants t and t'. Gray lines represent network links.

traceroute may end at an unintended vertex different from d. For convenience, let $V(p)$ be the set of vertices of path p.

Now, consider two traceroute paths $p_i(t) = \langle v_1\ v_2\ \ldots\ v_n \rangle$ and $p_i(t') = \langle v'_1\ v'_2\ \ldots\ v'_m \rangle$ between the same source-destination pair $i = (s,d)$, with $t' > t$, and assume that $p_i(t) \neq p_i(t')$. Fig. 5.1 shows two such paths: $i = (1,9)$, $p_i(t) = \langle 1\ 2\ 3\ 4\ 5\ 8\ 9 \rangle$, and $p_i(t') = \langle 1\ 2\ 6\ 7\ 8\ 9 \rangle$. Since the path from s to d has changed between t and t', we call the pair consisting of $p_i(t)$ and $p_i(t')$ a *transition*, indicated by τ_i, and say that it is *active* at any time between the *endpoints* t and t', excluding t'. To analyze the path change, we focus on the portion of the two paths that has changed in the transition: let $\delta^{\text{pre}}(\tau_i)$ indicate the shortest subpath of $p_i(t)$ that goes from a vertex u to a vertex v such that all the vertices between s and u and between v and the end of the path are unchanged in $p_i(t')$. If there is no such v (for example because the destination is unreachable at t or t'), $\delta^{\text{pre}}(\tau_i)$ goes from u to the end of $p_i(t)$. Referring to the example in Fig. 5.1, it is $\delta^{\text{pre}}(\tau_i) = \langle 2\ 3\ 4\ 5\ 8 \rangle$. We define $\delta^{\text{post}}(\tau_i)$ as an analogous subpath of $p_i(t')$. Referring again to Fig. 5.1, it is $\delta^{\text{post}}(\tau_i) = \langle 2\ 6\ 7\ 8 \rangle$. In principle, $\delta^{\text{pre}}(\tau_i)$ may have several vertices in common with $\delta^{\text{post}}(\tau_i)$ besides the first and the last one: we still consider $\delta^{\text{pre}}(\tau_i)$ as a single continuous subpath, with negligible impact on the effectiveness of our methodology. The same applies to $\delta^{\text{post}}(\tau_i)$.

We can now introduce the concept of empathy, that determines when two traceroute paths exhibit a similar behavior over time. Consider two transitions τ_1, with source s_1 and destination d_1, and τ_2, with source s_2 and destination d_2, such that both transitions are active between t and t', $t' > t$, at least one has an endpoint in t, and at least one has an endpoint in t'. We say that (s_1,d_1) *and* (s_2,d_2) *are pre-empathic at any time* $t \leq \hat{t} < t'$ if the portions of $p_1(t)$ and $p_2(t)$ that change during τ_1 and τ_2 overlap, namely $V(\delta^{\text{pre}}(\tau_1)) \cap V(\delta^{\text{pre}}(\tau_2)) \neq \emptyset$. Intuitively, traceroute paths relative to pre-empathic sd-pairs stop traversing a network portion that they shared before an event occurred. Similarly, we say that (s_1,d_1) *and* (s_2,d_2) *are post-empathic at any time* $t \leq \hat{t} < t'$ if $V(\delta^{\text{post}}(\tau_1)) \cap V(\delta^{\text{post}}(\tau_2)) \neq \emptyset$. Post-empathy captures a different kind of path change: traceroute paths of post-empathic sd-pairs start traversing

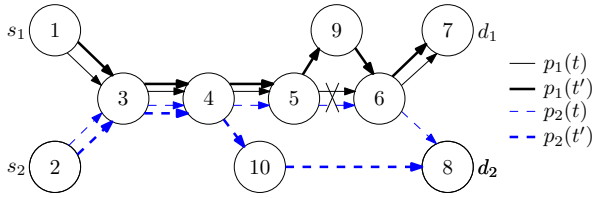


Figure 5.2: An example showing empathy relations. In this scenario link $(5,6)$ fails, and we have $(s_1,d_1) \overset{\text{pre}}{\sim}_t (s_2,d_2)$ but $(s_1,d_1) \overset{\text{post}}{\not\sim}_t (s_2,d_2)$.

a common portion that they did not use before the event occurred. Fig. 5.2 shows two traceroute paths p_1, from s_1 to d_1, and p_2, from s_2 to d_2, that change between t and t' due to the failure of link $(5,6)$. Considering the corresponding transitions τ_1 and τ_2, we have $\delta^{\text{pre}}(\tau_1) = \langle 5\ 6\rangle$, $\delta^{\text{post}}(\tau_1) = \langle 5\ 9\ 6\rangle$, $\delta^{\text{pre}}(\tau_2) = \langle 4\ 5\ 6\ 8\rangle$, and $\delta^{\text{post}}(\tau_2) = \langle 4\ 10\ 8\rangle$. Since $\delta^{\text{pre}}(\tau_1)$ and $\delta^{\text{pre}}(\tau_2)$ share vertices 5 and 6, (s_1,d_1) and (s_2,d_2) are pre-empathic between t and t', whereas (s_1,d_1) and (s_2,d_2) are not post-empathic despite the fact that $p_1(t')$ and $p_2(t')$ share a subpath. Indeed, p_1 and p_2 behave similarly before the link fails and change to two independent routes afterwards.

In order to understand how empathy is useful to infer network events, we need to formally introduce the notion of event, qualifying it as physical to distinguish it from events inferred by our algorithm. We call *physical event* at time $\bar{t}$ the simultaneous disappearance of a set $E^{\downarrow}$ of links from E (*down event*) or the simultaneous appearance of a set $E^{\uparrow}$ of links in E (*up event*), such that:

- either $E^{\downarrow} = \emptyset$ or $E^{\uparrow} = \emptyset$ (a physical event is either the disappearance or the appearance of links, not both);
- $E^{\downarrow} \subseteq E$ (only existing links can disappear);
- $E^{\uparrow} \cap E = \emptyset$ (only new links can appear);
- $\exists v \in V \mid \forall (u,w) \in E^{\downarrow} : u = v$ or $w = v$, and the same holds for $E^{\uparrow}$ (all disappeared/appeared edges have one endpoint vertex in common). Vertex v is called *hub* of the event (an event involving a single edge (u,v) has two hubs: u and v; any other event has a unique hub).

When the type of an event is not relevant, we indicate it as $E^{\downarrow\uparrow}$. This event model captures the circumstance in which one or more links attached to a network device fail or are brought up, including the case in which a whole device fails or is activated. Such

events may be caused, for example, by failures of network interface cards, line cards, or routers, by accidental link cuts, by provisioning processes, and by administrative reconfigurations. Congestion is normally not among these causes, but may be detected as an event if it makes a balancer shift traffic away from a set of links. Failures or activations of links that do not have a vertex in common are considered distinct events. We only consider *visible* physical events, namely those that cause at least a transition to occur. Moreover, we assume that every transition comprises at least one edge involved in an event, an assumption that is long-argued in the literature about root-cause analysis (see, e.g., [JCC+13]) and yet we deem reasonable because our goal is to detect events, not reconstruct their original cause. Given a physical event $E^{\downarrow\uparrow}$ occurred at time $\bar{t}$, we define the *scope* $S(E^{\downarrow\uparrow})$ of $E^{\downarrow\uparrow}$ as the set of sd-pairs $i = (s,d)$ involved in the transitions that are active at $\bar{t}$. We also call *impact* of $E^{\downarrow\uparrow}$ the cardinality of $S(E^{\downarrow\uparrow})$.

Intuitively, our algorithm infers network events based on observed transitions and on empathy relationships that bind the involved sd-pairs: empathies are considered a good evidence of path changes that are due to the same physical event.

5.4 Seeking Events: Methodology and Algorithm

In this section, we describe our inference algorithm for detecting routing events. The algorithm takes as input a set of traceroute paths, and produces as result a list of inferred events, each equipped with the following information: i) an interval of time in which the event is supposed to have occurred, ii) a set of sd-pairs affected by the event (the scope), iii) the type (up, down) of the event (when inferred), and iv) a set of IP addresses that, after the event has occurred, (dis)appeared in all the traceroutes performed between sd-pairs in its scope (these IPs are good hints for identifying the cause of the event).

We refer to the model illustrated in the previous section, considering the general case of non-synchronized traceroute measurements. That is, for an sd-pair $i = (s,d)$ traceroute paths $p_i(t)$ are only available at specific time instants $t \in \mathbb{R}$ that depend on s (if probes are synchronized via NTP, whose precision is high enough for our needs, we can refer to a universal clock). As we will show later, unsynchronized traceroutes can improve the accuracy of the interval reported by our algorithm for an inferred event. For convenience, for a transition τ_i we define the *changed set* $\Delta(\tau_i)$ consisting of *extended addresses*, namely IP addresses in $V(\delta^{\text{pre}}(\tau_i))$ labeled with a tag pre and IP addresses in $V(\delta^{\text{post}}(\tau_i))$ labeled with a tag post.

Our algorithm consists of three phases.

Phase 1 – Identification of transitions: in this phase, for each sd-pair i, input

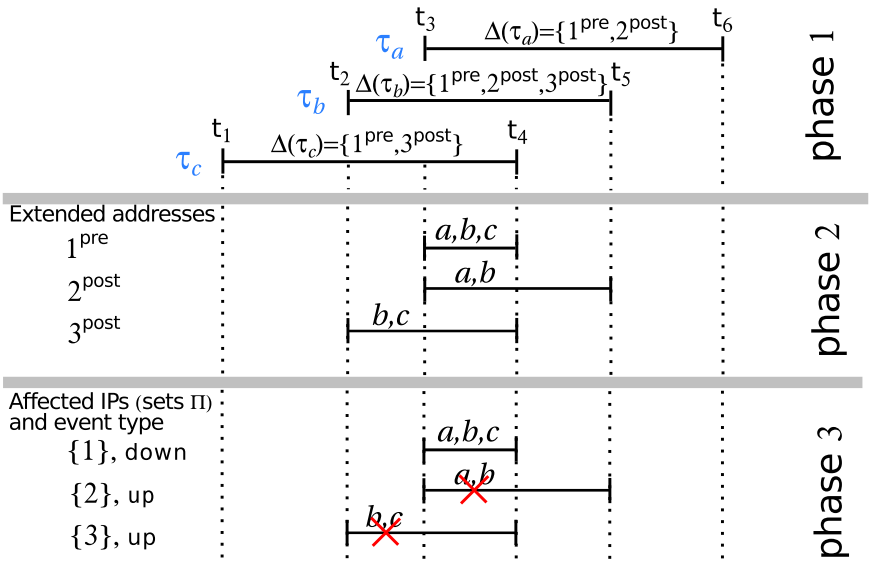


Figure 5.3: Sample outputs of the various phases of our algorithm.

samples $p_i(t)$ are scanned and all transitions τ_i, with the corresponding $\Delta(\tau_i)$, are identified. The upper part of Fig. 5.3 shows an example with 3 transitions τ_a, τ_b, and τ_c, represented as segments terminating at the transitions' endpoints, and the corresponding changed sets (IP addresses are represented as numbers). The transitions in the figure can be the consequence of a physical down event with hub 1.

Phase 2 – Construction of candidate events: in this phase, the algorithm tracks the evolution of empathy relationships between sd-pairs involved in transitions. As detailed in Fig. 5.4, the algorithm linearly sweeps on time instants corresponding to transition endpoints and, for every instant t and every extended address $\mathcal{A}$ (lines 2 and 3), updates a set $S_\mathcal{A}$ of sd-pairs i corresponding to active transitions that are empathic with each other because they have $\mathcal{A}$ in their changed set $\Delta(\tau_i)$ (line 6). Sets $S_\mathcal{A}$, as well as the time instants $t_\mathcal{A}$ at which they are updated, are kept in special variables which allow access to the last 2 assigned values using operators $prev()$ and $pprev()$. When the size of each $S_\mathcal{A}$ reaches a local maximum at time t (line 7), the algorithm reports a candidate event. This corresponds to seeking for the time instant at which the highest number of sd-pairs have seen IP address $\mathcal{A}$ (dis)appear in their traceroute paths. The interval $[prev(t_\mathcal{A}), t_\mathcal{A}]$ of validity of the local maximum (line 8) is a good

Input: a set T of transitions
Output: a set *CEvents* of candidate events, namely tuples $(t_1, t_2, S, \mathcal{A})$ indicating time intervals in which all the sd-pairs in S are pre-empathic or post-empathic with each other and all the corresponding transitions τ_i have extended address $\mathcal{A}$ in their changed set $\Delta(\tau_i)$.
▷ $S_\mathcal{A}$ and $t_\mathcal{A}$ are special variables for which the last two assigned values can be accessed by $prev()$ and $pprev()$ (if unset, they are $\emptyset$ and $-\infty$).

```
1:  E = ∅
2:  for every endpoint t of transitions in T, in order of time do
3:      for every address A in the changed set of any transitions do
4:          T_A = set of transitions τ_i active at t such that A ∈ Δ(τ_i)
5:          sdp = the union of sd-pairs of transitions in T_A
6:          if S_A ≠ sdp then S_A = sdp; t_A = t
7:          if |pprev(S_A)| ≤ |prev(S_A)| and |prev(S_A)| > |S_A| then
8:              add (prev(t_A), t_A, prev(S_A), A) to CEvents
9:          end if
10:     end for
11: end for
12: return CEvents
```

Figure 5.4: Phase 2 of our algorithm: it computes candidate events by finding sets of sd-pairs that are all pre-empathic or post-empathic with each other.

candidate for being the time window within which an event has occurred. The middle part of Fig. 5.3 shows a sample output of this phase, where each segment represents a candidate event: for each extended address appearing in the changed sets of τ_a, τ_b, and τ_c, the corresponding sets of sd-pairs $S_{1^{\text{pre}}}$, $S_{2^{\text{post}}}$, and $S_{3^{\text{post}}}$ that involved that address are constructed and updated. In particular, set $S_{1^{\text{pre}}}$ reaches its maximum size at time t_3, when extended address 1^{pre} is in the changed set of τ_a, τ_b, and τ_c, namely IP address 1 has disappeared for sd-pairs a, b, and c. The reported candidate event ends at t_4, when the size of $S_{1^{\text{pre}}}$ is again reduced: it is therefore $(t_3, t_4, \{a, b, c\}, 1^{\text{pre}})$. Similar considerations apply for the construction of the other two candidate events $(t_3, t_5, \{a, b\}, 2^{\text{post}})$ and $(t_2, t_4, \{b, c\}, 3^{\text{post}})$.

Phase 3 – Event inference: in this phase, detailed in Fig. 5.5, candidate events are sieved to build a set of inferred events, each consisting of a time window, a scope, a set of involved IP addresses (which contains the hub of the event), and a type (up/down/unknown). As a first clean-up step, all candidate events whose set of sd-pairs is properly contained in the set of sd-pairs of another candidate event that overlaps in time are discarded (lines 2-6). In this way, only events with maximal impact are reported. Afterwards, the algorithm considers groups $CEvents(S, t_1, t_2)$ of candidate events spanning the same time interval $[t_1, t_2]$ and having S as set of sd-pairs (line 7), and constructs an inferred event for every set S whose size exceeds a configured *threshold*: this filters out events with negligible impact. The inferred

Input: a set *CEvents* of candidate events produced in phase 2 (see Fig. 5.4)
Output: a set *Events* of tuples $(t_1, t_2, S, \Pi, type)$, each representing an inferred event occurred between t_1 and t_2, whose scope is S, which involved the IP addresses in Π, and whose type is *type*.

1: $Events = \emptyset$
2: **for** every pair $e = (t_1, t_2, S, \mathcal{A})$, $\tilde{e} = (\tilde{t}_1, \tilde{t}_2, \tilde{S}, \tilde{\mathcal{A}})$ in *CEvents* **do**
3: **if** e and $\tilde{e}$ overlap in time and $\tilde{S} \subset S$ **then**
4: remove $\tilde{e}$ from *CEvents*
5: **end if**
6: **end for**
7: group candidate events $(t_1, t_2, S, \mathcal{A})$ in *CEvents* by S, t_1, and t_2
8: **for** every computed group $CEvents(S, t_1, t_2)$ **do**
9: **if** $|S| >$ *threshold* **then**
10: $eaddr = \bigcup_{(t_1, t_2, S, \mathcal{A}) \in CEvents(S)} \mathcal{A}$
11: Π = *eaddr* (without labels)
12: *type* = unknown
13: *type* = down if all addresses in *eaddr* are tagged as pre
14: *type* = up if all addresses in *eaddr* are tagged as post
15: add $(t_1, t_2, S, \Pi, type)$ to *Events*
16: **end if**
17: **end for**
18: **return** *Events*

Figure 5.5: Phase 3 of our algorithm: it reports inferred events starting from a set of candidate events produced in Phase 2.

event has the following structure (lines 10-14): the time interval is $[t_1, t_2]$; the scope is S; the involved IP addresses are the union of the addresses of candidate events in $CEvents(S, t_1, t_2)$; and the type is inferred based on the labels of the extended addresses of candidate events in $CEvents(S, t_1, t_2)$ (lines 12-14). A sample result of the application of this phase is in the lower part of Fig. 5.3: candidate events $(t_3, t_5, \{a, b\}, 2^{\text{post}})$ and $(t_2, t_4, \{b, c\}, 3^{\text{post}})$ (segments in the second and third row of phase 2, respectively) are discarded because their sets of sd-pairs are contained in the one of the overlapping candidate event $(t_3, t_4, \{a, b, c\}, 1^{\text{pre}})$. At this point, there is only one set of sd-pairs left, $\{a, b, c\}$: assuming no threshold, the only candidate event having such set is reported as an event, which affected IP address 1 (that is also the hub of the event) and whose type is down because of the label of 1^{pre}.

Our algorithm is correct and complete, as stated by the following theorems.

Theorem 2 (Correctness)**.** *Each event inferred by our algorithm corresponds to a physical event.*

Proof: Let $(t_1, t_2, S, \Pi, type)$ be an inferred event. By construction, every address π in Π has (dis)appeared in all transitions τ_i for every $i \in S$, and S has maximal size: therefore π is a candidate for being the hub of a physical event. Moreover, since

π (dis)appears in traceroutes in the interval in which transitions τ_i intersect, namely between t_1 and t_2, this is also the time window in which the event has occurred. □

Theorem 3 (Completeness)**.** *For every visible physical event, an inferred event is reported by our algorithm.*

Proof: Suppose a physical event $E^{\downarrow}$ with hub h occurs at time $\bar{t}$: the traceroutes for all sd-pairs i that are in the scope $S(E^{\downarrow})$ of $E^{\downarrow}$ will therefore change after $\bar{t}$, and phase 1 of the algorithm constructs transitions τ_i whose intervals contain $\bar{t}$. All such transitions must intersect at a common interval $[t_1,t_2]$ comprising $\bar{t}$ and have $h^{\text{pre}} \in \Delta(\tau_i)$. By the definition of scope, the cardinality of set $S_{h^{\text{pre}}}$ reaches a local maximum between t_1 and t_2 in phase 2 of the algorithm, and a candidate event $e = (t_1,t_2,S(E^{\downarrow}),h^{\text{pre}})$ is thus constructed. Set $S(E^{\downarrow})$ is the largest possible set of sd-pairs affected by $E^{\downarrow}$, therefore e is not filtered in phase 3 and an event $(t_1,t_2,S(E^{\downarrow}),\Pi,type)$ with $h \in \Pi$ is reported. Analogous arguments can be applied to the case of an event $E^{\uparrow}$. □

The computational complexity of our inference algorithm is $O(|T|+|CEevents|^2 \cdot I)$, where T is the set of transitions and I is the maximum impact. In fact, phase 1 takes $O(|T|)$. Since the size of the changed set of every transition is bounded by the maximum length of traceroute paths and sets $T_{\mathcal{A}}$ and sdp can be updated during the sweep, phase 2 also takes $O(|T|)$. Phase 3 takes $O(|CEvents|^2 \cdot I)$ because of the overlap check at lines 2-6 (the following event construction can be performed efficiently by scanning candidate events).

Several issues are inherent in using real-world traceroute data, but our model and algorithm can effectively cope with them, as also confirmed by experimental results. First of all, a single network device equipped with multiple network interfaces (e.g., a router) may reply with different IP addresses in different traceroutes, a phenomenon known as *aliasing* [MPP13]. As a consequence, detection of some empathies may fail, causing our algorithm to infer multiple small events rather than a single larger one in the worst case. Delays in the propagation of routing changes (for example due to routing protocol timers) have a similar effect on the algorithm's output, which may include multiple copies of the same event with slightly different time intervals and scopes. On the other hand, multiple simultaneous events can interfere with each other, namely the corresponding transitions may overlap and their changed sets may have elements in common. Such events can still be detected by our algorithm, even if their scope can only be identified with a limited precision. Under rare circumstances, some fictitious or improperly time-skewed events may also be inferred. In practice, none of these cases prevents our algorithm from reporting events, and their incidence was negligible in our experiments.

One aspect that may indeed taint the output of our algorithm is that the vast majority of Internet paths traverse load balancers [AFT07]: they are the cause of a high number of apparent routing changes which may be improperly reported as physical events. Compensating this issue requires knowledge of the load balancers, which is realistic for an Internet Service Provider that wants to apply our methodology, and can otherwise be constructed by applying discovery techniques such as Paris Traceroute [ACO+06b]. Unfortunately, this technique was not yet available in the measurement networks we considered, therefore we preprocessed traceroutes by using a simple heuristic that cleaned up most of the noise introduced by load balancers: we analyzed all the input traceroutes in their time order and, for each destination, we tracked the evolution over time of the routing (actually the next hop) of every node along the traceroute paths. Nodes with unstable routing (i.e., that change the next hop in more than 20% of the samples) are considered to belong to a load balancer and their next hops are replaced by a single arbitrarily chosen representative IP address.

5.5 Experimental Results

We executed our algorithm on several sets of traceroute paths collected by currently active measurement networks, with the intent to verify that the inferred events matched physical events. We first considered traceroutes affected by a sequence of routing changes injected with a known schedule, used as ground truth. We then used our algorithm to detect spontaneous events happened in the network of a European operator.

Induced Event Analysis

For this experiment we partnered with an Italian ISP that has BGP peerings with three main upstream providers and with a number of ASes at three Internet eXchange Points (IXPs), i.e. MIX, NaMeX (the main IXPs in Italy), and AMS-IX[1]. An IP subnet reserved for the experiment was announced via BGP to different subsets of peers, according to the schedule in Table 5.1. During the experiment, 89 RIPE Atlas probes located in Italy were instructed to perform traceroutes every 10 minutes (between 2014-05-02 13:00 UTC and 2014-05-03 15:00 UTC) targeting a host inside the reserved subnet. After applying the load balancers cleanup heuristic described in Section 5.4, we fed our algorithm with the collected traceroute measurements.

The produced output, which took only a few seconds to compute, is plotted in Fig. 5.6: each inferred event $(t_1, t_2, S, \Pi, type)$ is represented by a point whose coor-

[1] NaMeX and AMS-IX are connected by a link. However, it was not used by any means in our specific setting.

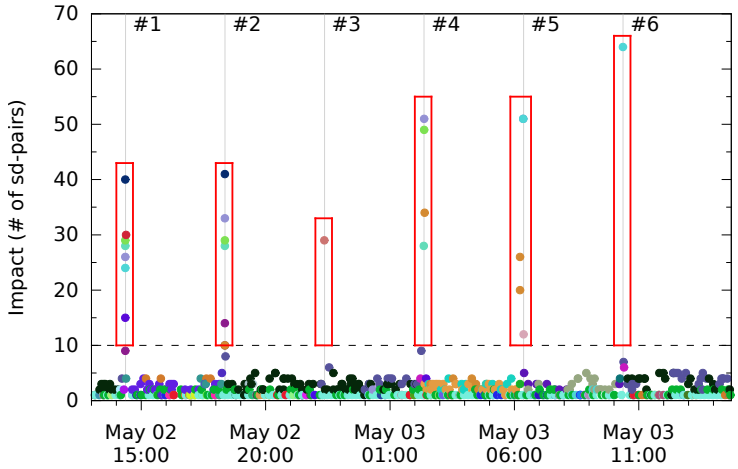


Figure 5.6: Impacts of the events inferred during experiment 1 (induced events).

dinates are the center of interval $[t_1, t_2]$ (X axis) and the event's impact $|S|$ (Y axis), and whose color identifies a specific set Π of involved IP addresses. Out of all the inferred events, 23 exceeded the impact threshold of 10 (dashed horizontal line in the figure), which clearly separates them from background noise. It is evident that these 23 events tend to concentrate (red boxes) around the time instants of BGP announcements (vertical gray lines numbered according to the rows of Table 5.1), and indeed the center of the time interval of each event falls within seconds from the corresponding announcement. In addition, the maximum extension of each interval $[t_1, t_2]$ was 2 minutes, confirming that our methodology can detect an event very quickly after the instant in which it actually happened. Set Π consisted of a single IP address for 87% of the events and of at most 4 IP addresses for 2 events, demonstrating a high precision in pointing out possible event causes.

For at least one announcement change (#3) the detection was optimal, namely we inferred a single event where all the 29 involved sd-pairs switched from MIX to NaMeX. Multiple events were instead inferred in the other cases, due to asterisks in traceroutes or interference between routing changes happening close in time to each other. One of the inferred events even allowed us to discover an undeclared backup peering whose existence was later confirmed by the ISP. Further details can be found in [DDP+14].

Table 5.1: Schedule of BGP announcements for the controlled experiment in Section 5.5.

	Time	Upstreams	MIX	NaMeX	AMS-IX
	May 02, before 14:22	✓	✓	✓	✓
#1	May 02, 14:22	✓			
#2	May 02, 18:22		✓	✓	
#3	May 02, 22:22			✓	
#4	May 03, 02:22		✓		
#5	May 03, 06:22				✓
#6	May 03, 10:22	✓	✓	✓	✓

Spontaneous Event Analysis

For the second experiment, we considered traceroute paths collected every 8 hours by 3320 probes distributed within the network of a European operator, called EOp in the following for privacy reasons. Traceroutes were performed towards destinations located both inside and outside EOp's network. In a private communication EOp informed us about a "routing failure" in one of its ASes occurred on day D, therefore we focused on traceroutes collected in a 9-days time window comprising this day. We applied our load balancers cleanup heuristic on a slightly richer data set consisting of almost 260.000 traceroutes collected over 15 days. Our algorithm then took about 3 minutes to completely process the cleaned set of traceroutes, computing almost 60.000 transitions in phase 1. We separately ascertained that the load balancers heuristic reduced the number of transitions by almost a factor of 3 and the number of inferred events by almost a factor of 20. For an improved accuracy, we filtered out inferred events that did not involve (in Π) any IP addresses within EOp's network, obtaining the events in Fig. 5.7.

Considering the average impact of the inferred events, we set the threshold at 100 (dashed horizontal line in the figure). The figure shows that events exceeding this threshold are mainly concentrated within a time window whose center falls within 24 hours from Day D, and are followed by some less impactful events occurring up to 2 days later (red box in the figure), totaling 199 events involving 838 unique sd-pairs. Our algorithm also singled out their candidate causes pretty accurately, given that the union of all sets Π consisted of as few as 7 IP addresses. Considering the frequency of traceroutes, these events were also somewhat precisely located in time: the length of their time intervals ranged from about 10 hours to as low as 1 second (due to traceroutes not being synchronized), with a standard deviation of 30 minutes.

As it can be seen from the figure, reported events are rather fragmented despite

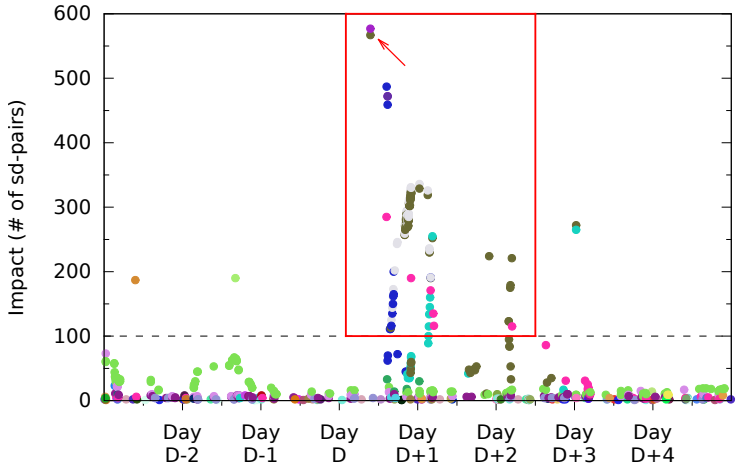


Figure 5.7: Impacts of the events inferred in experiment 2 (spontaneous events).

affecting the same set of IP addresses (points with the same colors in the figure): this is due to the fact that routing propagation delays caused many non-overlapping transitions to be constructed in phase 1. Interestingly, the two events with impact higher than 550 (indicated by an arrow in the figure) were of type down and up, indicating that all the traceroute paths of the involved sd-pairs switched to alternate routes sharing some common IP addresses (all within EOp's network). Events whose time window is centered between $D+1$ and $D+2$ are likely due to configuration changes undertaken to restore a working routing.

After submitting our results to EOp, they confirmed that inferred events with outstanding impacts had very good match with the incident, and subsequent events corresponded to actions aimed at restoring the full operational state.

We believe these experiments highlight the effectiveness of our methodology in detecting events, regardless of whether they are induced and recurring (experiment 1) or spontaneous and isolated (experiment 2), and there is evidence that detection can happen very quickly after the occurrence of an event.

5.6 Conclusions and Future Work

We have presented a model and methodology for the identification and analysis of network events based on the notion of empathic traceroute measurements. We have translated our theoretical approach into an algorithm and applied it to real-world data, proving the effectiveness of our methodology.

We plan to further validate our approach with other measurement platforms (see Section 5.2 for examples), topologies, and network events. We will focus in particular on intra-domain routing events as opposed to BGP routing changes. Further, we will study heuristics to merge two or more inferred events that are likely to represent one single network event, and work on devising an on-line version of our algorithm, which could effectively integrate the core of an alerting system.

Chapter 6

Visual Analysis of Routing Events

This chapter describes an approach based on *Routing Events* (see Chapter 1) for visually analyzing events of the type introduced in Chapter 5. The tackled challenges are *Data Size*, *Dynamics*, and *Relation To Geography*. Events are an abstraction of routing dynamics and are multivariate entities. The algorithm in Chapter 5 can automatically detect events from a set of traceroutes, thus reporting that some routing activity happened at a given moment, impacting some given probes, and involving some given intermediate nodes. However, events are treated as single objects, while the relationships between them can reveal patterns that shed further light on the routing dynamics. The described visualization approach allows the user to compare events according to a subset of their coordinates, and discover clusters of events. Also, the user can dynamically change the compared coordinates and see if the distribution of clusters is preserved under different points of view. The work ended with the production of a prototypical tool, RoutingWatch, which was evaluated by users with an expertise in networking. A preliminary version of this chapter was published in [CDD+16].

6.1 Introduction

Chapter 5 introduced *routing events*, namely high-level descriptions of significant network path changes. Also, an inference algorithm was described for discovering events from a large set of traceroutes collected by a probe system. This approach has several interesting characteristics and might become a valid complement of other classical tools for network monitoring, since: i) it exploits traceroutes performed by large networks of probes whose diffusion is rapidly growing, ii) to a certain extent, it enables investigation of events occurring in networks that are not under the direct control of

the network operator, easing responsibility assignment of incidents, and iii) it does not require special agents on network devices. However, many routing events can be ascribed to non-anomalous behaviour or to non-significant incidents, hence a tool to support further analysis of inferred routing events would be beneficial.

In this chapter we describe RoutingWatch, a visual tool to explore and analyze a large set of routing events. RoutingWatch aims at complementing existing monitoring and management tools, and has the following characteristics: i) it is conceived to display routing events, as opposed to low-level metrics (bandwidth, delay, etc.); ii) it provides the user with a combined view of spatial, temporal, and abstract attributes of the events; iii) it offers several selection and filtering tools that realize a continuous user interaction (as opposed to a query followed by a visualization), thus supporting the investigation of incidents based limited information; iv) it displays relationships between events based on a customizable notion of similarity, thus enabling quick recognition of patterns (e.g., similar impact on customers or devices, or recurring events). The target users of RoutingWatch are high-level administrators and technicians operating inside a Network Operations Center (NOC) of an ISP, who have a broad view of the network and would benefit from looking at highly informative aggregate reports rather than raw measurements. We assume that their main activities involve checking that a network is operating correctly and carrying out troubleshooting tasks for hard problems that a customer-facing help desk could not directly solve. Interactions with network experts helped us design RoutingWatch in such a way to effectively support at least the above activities. We also conducted a preliminary user study that confirmed this effectiveness.

The rest of this chapter is organized as follows. In Section 6.2 we describe our reference scenario in terms of input data, activities, and tasks. In Section 6.3 we review other visual network analysis techniques and tools. In Section 6.4 we describe the architecture of RoutingWatch, including the technique to infer routing events, as well as its user interface. In Section 6.5 we evaluate the effectiveness of the tool by illustrating use cases and by presenting the outcome of a preliminary user study. In Section 6.6 we draw conclusions and directions for future work.

6.2 Reference Scenario

We believe that a NOC high-level technician is more inclined to look at events, namely few high-level and informative concepts, rather than at the unmanageable amount of routing paths (or other information) they derive from. As a side benefit, inferring events from traceroute paths enables visibility of changes that span arbitrary geographical areas, thus allowing an operator to determine that the responsibility of a

fault is probably to be sought for in a third-party network.

RoutingWatch takes inferred routing events as input, and provides operators with a visual interface designed to support at least the following activities:
Monitoring – Checking the normal operational status of a network and ensuring that no unexpected events occurred.
Verification – Assessing the impact of applied changes and checking that planned variations to the topology or the configuration (e.g., due to infrastructure upgrades or network engineering) had the desired outcome.
Troubleshooting – Investigating a user-reported issue, assuming that the report includes a time and the customer's geographic location, besides a description of the experienced issue.

In order to better outline the application context, we describe the main kinds of interaction that we expect a user should have with RoutingWatch, in the form of tasks:
T1: *Circumscribe the set of events* – For all the considered activities, the user needs to isolate a subset of the routing events that is most relevant for his investigation. This task is more difficult to support when only approximate coordinates (e.g., time, geography) to look at are known (like it may happen during troubleshooting sessions).
T2: *Compare event distributions* – The user should be able to select some events based on one dimension (e.g., geography) and check how the selected events are distributed in other dimensions (e.g., time). This is mostly involved in verification and troubleshooting activities.
T3: *Find similar events* – The user may want to look for sets of related events (e.g., close or recurring in time, or with the same causes), especially during troubleshooting sessions.

6.3 Related Work

Detecting and analyzing network events is a perpetual need for network operators, and several methodologies and tools have been introduced to aid in accomplishing these tasks. We now review the main contributions related with the exploration of routing events.

First of all, events need to be singled out. Network management and monitoring tools such as [MG-15,Cis15,Nag15] usually raise event notifications based on unexpected changes in network performance metrics (e.g., bandwidth), device health flags (e.g., interface status), or configuration contents (e.g., parameter changes via the CLI). Therefore, they require polling devices or running agents on top of them, and they must be explicitly designed to support a wide range of technologies. On the other hand, event inference methodologies, such as [CGC13,CC14], exploit more so-

phisticated techniques to extract events from massive sets of publicly available data (e.g., routing tables), without the need to access devices. However, they are designed for long-term observations of the evolution of the Internet and they provide little information about each specific event. This gap is filled by other methodologies, such as [KYGS,DTDD07,DDP+15b], which extract high-level descriptions of routing events starting from simple low-level measurements (e.g., traceroutes). However, since they are also applied at Internet scale, the number of events that they infer is often too high to be handled by a network operator without the support of an additional tool.

To ease the exploration of large sets of detected routing events, several visualization systems have been introduced. A selection of visualization techniques is proposed in [TMW03] to aid in finding outliers and recurring patterns of interdomain routing changes. However, the limited integration between the various proposed visualizations can make the identification of events difficult. Some tools, such as [CDM+05b,LMZ06,CDDS13,COZ08,Tho], enable the exploration of dynamic aspects of network routing using animations and graph transformations, with the goal of helping the user in finding and investigating anomalous behaviors. However, the notion of event is rarely formalized in these contributions, and the input of these tools consists of raw routing and performance data. Conversely, RoutingWatch is specifically designed to display highly informative and soundly defined events that are inferred from such data. To the best of our knowledge, it is the first tool that supports a visual exploration of such information, and in doing so it also quite meets the research directions stated in [CGC13]: a combined investigation of space and time, an assessment of the type of routing change, and the application of event inference methodologies to router-level topologies.

6.4 RoutingWatch: A Visual Event Analysis Tool

In this section we describe RoutingWatch, a visual exploration and analysis tool that allows an operator to investigate a large number of routing events. RoutingWatch is meant to integrate other visual tools, like Radian (see Chapter 3), and network management systems, which are more focused on digging into the details and finding the root cause of single events. The design of RoutingWatch was devised by keeping in mind the scenario, activities, and tasks in Section 6.2, as well some requirements gathered by interaction with network operators (see Section 6.5).

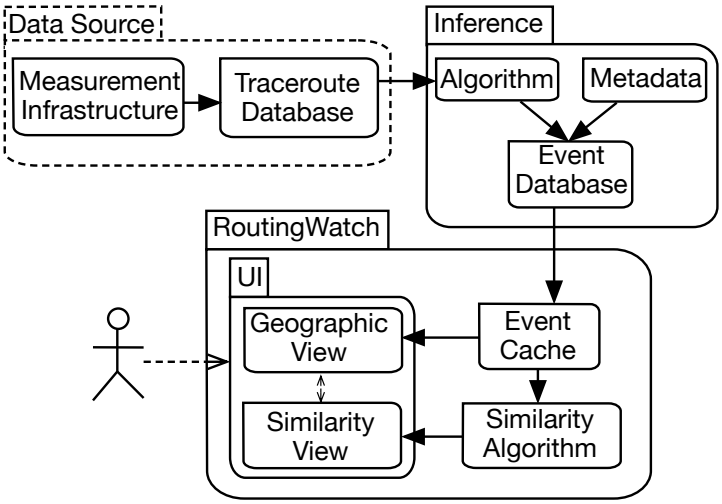


Figure 6.1: Architecture at the basis of RoutingWatch, showing the data processing steps. The dashed box surrounds steps that are accomplished by a third-party organization.

Architecture

The complete architecture at the basis of RoutingWatch is depicted in Fig. 6.1. It consists of three main components, which are meant to be distributed on different machines: a Data Source module, which is used to gather the traceroute paths collected from the probes, an Inference module, which extracts events from the paths, and RoutingWatch itself, which embodies the algorithms and user interface that make up our tool. We assume that the Data Source module is implemented by an organization that runs a distributed measurement infrastructure. The Inference module exploits the technique for inferring events described in Chapter 5. A large part of this chapter focus on the design, implementation, and evaluation of the RoutingWatch module which is our main novel contribution.

In this architecture, data are processed along a pipeline that is outlined by the arrows in Fig. 6.1. First of all, traceroute paths are collected by the probes in a distributed measurement infrastructure, such as [atl15] or [sam15], and stored in a database to support later batch retrieval.

Then, we run the inference algorithm on the traceroute paths in order to extract routing events, which in turn we store in an event database. Detecting events is com-

putationally expensive. Therefore, in our system the inference process is scheduled to run periodically, for example once per day, on data chunks consisting of the latest unprocessed traceroutes. As explained in Section 6.2, events are labeled with additional metadata that will be used by RoutingWatch: the geographic location of probes (administratively entered) and a mapping from IP addresses to the ASes they belong to (obtained from [Max15]), which is applied to the public IP addresses of probes and to the reported causes of each event.

Finally, RoutingWatch is a JavaScript-based tool that runs in a Web browser and visualizes the inferred events. When invoked by the user, the tool fetches data from the event database in a user-specified time window and, possibly, relative to a specific set of probes. To ensure prompt interaction, the resulting set of events is temporarily stored in the user browser's local memory (*event cache*). Moreover, to support task T3, the tool executes an algorithm to compute similarities among events. The visual interface, as well as the method to compute the similarity, are illustrated in the following subsections.

Overview of the User Interface

One of the first challenges in designing RoutingWatch was to integrate in a single dashboard the intuitive *spatio-temporal attributes* of events (geography and time) and their *abstract attributes* (cause IPs, targets, set of probes, Autonomous Systems, etc.), which are harder to interpret and convey visually. We opted for two coordinated views [NS00,Rob07] put side by side, which we call *space-time view* and *similarity view*. The latter is meant to support Task T3. In accordance with the usability principles in [Rob07], these views support continuous interaction between the user and the tool, as opposed to an approach where a query is followed by a visualization. Figure 6.2 shows an overview of the user interface of the tool, where these two views are prominently visible (❶ and ❷).

Both views display the same set of events, called *current event set*. This set can easily be altered by specifying general matching criteria using visual controls in a filtering panel (❸): they restrict events in the current set to those with a minimum impact, recorded by at least one probe in a specified AS ("Impacted AS"), affecting the reachability of at least a specified target, and whose causes comprise at least an IP address in a specified AS. These filtering actions support Task T1. Finally, an additional similarity panel (❹) can be used to tune the computation of similarities among events, influencing the associated view (❷).

The effect of any user interactions is immediately applied to both views, thus realizing the data analysis approach known as *brushing-and-linking* (see, e.g., [NS00]).

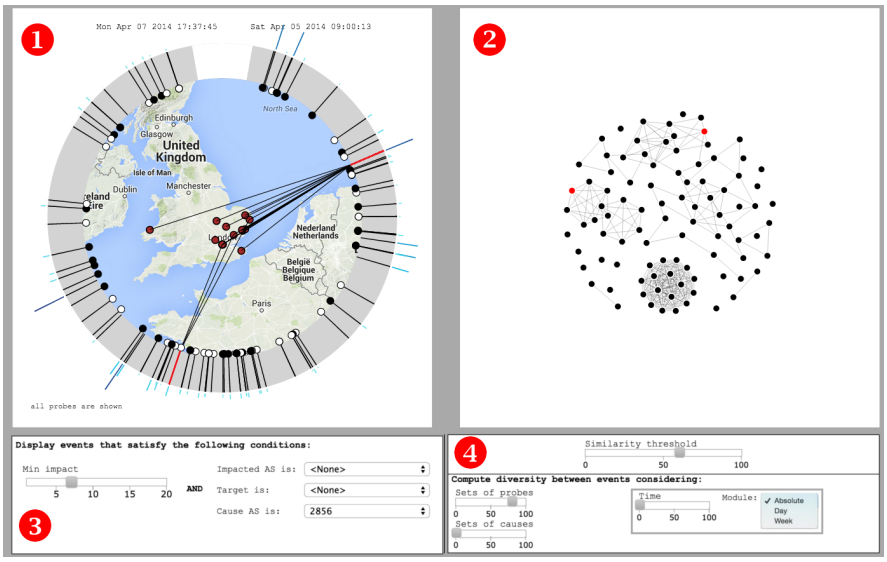


Figure 6.2: User interface of RoutingWatch, with its main components: 1 space-time view; 2 similarity view; 3 filtering panel; 4 similarity panel.

Space-Time View of the Events

The space-time view (Figure 6.3) consists of a geographic map (obtained from [Goo15]), that displays the location of the probes (1) involved in the current event set[1], surrounded by an annulus (2) that represents the extent of the time window of the observed events (also shown in 3). The user can zoom and pan the map enclosed in the annulus using standard interactions. A message (4) warns the user if any probes are off-screen. Time advances clockwise in the annulus, similarly to a watch. Each event in the current set is represented by a black marker in the annulus placed at its time of occurrence (which is displayed in a tooltip 5); its impact is shown by the length and color of a bar placed outside the annulus (6); its type is represented by a circle at the base of the marker, black-colored for down events and white-colored for up events (lack of the circle indicates unknown type). To support Task T1, the time window of the current event set can be dynamically changed by "zooming" in the annulus using the mouse wheel. Clicking on a marker adds or removes an event from the *current*

[1]The location of the targets may be impossible to establish (for example for anycast IP addresses), therefore they are not displayed on the map.

selection, causing it to be selected or deselected in the similarity view as well. This allows the user to focus on a set of events of interest, which remains unchanged even when the current event set is altered (for example by filters). Clicking on a probe (de)selects all the events recorded by that probe.

Each event in the current event set which is also in the current selection is highlighted in red in both views (see, e.g., Figure 6.2), so that events displayed in the two views can be easily matched. Moreover, such an event is also linked with lines (❼ in Figure 6.3) called *leaders* [Tam07] to all the probes that recorded it. This makes it also clear why we chose the annulus shape: on one hand it allows us to display wide time intervals without consuming too much space on the screen, while on the other hand it makes events more or less equally close to probes, making the leaders more readable.

Hovering with the mouse cursor on an event temporarily highlights it in orange both in the space-time view and in the similarity view, and reveals its leaders and timestamp. Similarly, hovering on a probe temporarily reveals leaders connecting it to all the events it recorded. This enables a user to quickly explore events, for example to check the geographic area affected by the current event set.

Event Similarity View

Discussions with network experts underlined the importance of Tasks T2 and T3, therefore we chose to support them with an additional view. The similarity view (Figure 6.4) focuses on the relationships between events. It consists of a graph, whose nodes represent elements in the current event set and whose edges connect events with similar attributes. To ensure an optimal layout, the drawing is produced by the force-directed algorithm implemented in the D3.js library [BOH11]. In order to support diverse analytical purposes, the notion of *similarity* can be customized by the user: different event attributes can contribute in user-configurable proportions to the computation of the similarity between two events e_1 and e_2. In particular, it is possible to consider the following normalized measures:

- the similarity S_p between the sets of probes that recorded the events, computed using the Jaccard similarity coefficient: $S_p = \frac{|probes(e_1) \cap probes(e_2)|}{|probes(e_1) \cup probes(e_2)|}$
- the similarity S_c between the sets of causes, again computed as Jaccard similarity: $S_c = \frac{|causes(e_1) \cap causes(e_2)|}{|causes(e_1) \cup causes(e_2)|}$
- the proximity S_t in time between e_1 and e_2, which is parametrized by a module M. When $M = \mathsf{absolute}$, the proximity is computed based on the time distance

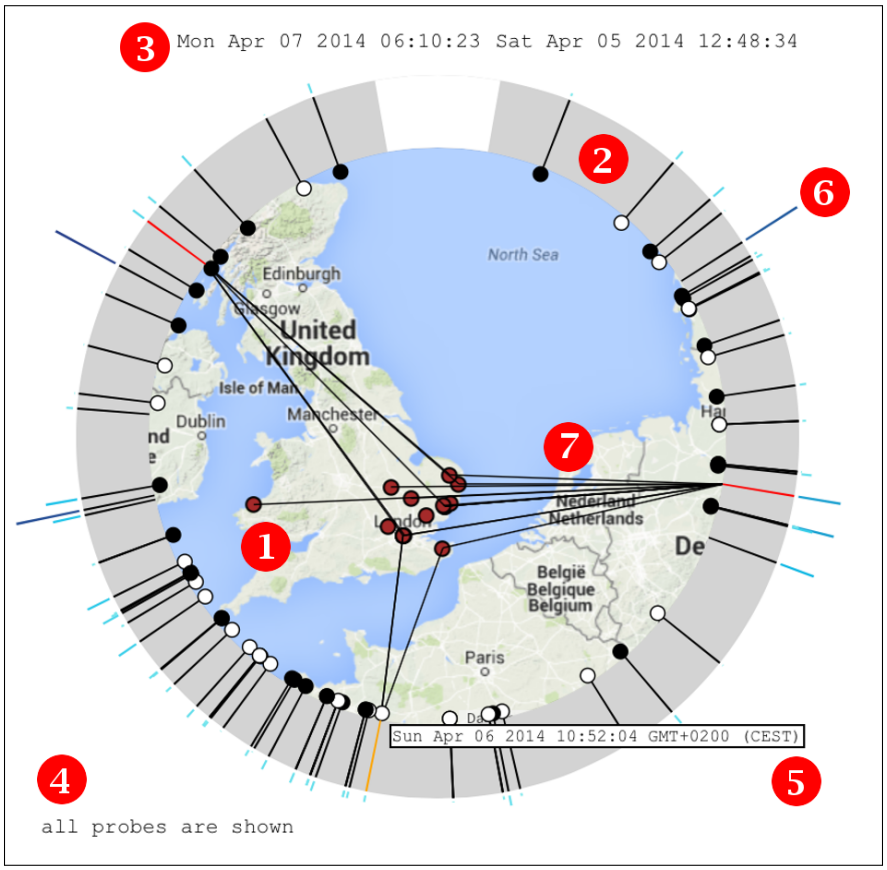


Figure 6.3: Detail of the space-time view of RoutingWatch, with its main elements: ❶ probes; ❷ annulus with event markers; ❸ time window of the current event set; ❹ out-of-screen warning message; ❺ event timestamp tooltip; ❻ event impact; ❼ leaders connecting events to probes.

between e_1 and e_2, relative to the time window T of the current event set: $S_t = 1 - \text{abs}(t(e_1) - t(e_2))/T$.
When $M \in \{1\ \text{day}, 1\ \text{week}\}$, the proximity considers the time distance modulo M, so that $S_t(M)$ is maximized when this time distance is equal to kM ($k \in \mathbb{N}$) and minimized when it is equal to $(k+1/2)M$:

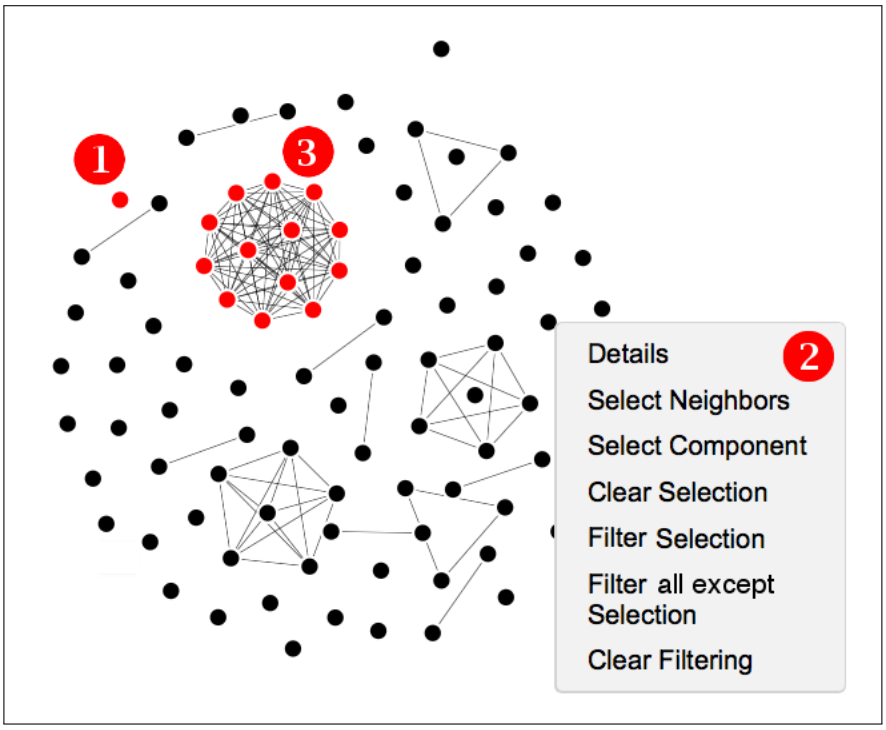


Figure 6.4: Detail of the similarity view of RoutingWatch, with its main elements: 1 selected event; 2 event context menu; 3 selected connected component.

$$S_t(M) = 1 - \frac{\min((t(e_1)-t(e_2)) \bmod M, (t(e_2)-t(e_1)) \bmod M)}{M/2}$$

The overall similarity between two events e_1 and e_2 is computed as a normalized linear combination of the above similarity measures: $S = (c_p \cdot S_p + c_c \cdot S_c + c_t \cdot S_t(M)) / \sum c_i$. The coefficients c_p, c_c, c_t, as well as the modulus M, can be tuned by using the controls in the similarity panel (4 in Figure 6.2). In order to help the user in better isolating related events, an additional control ("Similarity threshold") establishes the minimum value of the similarity S for which an edge between two events is displayed. Changes in the position of all the controls in the similarity panel are immediately applied to the similarity view.

With the goal of realizing Tasks T2 and T3, the similarity view supports various kinds of interaction. Clicking on an event adds or removes it from the current selection

(1). Each event has a context menu (2) with several functions:

- Displaying the full details of the picked event (IP addresses and AS numbers of probes and causes, as well as the IP addresses of the targets). These details may include pre-cooked links that invoke external analysis tools for an in-depth investigation of the set of involved traceroutes (in the current implementation we generate links to [CDDS13]).
- Selection of a group of similar events, implemented by two functions: selection of all the events that are similar to the picked one (namely its neighbors in the similarity graph), or selection of events that are related with the picked one even indirectly (namely those that are in the same connected component as the picked event). The latter operation can be especially effective when searching for recurring events, because it extends the selection even to those that are not perfectly aligned in time. For example, in Figure 6.5 the similarity has been tuned to only relate events occurring every 24 hours ($c_p = c_c = 0$, $c_t = 1$, $M =$ 1 day): selecting an entire connected component reveals groups of events that happen roughly in the same time slot of the day, which are visible in the annulus as clusters of red markers. A user can then verify, for example, whether these events have impacted a specific geographic area (set of probes), confirming that they are recurrent and deserve further investigation.
- Using the current selection (or its complement) as a filter, allowing the user to define a custom event set (Task T1).

6.5 Evaluation

In this section we first illustrate some use cases that show how RoutingWatch can support the activities described in Section 6.2. In all the use cases, we assume that the users are technical operators in a NOC, who can access and explore event reports and, if required, dig into the details of specific routing changes. After that, we describe the organization and outcome of a preliminary user study that we conducted to gather feedback on the tool from network operators.

Activity 1: Monitoring

During a monitoring session, Alice wants to verify whether a set of routing changes occurred overnight are anomalous when compared with those happened in the last week.

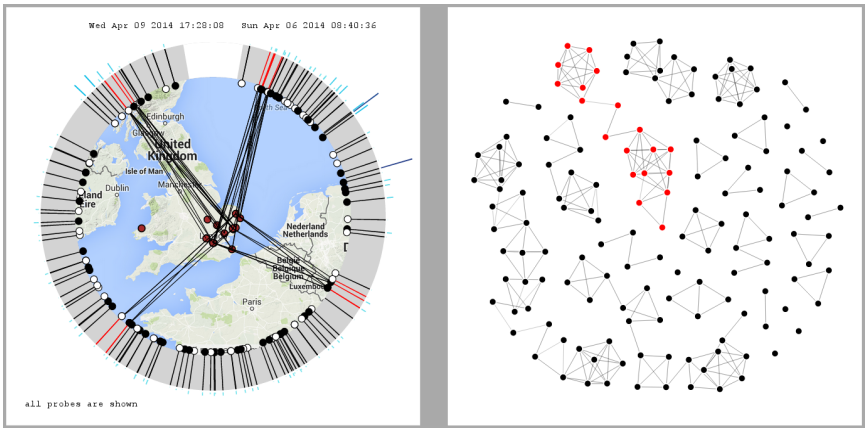


Figure 6.5: Selection of a connected component in the similarity graph, where the overall similarity is tuned to only consider the time distance modulo 24 hours.

(*i*) Alice launches RoutingWatch and loads the events spanning the last week. She selects the events that occurred during the last hours and watches them in the similarity view.

(*ii*) By tuning the sliders in the similarity panel, Alice sets up the similarity to only consider the event causes. She then observes if some of the selected events form a cluster of their own, showing that they have been attributed to causes that were not common in last week's events.

(*iii*) Analogously, by tuning the similarity to solely consider the sets of probes, Alice assesses whether events occurred last night are unusual with respect to the affected probes.

(*iv*) In both cases, by using the slider in the filtering panel, Alice can isolate events with a major impact. If any such events occurred, she can further investigate their nature by displaying their details in separate browser tabs.

Activity 2: Troubleshooting

Troubleshooting is the most challenging activity among the ones listed in Section 6.2. For this example we invite the reader to consider Figure 6.6 as a reference. We assume that the help desk of an ISP is collecting complaints from many customers in a limited

geographic area, reporting slow connection to all Web sites. The help desk passes this information to Bob, a technical operator in the ISP's NOC.

(*i*) Bob launches RoutingWatch, loading a set of events that span a time window starting slightly before the time indicated by the help desk and ending at the current time. Lots of events are displayed on the annulus (see Figure 6.6a), therefore in the next steps Bob pans the map to focus on the geographic area indicated by the help desk.

(*ii*) By moving the sliders in the similarity panel, Bob tunes the similarity to mostly consider the sets of probes, and by adjusting the similarity threshold he sees that there are groups of events forming clusters in the similarity view (A, B, C, D, E in Figure 6.6b).

(*iii*) Bob selects, one after the other, every cluster in the similarity view, using the "select connected component" function in the context menu (Figure 6.6b shows a selected cluster). By doing so, he discovers that the probes associated with all the events are located in the geographic region he is currently observing, with the exception of events in cluster D (see Figure 6.6c, where most leaders point to off-screen probes).

(*iv*) Bob selects events in cluster D and filters them out. The result is in Figure 6.6d.

(*v*) Bob raises the minimum impact threshold of displayed events, resulting in a much reduced set of events to look at (see Figure 6.6e).

(*vi*) By selecting one of the connected components in the similarity view, Bob verifies the spatial and temporal attributes of the corresponding events (see Figure 6.6f).

(*vii*) Bob increases the weight of event causes in the computation of the similarity, to understand how the clusters formed based on the sets of probes are related with those causes. In addition, he raises the similarity threshold in order to hide edges that represent poorly related events. Because of this, clusters A and B split up into two parts each: A1, A2, B1, and B2 (see Figure 6.6g). The resulting clusters can easily be matched with the previous A and B by looking at the current selection. Each cluster is made of events that have a similar cause and have been seen by a common set of probes.

(*viii*) By looking at some event details, Bob discovers that all the events in each cluster had a single cause. He therefore identifies 5 IP addresses (one per cluster) that are potential causes of the problems reported by the help desk.

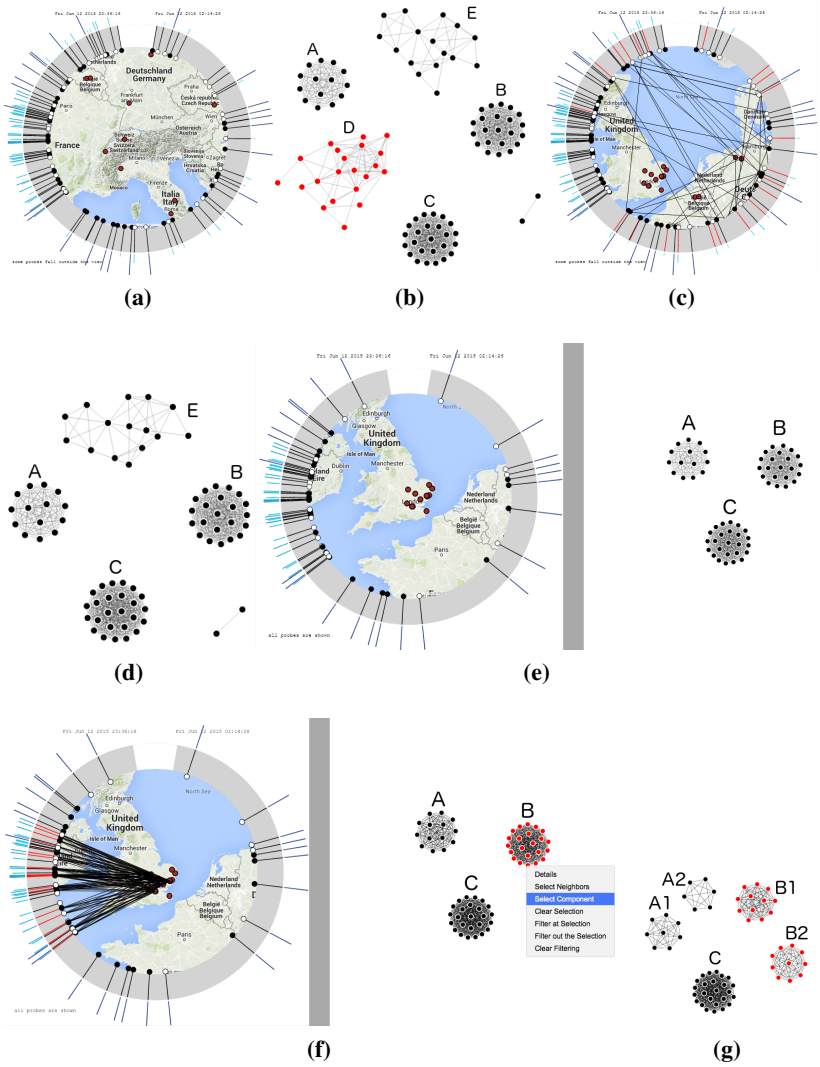


Figure 6.6: Event analysis steps during a troubleshooting session accomplished using RoutingWatch. The letters close to each cluster in the similarity graph are annotations inserted for clarity, and are not rendered by the tool.

(*ix*) Hence, Bob can deepen his analysis on these IP addresses, for example by checking in the logs of the associated devices whether they underwent reboots for planned maintenance. Bob can then ask for an inspection of the devices to assess whether misbehaviors of their network cards could have led to frequent routing switching, a potential cause of the user-reported poor quality of experience.

After post-maintenance tests are passed, Bob can carry out a verification activity to check that the correct operation of the devices has been restored: he selects cluster C, filters out all the other events, and verifies on the annulus that the sequence of events ends at the time when the faulty cards were repaired.

User Study

We conducted an informal user study during an intermediate stage of the development of our system, in order to check whether it fitted the requirements of prospective users and to gather ideas for improving its features. In the context of a research project that was active at the time of the user study, we interviewed 6 selected employees of the R&D section of a prominent Italian ISP. Their areas of expertise covered IP edge innovation, cyber-security threat evolution, security solutions analysis, and video & multimedia platforms, making them familiar with the actual needs and issues of the ISP.

The interview lasted two hours and half and was split into three parts: 1) a presentation of the tool, to explain the motivations, the input data, and the functionalities; 2) a supervised session of usage of the tool; 3) a questionnaire that the experts were asked to fill out with their opinions about the motivations of our study and the effectiveness of the tool. During the supervised usage session, the experts could use RoutingWatch with sample data and receive help from us. They were supposed to hand in the filled questionnaires at the end of the interview, but the presentation and the usage session took more time than expected, and a few of them did not make it in time. For this reason we decided to let all the participants send us the results the day after, by email. During this time we left the prototype remotely accessible.

In the questionnaire the experts were asked to rank between 1 (min) and 5 (max) whether (min/avg/max assigned ranks in parentheses): a) events are a useful and comprehensible aggregation of routing changes (3/3.83/5); b) comparing events by probes, geography, and time is useful to find related events (3/3.67/4) and how good the tool is at supporting it (2/3.33/5); c) finding related events is useful for the activities of a network operator (3/3.83/4) and how good the tool is at supporting it (3/3.67/4).

User comments in the questionnaire confirmed the validity of our motivations, because measurement infrastructures collect massive amounts of traceroute data. The abstraction of considering events and the visual interface were appreciated. Somebody observed that some devices may not be revealed by a traceroute, for example because a device is explicitly configured not to respond to measurements or because of the encapsulation of packets into tunnels. However, they all agreed that for an ISP traceroute paths are one of the few available sources of information about external networks.

Since the first questions they posed, the experts defined RoutingWatch as "a tool for mining traceroute data", which we considered an appropriate description. The geographic map, the circular displacement of events, and the usage of leaders were all considered effective. Moreover, the users considered very important to have a set of readily usable filters (e.g., by target or time of the day) to avoid seeing "too much data" to be understood at a glance. The possibility to easily spot patterns of events and similarities between them was very well evaluated. For example, when the node causing a problem is known in advance, the user might want to know if it had already caused other problems in the past. Also, they were interested in finding recurring events, typically happening on the same day of the week or at the same time of the day (e.g., the number of connected home users is much higher in the evening, potentially causing load issues). The concept of similarity was missing from the version of the tool we demonstrated to the experts. However, their feedback confirmed that events can give additional fundamental information when they are compared to each other, justifying the introduction of the similarity and its explicit visual representation.

6.6 Conclusions and Future Work

We presented RoutingWatch, a visual tool that allows Internet Service Providers to analyze routing events using easy-to-understand controls and a continuous interaction model. The tool encompasses a notion of *similarity* among events, which supports advanced use cases such as searching for recurring events. We implemented the tool showing that its interaction model can be supported by currently available Web browsers and we presented it to a few network operators obtaining good feedback. We believe the approach pursued in the design of the visual interface is general enough to support even other (i.e., not network-specific) notions of events, for example related with security-sensitive contexts.

Further work is required: at the moment we do not allow the user to "reproduce" a previously accomplished analysis session and we do not support any kind of session history (e.g., to undo filters). Such operations would allow the user to share the results

of problem investigations with other network experts, enabling the development of a knowledge-based decision support system for tasks like troubleshooting. We still need to solve some scalability issues affecting the user interface. For example, an excessive number of displayed leaders can be confusing and can be avoided by dynamically clustering probes, events, and the leaders themselves when they are too close to each other. We also plan to extend the functionalities of the tool, for example by supporting more flexible notions of similarity or real-time analysis (at the moment, data can only be fetched from static data sources). This would obviously require redesigning the back-end infrastructure.

Part III

Interplay Between Routing and Geography

Chapter 7

Planarity of Georeferenced Graphs

This chapter describes an approach based on *Topology + Geography* (see Chapter 1) for drawing graphs whose vertices have a geographical position, or *georeferenced graphs*, which is the case of routing graphs. The approach deals with the *Relation To Geography* challenge. The work is based on the intuition that vertices can be moved within a limited span around their given positions to improve the readability of the drawing. According to this intuition, a graph planarity problem is introduced and studied under different distance metrics and drawing styles. A polynomial-time algorithm is presented for one of the studied cases, while the others resulted NP-hard, thus revealing an intrinsic difficulty to implement the intuition on georeferenced graphs with a planarity-based approach. A preliminary version of this chapter was published in [ADD+14].

7.1 Introduction

Several applications require drawing graphs whose vertices are constrained to be not too much *distant* from specific points [AAHS05,LMR98]. As an example, consider a graph whose vertices are cities and whose edges are relationships between cities. It is conceivable that the user wants to draw the graph on a geographic map where vertices have the coordinates of the corresponding cities. Unfortunately, depending on the local density of the cities, the drawing may be cluttered or may contain crossings between edges that might disappear if the vertices could move from their locations. Hence, the user may be interested to trade precision for quality of the drawing, accepting that the vertices move of a certain distance from the location of the cities, provided that the readability of the drawing increases. Problems in which the input

consists of a set of imprecise points have also been studied in Computational Geometry [DM03,LvK10].

In this chapter we consider the following problem, that we call ANCHORED GRAPH DRAWING (AGD)[1]. Given a graph $G = (V,E)$, an initial placement for its vertices, and a distance δ, we ask whether there exists a planar drawing of G, according to a certain drawing convention, such that each vertex $v \in V$ can move at distance at most δ from its initial placement. Note that the problem can have different formulations depending on how the concepts of "readability" and "distance" are defined.

We consider both straight-line planar drawings and rectilinear planar drawings. Further, in addition to the traditional L_2 Euclidean distance, we consider the L_1 Manhattan distance and the L_∞ 'uniform' distance. Note that, adopting L_2 distance is equivalent to allowing vertices to be placed into circular regions centered at their original positions, and adopting L_1 or L_∞ distances is equivalent to allowing vertices to be placed into diamond-shaped or square-shaped areas, respectively.

Observe that, if the regions of two vertices overlap, the positions of the two vertices can be swapped with respect to their initial placement, which may be confusing to a user of the drawing. Moreover, overlapping between vertex regions would make problem AGD as difficult as known Clustered Planarity variants, such as the Strip Planarity problem [ADDF13] in the straight-line setting, whose complexity is a non-trivial open problem. Hence, we restrict to instances such that the regions of the vertices do not overlap.

We remark that the version of the problem where each circle may have a different size was shown to be NP-hard in [God95] by reducing Planar-$(3,4)$-SAT with variable repetitions (where repeated occurrences of one variable in one clause are counted repeatedly). The proof in [God95] uses disks with radius zero and disks with large radii. Also, the reduction relies on overlapping disks.

Furthermore, we observe that the NP-hardness of the problem with different distances and overlapping areas trivially follows from the NP-hardness of extending a planar straight-line drawing [Pat06] by setting $\delta(v) = 0$ for each fixed vertex v and allowing suitably large distances for vertices that have to be planarly added to the drawing.

In this chapter we show that the ANCHORED GRAPH DRAWING problem is NP-hard for any combination of metrics and drawing standards that we considered, with the exception of rectilinear drawings and uniform distance metric (square-shaped regions). These results, summarized in Table 7.1, were somehow unexpected, as computing a planar rectilinear drawing of a graph, without any further constraint, is NP-hard [GT01].

[1] We remark that the term 'anchored graph' was used within a different setting in [CM13].

Metric	Distance	Region Shape	Straight-line	Rectilinear
L_1	Manhattan	$\diamond$	NP-hard	NP-hard
L_2	Euclidean	$\bigcirc$	NP-hard	NP-hard
L_∞	Uniform	$\square$	NP-hard	Polynomial

Table 7.1: The complexity of the ANCHORED GRAPH DRAWING problem depending on the metric and drawing style adopted when the areas of the vertices do not overlap.

The chapter is organized as follows. Section 7.2 contains basic definitions and terminology. Section 7.3 describes a polynomial-time algorithm when the considered distance is the uniform distance L_∞ and edges are required to be drawn as either horizontal or vertical segments. Section 7.4 is devoted to the NP-hardness proofs of all the other considered settings of the problems. Finally, Section 7.5 discusses some open problems.

7.2 Problem Definition and Instances Classification

A *straight-line planar drawing* of a graph G is a drawing of G where edges are straight-line segments that do not intersect except at common end-points. A *rectilinear planar drawing* is a straight-line planar drawing where edges are parallel to the axes.

Given two points p and q in the plane, denote by $dx(p,q)$ and $dy(p,q)$ the differences of their coordinates, i.e., $dx(p,q) = |x(p) - x(q)|$ and $dy(p,q) = |y(p) - y(q)|$, where $x(r)$ and $y(r)$ are the x- and y-coordinate of a point r, respectively. The *Euclidean distance* $d_2(p,q)$ of p and q is defined as $d_2(p,q) = (dx(p,q)^2 + dy(p,q)^2)^{\frac{1}{2}}$. The *Manhattan distance* is defined as $d_1(p,q) = dx(p,q) + dy(p,q)$. The *uniform distance* $d_\infty(p,q) = \lim_{i\to\infty}(dx(p,q)^i + dy(p,q)^i)^{\frac{1}{i}} = \max(dx(p,q), dy(p,q))$.

We define the ANCHORED GRAPH DRAWING problem parametrically in the metric L_k and the drawing style $\mathcal{X}$, which can be straight-line ($\mathcal{X} = \mathcal{S}$) or rectilinear ($\mathcal{X} = \mathcal{R}$). Hence, for any $L_k \in \{L_1, L_2, L_\infty\}$ and any $\mathcal{X} \in \{\mathcal{S}, \mathcal{R}\}$ we define: **Problem:** ANCHORED GRAPH DRAWING-L_k-$\mathcal{X}$ (AGD-L_k-$\mathcal{X}$). **Instance:** A graph $G = (V,E)$, an initial placement for its vertices $\alpha(v) : V \to \Re^2$, and a distance δ. **Question:** Does there exist a planar drawing of G according to the $\mathcal{X}$ drawing convention such that each vertex $v \in V$ is at distance L_k at most δ from $\alpha(v)$?

We define *anchored drawing* as a planar drawing satisfying all the requirements of the particular version of problem ANCHORED GRAPH DRAWING.

Given an instance $\langle G, \alpha, \delta \rangle$ of the ANCHORED GRAPH DRAWING problem, each vertex v identifies a region $R(v)$ of the plane, called *vertex region*, that encloses the

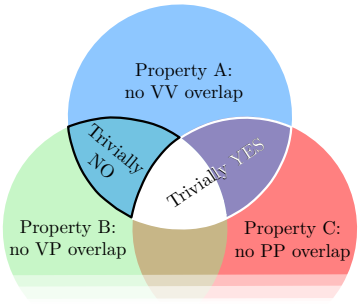


Figure 7.1: Venn diagram describing the logical relationships among Properties A–C.

initial position of the vertex and whose shape depends on the metric adopted for computing the distance. In particular, for the Euclidean distance the vertex regions are circles, for the Manhattan distance they are diamonds, and for the uniform distance they are squares. Each edge (u,v) of the graph, instead, identifies a *pipe* $P(u,v)$, defined as follows. Consider the convex hull H of $R(u)$ and $R(v)$; pipe $P(u,v)$ is the closed region obtained by removing $R(u)$ and $R(v)$ from H.

Instances can be classified based on the intersections among vertex and pipe regions. Namely, we can have instances satisfying the following properties:
Property A. No overlap between two vertex regions (VV-overlaps);
Property B. No overlap between a vertex region and a pipe (VP-overlaps);
Property C. No overlap between pipes (PP-overlaps) not incident to the same vertex.

The Venn diagram in Fig. 7.1 shows the logical relationships between the three properties. The following observation is immediate.

Observation 1. *If Properties A, B, and C are all satisfied, then the instance is trivially positive, since choosing any point in the vertex region (including the initial placement of the vertex) yields an anchored drawing of the input graph.*

We always assume that Property A is satisfied. In fact, if vertex regions were allowed to overlap, then it would be possible to reduce to this problem a variant of the Clustered Planarity problem whose complexity is still unknown. In this variant, which includes Strip Planarity [ADDF13] as a special case, the cluster regions are already drawn and edges are straight-line.

Two further observations can be made which reduce the set of instances of interest.

Observation 2. *An instance satisfying Property B but not satisfying Property C (i.e., with PP-overlaps but without VP-overlaps) is trivially false, as in this case any PP-*

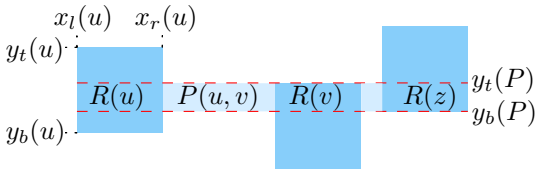


Figure 7.2: Geometric description of a region $R(u)$ and of a pipe $P(u,v)$, after procedure PIPEEQUALIZER has been applied.

overlap would enforce a crossing between two edges for any placement of their end-vertices in the corresponding vertex regions.

Observation 3. *An instance satisfying Property C but not satisfying Property B (i.e., with VP-overlaps but without PP-overlaps) is trivially true.*

Proof: Since Property C holds, no crossing can occur outside a vertex region. First, suppose that regions are diamonds or squares. If the center of region $R(v)$ of a vertex v lies inside a pipe $P(x,y)$, then at least two consecutive vertices, say a and b, delimiting $R(v)$ lie inside $P(x,y)$. This implies that v has degree at most 1, as otherwise there would be a *PP*-overlap between $P(x,y)$ and a pipe $P(v,w)$ delimited by either a or b.

As for the case in which regions are circles, if the center of $R(v)$ lies inside $P(x,y)$, then at least half of the circle delimiting $R(v)$ lies inside $P(x,y)$. Hence, a similar argument applies to prove that $\deg(v) \leq 1$.

In all the three cases, since $\deg(v) \leq 1$ and $R(v)$ is not completely contained into $P(x,y)$, v can be placed on any point of $R(v)$ outside $P(x,y)$. Hence, placing each other vertex at the center of its region yields an anchored drawing. □

Due to the above properties and observations, the remaining part of this chapter focuses on the instances for which Property A holds, while Properties B and C do not. These instances correspond to the blue region at the top of Fig. 7.1.

7.3 Polynomial-Time Algorithm

In this section we describe an algorithm, called **Algo-AGD-L_∞-$\mathcal{R}$**, that decides in polynomial time instances $\langle G, \alpha, \delta \rangle$ of problem AGD-L_∞-$\mathcal{R}$ such that G is connected.

For each vertex $v \in V$, denote by $x_l(v)$ and $x_r(v)$ the x-coordinate of the left and right side of $R(v)$, respectively. Similarly, denote by $y_t(v)$ and $y_b(v)$ the y-coordinate of the top and bottom side of $R(v)$, respectively. See region $R(u)$ in Fig. 7.2.

First note that, for each edge $(u,v) \in E$, the relative placement of $R(u)$ and $R(v)$ determines whether (u,v) has to be drawn as a vertical or a horizontal segment, or (u,v) cannot be drawn neither horizontal nor vertical with its endpoints lying inside their corresponding regions. In the latter case, instance I is negative. An edge that has to be drawn as a horizontal (vertical) segment is a *horizontal* (*vertical*) edge. In the following we assume w.l.o.g. that any horizontal edge (u,v) is such that $x_r(u) < x_l(v)$, while any vertical edge (u,v) is such that $y_t(u) < y_b(v)$. A path composed only of horizontal (vertical) edges is a *horizontal* (*vertical*) path. Given that each edge (u,v) can be categorized as either horizontal or vertical, we can label its pipe $P(u,v)$ as either `horizontal` or `vertical` accordingly. Also, we can determine the minimum and maximum y-coordinate (x-coordinate) that a horizontal (vertical) edge (u,v) can assume while placing both its endvertices inside their regions. In the following we describe pipe $P(u,v)$ by means of these coordinates, which are denoted by $y_b(P)$ and $y_t(P)$ ($x_l(P)$ and $x_r(P)$), respectively. See `horizontal` pipe $P(u,v)$ in Fig. 7.2.

Also note that, if a vertex v of degree 2 is incident to two horizontal (vertical) edges (u,v) and (v,z), then replacing v and its incident edges with a horizontal (vertical) edge (u,z) yields an equivalent instance. Hence, we assume that, if there exists a vertex of degree 2, then it is incident to both a horizontal and a vertical edge.

As a preliminary step of the algorithm, we initialize the geometric description of each pipe $P(u,v)$ as follows. If P is `vertical`, then set $x_r(P) = min(x_r(u), x_r(v))$ and $x_l(P) = max(x_l(u), x_l(v))$. If P is `horizontal`, then set $y_t(P) = min(y_t(u), y_t(v))$ and $y_b(P) = max(y_b(u), y_b(v))$. Here and in the following, whenever a vertex region $R(w)$ (a pipe $P(u,v)$) is modified by the algorithm, we assume the pipes incident to w (the regions $R(u)$ and $R(v)$) to be modified accordingly.

In order to ensure that `horizontal` (`vertical`) pipes whose edges belong to the same horizontal (vertical) path have the same geometric description, we refine the pipes by applying the following procedure, that we call PIPEEQUALIZER. As long as there exist two `vertical` pipes $P'(u,v)$ and $P''(v,w)$ incident to the same vertex v such that $x_l(P') \neq x_l(P'')$ or $x_r(P') \neq x_r(P'')$, set $x_l(P') = x_l(P'') = max(x_l(P'), x_l(P''))$ and $x_r(P') = x_r(P'') = min(x_r(P'), x_r(P''))$. Analogously, as long as there exist two `horizontal` pipes $P'(u,v)$ and $P''(v,w)$ incident to the same vertex v such that $y_b(P') \neq y_b(P'')$ or $y_t(P') \neq y_t(P'')$, set $y_b(P') = y_b(P'') = max(y_b(P'), y_b(P''))$ and $y_t(P') = y_t(P'') = min(y_t(P'), y_t(P''))$. See pipe $P(u,v)$ in Fig. 7.2 after the application of PIPEEQUALIZER.

We then perform the following procedure, that we call PIPECHECKER. It first checks whether there exists a pipe P such that $x_r(P) < x_l(P)$ or $y_t(P) < y_b(P)$. Then, it checks whether there exists a PP-overlap between two pipes $P(u,v)$ and $P(w,z)$ such that: (i) neither of $R(u)$ or $R(v)$ has a VP-overlap with $P(w,z)$; and (ii) neither of $R(w)$ or $R(z)$ has a VP-overlap with $P(u,v)$. If one of the two checks succeeds, then

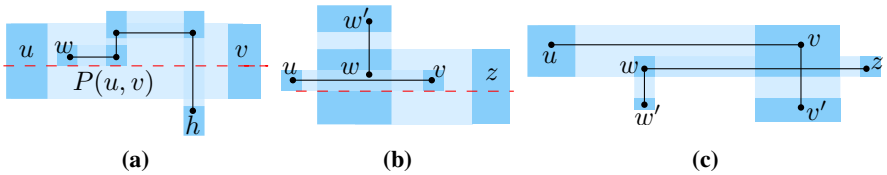


Figure 7.3: Vertices exiting pipes. (a) Vertex w exits $P(u,v)$ from below. The cut of $P(u,v)$ applied by procedure PIPEBLOCKCHECKER is described by a dashed line. (b) Vertex v exits $P(w,z)$ through w. The cut of $R(w)$ and the consequent cut of $P(w,z)$ applied by procedure VERTEXCHECKER is described by a dashed line. (c) A situation recognized by procedure PIPEINTERLEAVECHECKER.

we conclude that instance I is negative, otherwise we proceed with the algorithm.

In the following, every time a pipe is modified, we will apply procedure PIPEEQUALIZER to extend this modification to other pipes, and procedure PIPECHECKER to test whether such modifications resulted in uncovering a negative instance.

The general strategy of the main part of the algorithm is to progressively reduce the size of the pipes. In particular, at each step we consider the current instance I^i and modify it to obtain an instance I^{i+1} with smaller pipes than I^i that admits an anchored drawing if and only if I^i admits an anchored drawing. Eventually, such a process will lead either to an instance I^m for which it is easy to construct an anchored drawing or to conclude that instance $I = I^1$ is negative.

Let $P(u,v)$ be a `horizontal` pipe, and w be a vertex having a VP-overlap with $P(u,v)$. Refer to Fig. 7.3a. We say that *w exits P from below* if there exists a vertex h such that: (i) $y_b(P) < y_b(w) < y_t(P)$ and $x_r(u) < x_l(w) < x_r(w) < x_l(v)$; (ii) $y_t(h) < y_b(P)$ and $x_r(u) < x_l(h) < x_r(h) < x_l(v)$; and (iii) there exists a path $\gamma = (w,\dots,h)$ in G connecting w to h in which every internal vertex r is such that $R(r)$ intersects P. Symmetrically, we say that *w exits P from above* if there exists a vertex h with the same properties as before, except for the fact that $y_b(P) < y_t(w) < y_t(P)$, $y_b(h) > y_t(P)$. Otherwise, we say that *w exits P through a vertex*, either u or v. In Fig. 7.3b, vertex v exits pipe $P(w,z)$ through w. Observe that, since G is connected and no VV-overlap occurs in I, there always exists a path $\gamma = (w,\dots,h)$ in G connecting w to a vertex h such that h does not have any VP-overlap with P; hence, w always exits P, either from above or below, or through a vertex.

For the case of a `vertical` pipe $P(u,v)$, we assume analogous definitions of vertices exiting *P from left, right* or *through a vertex*, either u or v. As long as one of the following conditions is satisfied, we apply one of the procedures described hereunder.

Procedure VERTEXCHECKER: Consider a vertex w having a VP-overlap with a `horizontal` (`vertical`) pipe $P(u,v)$ such that $y_b(w) \leq y_b(P) < y_t(P) \leq y_t(w)$ (resp., $x_l(w) \leq x_l(P) < x_r(P) \leq x_r(w)$). If w is incident to two `vertical` (`horizontal`) pipes, then we conclude that instance I is negative. Otherwise, if w is incident to a `vertical` (`horizontal`) pipe $P(w,w')$, then set $y_b(w) = max(y_b(w), y_b(P))$ (set $x_l(w) = max(x_l(w), x_l(P))$. See Fig. 7.3b. Analogously, if w is incident to a `vertical` (`horizontal`) pipe $P(w',w)$, then set $y_t(w) = min(y_t(w), y_t(P)$ (set $x_r(w) = min(x_r(w), x_r(P))$.

Procedure PIPEBLOCKCHECKER: Consider a pipe $P(u,v)$ having a VP-overlap with a vertex w such that w does not exit through a vertex. If w exits $P(u,v)$ both from above and from below (a `vertical` pipe both from left and from right), then we conclude that instance I is negative. Otherwise, if w exits P from (i) above, we set $y_t(P) = y_t(w)$; (ii) below, we set $y_b(P) = y_b(w)$; (iii) left, we set $x_l(P) = x_l(w)$; or (iv) right, we set $x_r(P) = x_r(w)$. See Fig. 7.3a.

Procedure PIPESIDECHECKER: Consider a `horizontal` (`vertical`) pipe $P(u,v)$ and a vertex w exiting $P(u,v)$ both through vertex u and through vertex v. If u and v are incident to `vertical` (`horizontal`) pipes, either $P(u,u')$ and $P(v',v)$, or $P(u',u)$ and $P(v,v')$, respectively, then we conclude that instance I is negative.

Procedure PIPEINTERLEAVECHECKER: Suppose that there exist two `horizontal` (`vertical`) pipes $P(u,v)$ and $P(w,z)$ such that v and $P(w,z)$ have a VP-overlap, and w and $P(u,v)$ have a VP-overlap. If either v is incident to a `vertical` (`horizontal`) pipe $P(v,v')$ and w is incident to a `vertical` (`horizontal`) pipe $P(w,w')$, or v is incident to a `vertical` (`horizontal`) pipe $P(v',v)$ and w is incident to a `vertical` (`horizontal`) pipe $P(w',w)$, then we conclude that instance I is negative. See Fig. 7.3c.

If none of the above procedures can be applied, then we conclude that I is a positive instance.

Theorem 4. *Let $I = \langle G, \alpha, \delta \rangle$ be an instance of* AGD-L_∞-$\mathcal{R}$ *such that G is connected. Algorithm* **Algo-AGD-L_∞-$\mathcal{R}$** *decides in polynomial time whether $\langle G, \alpha, \delta \rangle$ admits an anchored drawing.*

Proof: The initialization of the pipes and their refinement operated by procedure PIPEEQUALIZER, both after the initialization and after each further modification, is trivially necessary to meet the requirements that vertices are placed inside their regions and edges are drawn as either horizontal or vertical segments.

Suppose that procedure PIPECHECKER concluded that instance I is negative at some point of the algorithm. If $x_r(P) < x_l(P)$ (if $y_t(P) < y_b(P)$), then there exist two `vertical` (`horizontal`) pipes sharing a vertex that cannot be placed inside its region while drawing both its incident edges as rectilinear segments. Otherwise, there exists a PP-overlap between two pipes $P(u,v)$ and $P(w,z)$ not overlapping with regions $R(u)$, $R(v)$, $R(w)$, and $R(z)$. By Observation 2, the instance is negative.

We prove that the modifications operated by VERTEXCHECKER, when a vertex w has a VP-overlap with a `horizontal` (`vertical`) pipe $P(u,v)$ and w is incident to a `vertical` (`horizontal`) $P(w,w')$, do not restrict the possibility of constructing an anchored drawing of $\langle G,\alpha,\delta\rangle$. Refer to Fig. 7.3b. In fact, in this case, in any anchored drawing of $\langle G,\alpha,\delta\rangle$, edge (w,w') cannot traverse $P(u,v)$ from top to bottom. As for the fact that an instance in which w is incident to two `vertical` (`horizontal`) pipes is correctly recognized as negative, observe that in this case one of the two vertical edges incident to w necessarily crosses edge (u,v).

We prove that the modifications operated by PIPEBLOCKCHECKER, when a vertex w overlaps a pipe $P(u,v)$ and does not exit through one of its vertices, do not restrict the possibility of constructing an anchored drawing of $\langle G,\alpha,\delta\rangle$. Suppose that w exits $P(u,v)$ from below, the other cases being analogous. Refer to Fig. 7.3a. The statement follows from the fact that, in any anchored drawing of $\langle G,\alpha,\delta\rangle$, the drawing of path $\gamma = (w,\ldots,h)$ blocks visibility from $R(u)$ to $R(v)$ inside $P(u,v)$ at least for all the y-coordinates in the range between the point where w is placed and the point where h is placed. Since $y_t(h) < y_b(P)$, the point where w is placed determines a new lower bound for the value of $y_b(P)$. Since such a point cannot be below $y_b(w)$, the statement follows. As for the fact that an instance containing a vertex w that exits a `horizontal` pipe both from above and from below (a `vertical` pipe both from left and from right) is correctly recognized as negative, observe that in this case the two paths starting from w completely block visibility between from $R(u)$ to $R(v)$ inside $P(u,v)$.

We prove that an instance containing a vertex w that exits $P(u,v)$ through both of its vertices, and such that u and v are also incident to pipes $P(u,u')$ and $P(v',v)$, or vice versa, reaching them from different sides, is correctly recognized as negative by procedure PIPESIDECHECKER. Namely, observe that in this case path (u',u,v,v') necessarily crosses one of the two paths starting from w.

Finally, suppose that procedure PIPEINTERLEAVECHECKER concluded that instance I is negative. Refer to Fig. 7.3c. It is easy to observe that the fact that v and w are reached from the same side is not compatible with an anchored drawing of $\langle G,\alpha,\delta\rangle$.

We conclude the proof of the theorem by showing that, when none of the described procedures can be applied, it is always possible to draw every edge (u,v) inside its pipe

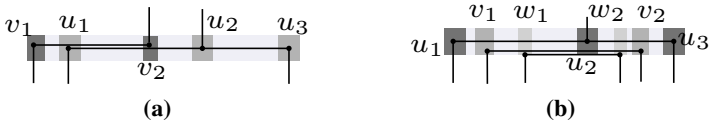


Figure 7.4: Construction of the drawing when none of the procedures can be applied. (a) Two maximal horizontal paths (u_1, u_2, u_3) and (v_1, v_2) whose pipes have the same y-coordinates. Path (v_1, v_2) is assigned a y-coordinate slightly larger than (u_1, u_2, u_3). (b) Three maximal horizontal paths (u_1, u_2, u_3), (v_1, v_2), and (w_1, w_2)whose pipes have the same y-coordinates. Path (v_1, v_2) is assigned a y-coordinate slightly larger than (w_1, w_2).

$P(u, v)$, as follows.

Consider every maximal horizontal path $(u_1, \ldots, u_r)$. Note that, each vertex u_i, with $1 \leq i \leq r$, is incident to at least a `vertical` pipe, either (u_i, u_i') or (u_i', u_i), as otherwise edges (u_{i-1}, u_i) and (u_i, u_{i+1}) would have been replaced with edge (u_{i-1}, u_{i+1}). If all the vertices u_i are incident to a `vertical` (u_i, u_i'), then assign y-coordinate equal to $y_t(u_i)$ to u_i, for $i = 1, \ldots, r$; if all the vertices u_i are incident to a `vertical` (u_i', u_i), then assign y-coordinate equal to $y_b(u_i)$ to u_i, for $i = 1, \ldots, r$; finally, if there exists at least a vertex u_i incident to a `vertical` (u_i, u_i') and at least a vertex u_j incident to a `vertical` (u_j', u_j), then assign y-coordinate equal to $\frac{y_b(u_i)+y_t(u_i)}{2}$ to u_i, for $i = 1, \ldots, r$. Assign x-coordinates to vertices of every maximal vertical path equal to $x_l(u_i)$, to $x_r(u_i)$, or to $\frac{x_l(u_i)+x_r(u_i)}{2}$, in an analogous way.

With a straightforward case analysis, it is possible to observe that, since none of the conditions activating the described procedures is satisfied, there exists no crossing in the drawing, apart from possible overlaps between edges belonging to different maximal horizontal (vertical) paths whose pipes have the same bottom and top y-coordinates (the same left and right x-coordinates). However, these overlaps can be always eliminated by increasing (decreasing) of an arbitrarily small amount the coordinates of the overlapping paths (see two examples in Fig. 7.4a and 7.4b), again due to the fact that none of the conditions activating the described procedures is satisfied. □

7.4 Hardness Results

In this section we prove the hardness of the ANCHORED GRAPH DRAWING problem in different settings. In particular, Theorem 5 is devoted to the hardness of the AGD-L_2-$\mathcal{S}$ problem, i.e., the problem of generating planar straight-line drawings of the input

graph where the vertex regions are circles of radius δ. Theorem 6, instead, is devoted to the hardness of the AGD-L_2-$\mathcal{R}$ problem, where the regions are circles of radius δ and edges are required to be drawn as horizontal or vertical segments.

The proofs of hardness for the remaining variants of the problem listed in Table 7.1 can be derived from these two and, thus, will not be explained in detail. Namely, the reduction to AGD-L_1-$\mathcal{R}$ is very similar to that used for AGD-L_2-$\mathcal{R}$, and can be obtained by suitably replacing circles with diamond-shaped regions that ensure analogous geometric visibility and obstruction properties. The same holds for the hardness proof of AGD-L_∞-$\mathcal{S}$, that can be obtained from AGD-L_2-$\mathcal{S}$ with small adaptations of the gadgets. Finally, the reduction to AGD-L_1-$\mathcal{S}$ is the same as the one for AGD-L_∞-$\mathcal{S}$ where all the geometric constructions are rotated by 45^o, transforming the square-shaped regions of AGD-L_∞-$\mathcal{S}$ into the diamond-shaped regions of AGD-L_1-$\mathcal{S}$.

All our proofs are based on a reduction from the NP-complete problem PLANAR 3-SATISFIABILITY [Lic82], defined as follows. **Problem:** PLANAR 3-SATISFIABILITY (P3SAT). **Instance:** A planar bipartite graph $G = (V_v, V_c, E)$ where: (i) V_v is a set of variables; (ii) V_c is a set of clauses, each consisting of exactly three literals representing variables in V_v; and (iii) E is a set of edges connecting each variable $v \in V_v$ to all the clauses containing a literal representing v. **Question:** Does there exist a truth assignment to the variables so that each clause has at least one `true` literal?

For each of our problems, we describe gadgets that, given an instance ϕ of P3SAT, can be combined to construct an instance γ of the considered problem. Namely, we describe a gadget for each of the following: *variable*, *not*, *turn*, *split*, and *clause*.

The variable gadget has two families of planar drawings, corresponding to the two truth values. The not gadget admits planar drawings that invert its input truth value. The turn gadget admits planar drawings that propagate its input truth value in a direction that is orthogonal to the original one. The split gadget admits planar drawings that propagate its input truth value to two different directions. Finally, the clause gadget is planar if and only if at least one of its input literals is `true`. The gadgets are combined following the structure of a planar drawing of ϕ, so that any planar drawing of γ corresponds to a truth assignment for the variables satisfying ϕ. Similarly, given a truth assignment for the variables that satisfies ϕ, the gadgets for variables can be drawn accordingly to obtain a planar drawing of γ.

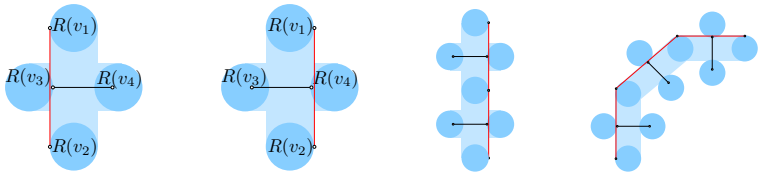


Figure 7.5: *Variable* gadget for the reduction to the AGD-L_2-$\mathcal{S}$ problem in its false (a) and true (b) configurations. (c) Propagation of the true configuration of a variable gadget. (d) *Turn* gadget in its false configuration.

Theorem 5. AGD-L_2-$\mathcal{S}$ *is NP-hard.*

Proof: To prove hardness we reduce problem P3SAT to AGD-L_2-$\mathcal{S}$, under the hypothesis that Property A is satisfied (no overlap among vertex regions).

Let ϕ be an instance of P3SAT with n variables and m clauses. We describe how to construct an equivalent instance γ of AGD-L_2-$\mathcal{S}$. For each variable x_i, $i = 1,\ldots,n$, we create a *variable* gadget, whose two families of planar drawings are depicted in Figs. 7.5a and 7.5b, consisting of four vertices v_1, v_2, v_3, and v_4 and edges (v_1, v_2) and (v_3, v_4). The regions assigned to the vertices are placed as follows: (i) the centers of $R(v_1)$ and of $R(v_2)$ lie on the same vertical line; (ii) the centers of $R(v_3)$ and of $R(v_4)$ lie on the same horizontal line; (iii) pipe $P(v_1, v_2)$ has an intersection of arbitrarily small area with both $R(v_3)$ and $R(v_4)$; and (iv) pipe $P(v_3, v_4)$ intersects neither $R(v_1)$ nor $R(v_2)$. Hence, in any anchored drawing of γ, edge (v_1, v_2) is drawn either to the left of v_3 (as in Fig. 7.5a) or to the right of v_4 (as in Fig. 7.5b). In both cases, edge (v_1, v_2) is drawn almost vertical. We call these two configurations *false* and *true* configurations for the variable gadget, respectively, and associate them with the `false` and `true` values for the corresponding variable x_i. The truth value of a variable can be propagated by concatenating a sequence of variable gadgets $\mu_1, \ldots, \mu_k$ in which $R(v_1)$ of μ_i is identified with $R(v_2)$ of μ_{i+1}, for each $i = 1, \ldots, k-1$. See Fig. 7.5c.

The *turn* gadget can be constructed by concatenating three variable gadgets, μ_1, μ_2 and μ_3, as depicted in Fig. 7.5d, in such a way that μ_2 has a clockwise rotation of 45° with respect to μ_1, and μ_3 has a clockwise rotation of 90° with respect to μ_1.

The *split* gadget can be constructed by combining two turn gadgets $\tau_L = \langle \mu_1^L, \mu_2^L, \mu_3^L \rangle$ and $\tau_R = \langle \mu_1^R, \mu_2^R, \mu_3^R \rangle$, with $\mu_1^L = \mu_1^R$ and where τ_L is obtained from τ_R by a vertical mirroring. See Fig. 7.6a.

The *not* gadget is constructed as follows. Consider two horizontally (vertically)

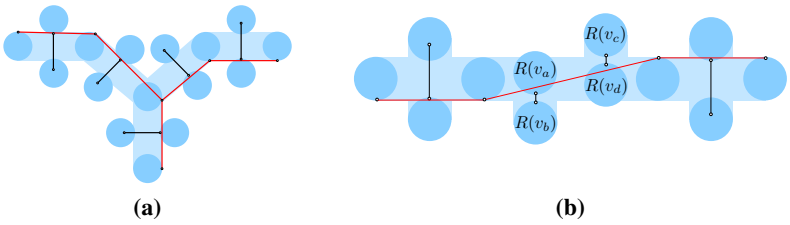


Figure 7.6: (a) *Split* gadget in its true configuration. (b) *Not* gadget.

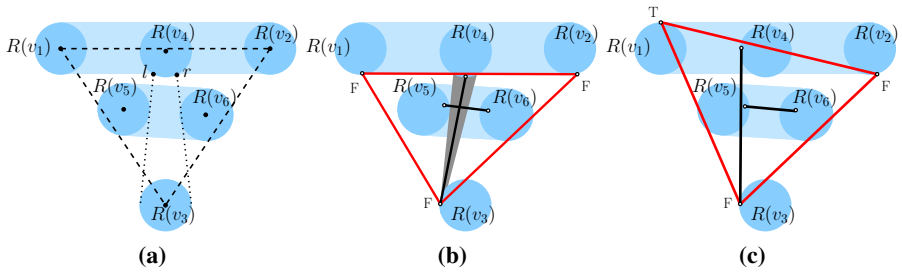


Figure 7.7: Clause for the reduction to the AGD-L_2-$\mathcal{S}$ problem. Vertices v_1, v_2, and v_3 represent the three literals of the clause. For readability, we show only pipes $P(v_1, v_2)$ and $P(v_5, v_6)$. (a) Arrangement of the regions. (b) All literals are assigned `false`, and edges (v_5, v_6) and (v_3, v_4) cross. The darker wedge represents all the possible positions for edge (v_3, v_4) in this truth assignment, which implies that the crossing is unavoidable. (c) Assigning `true` to any of the literals allows for a planar drawing.

aligned variable gadgets μ_1 and μ_2. Add an edge connecting v_2 of μ_1 to v_1 of μ_2, as in Fig. 7.6b. Place between μ_1 and μ_2 two pairs of adjacent vertices (v_a, v_b) and (v_c, v_d) whose regions are placed in such a way that: (i) any drawing of edge (v_a, v_b) blocks the visibility between the true configurations of μ_1 and μ_2; (ii) any drawing of edge (v_c, v_d) blocks the visibility between the false configurations of μ_1 and μ_2; and (iii) edges (v_a, v_b) and (v_c, v_d) can be drawn in such a way that there exists visibility between different truth configurations of μ_1 and μ_2. Hence, in any anchored drawing of γ, the configurations of μ_1 and μ_2 are different.

The *clause* gadget is constructed as follows. Refer to Fig. 7.7a. Consider three vertices v_1, v_2, and v_3 whose regions are placed in such a way that their centers induce a non-degenerated triangle $\mathcal{T}$ and the centers of $R(v_1)$ and of $R(v_2)$ lie on the same

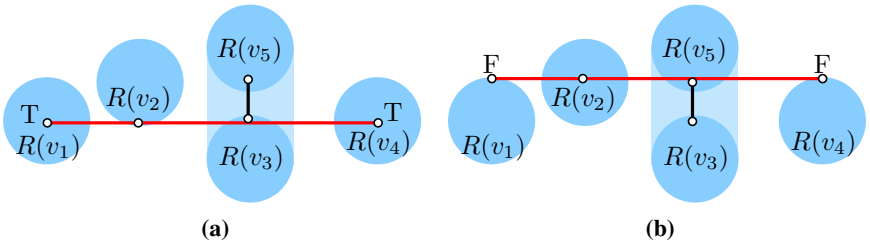


Figure 7.8: *Variable* gadget for the reduction to the AGD-L_2-$\mathcal{R}$ problem in its `true` (a) and `false` (b) configurations.

horizontal line. These three vertices represent the three literals of the clause. While $R(v_2)$ and $R(v_3)$ maintain the usual convention to encode the truth value of the represented variable, for $R(v_1)$ it is inverted. It can be easily realized by negating the value of the variable. The gadget contains three more vertices: v_4, v_5, and v_6, and edges (v_1,v_2), (v_2,v_3), (v_1,v_3), (v_3,v_4), and (v_5,v_6). The center of $R(v_i)$, with $i = 4,5,6$, lies inside $\mathcal{T}$. Region $R(v_4)$ is completely contained in pipe $P(v_1,v_2)$, except for an arbitrarily small part Π, which lies inside $\mathcal{T}$. Consider the two points l and r in which the boundary of $R(v_4)$ intersects $P(v_1,v_2)$. The boundary of $R(v_5)$ is tangent to the leftmost segment of the convex hull H of $\{l,r\} \cup R(v_3)$. Region $R(v_6)$ completely lies to the right of the rightmost segment of H, except for an arbitrarily small part. Neither $R(v_5)$ nor $R(v_6)$ intersects $P(v_1,v_2)$.

If all the literals are set to `false`, then v_4 must lie below edge (v_1,v_2) (and hence in Π). However, visibility between Π and $R(v_3)$ is prevented by edge (v_5,v_6) (Fig. 7.7b). Otherwise, if at least one of the literals is set to `true`, then an anchored drawing of γ can be realized (see Fig. 7.7c for an example). □

Theorem 6. AGD-L_2-$\mathcal{R}$ *is NP-hard.*

Proof: To prove hardness we reduce problem P3SAT to AGD-L_2-$\mathcal{R}$, under the hypothesis that Property A is satisfied (no overlap among vertex regions).

Let ϕ be an instance of P3SAT with n variables and m clauses. We describe how to construct an equivalent instance γ of AGD-L_2-$\mathcal{R}$.

For each variable x_j, with $j = 1,2,\ldots,n$, we introduce a *variable* gadget μ_j (see Fig. 7.8) composed of vertices $v_1,v_2,\ldots,v_5$ and edges (v_1,v_2), (v_2,v_4), and (v_3,v_5). The regions $R(v_i)$ for the vertices of the gadget are placed as follows: (a) for $i = 2,3,4$ region $R(v_i)$ completely lies to the right of $R(v_{i-1})$; (b) the centers of $R(v_5)$ and of

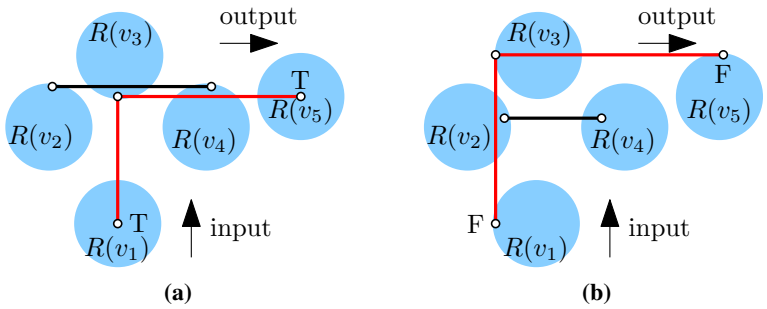


Figure 7.9: *Turn* gadget for the reduction to the AGD-L_2-$\mathcal{R}$ problem in its `true` (a) and `false` (b) configurations.

$R(v_3)$ are vertically aligned; (c) the centers of $R(v_1)$ and of $R(v_4)$ are horizontally aligned; (d) the centers of $R(v_2)$ and of $R(v_3)$ lies inside $P(v_1,v_4)$; and (e) the projections of the centers of $R(v_3)$, $R(v_1)$, $R(v_2)$, and $R(v_5)$ appear bottom-up in this order. In any anchored drawing, edge (v_2,v_4) is forced to pass either below (v_3,v_5) (`true` configuration, Fig. 7.8a) or above (v_3,v_5) (`false` configuration, Fig. 7.8b).

In order to propagate the truth value of variable x_j it is possible to concatenate a sequence of variable gadgets $\mu_1,\ldots,\mu_k$ by identifying $R(v_4)$ of μ_i with $R(v_1)$ of μ_{i+1}, for $i = 1,\ldots,k-1$.

The *turn* gadget (Fig. 7.9) is composed of vertices v_i, for $i = 1,\ldots,5$, and edges (v_1,v_3), (v_2,v_4), and (v_3,v_5). The regions of the vertices are placed in such a way that:

(a) their centers appear left-to-right in this order: $R(v_2)$, $R(v_1)$, $R(v_4)$, and $R(v_5)$; (b) their centers appear bottom-up in this order: $R(v_1)$, $R(v_4)$, $R(v_5)$ and $R(v_3)$; (c) $R(v_2)$ and $R(v_4)$ are horizontally aligned; (d) $R(v_1)$ and $R(v_3)$ are vertically aligned; (e) $R(v_2)$ lies outside $P(v_1,v_3)$ except for an arbitrarily small part; and (f) $R(v_3)$ lies outside $P(v_2,v_4)$ except for an arbitrarily small part. In any anchored drawing, edge (v_3,v_5) is enforced to lie either below (`true` configuration, Fig. 7.8a) or above (v_2,v_4) (`false` configuration, Fig. 7.9b). The relative positions of these edges enforce the correspondence of the positions of v_1 and v_5, and thus the transmission of the correct truth value.

The *not* gadget (Fig. 7.10a) is composed of vertices v_i, for $i = 1,\ldots,6$, and edges (v_1,v_2), (v_2,v_6), (v_5,v_6), and (v_3,v_4). The regions of the vertices are placed in such a way that: (a) their centers appear left-to-right in this order: $R(v_5)$, $R(v_3)$, $R(v_2)$, $R(v_1)$, and $R(v_4)$; (b) their centers appear bottom-up in this order: $R(v_1)$, $R(v_2)$, $R(v_4)$,

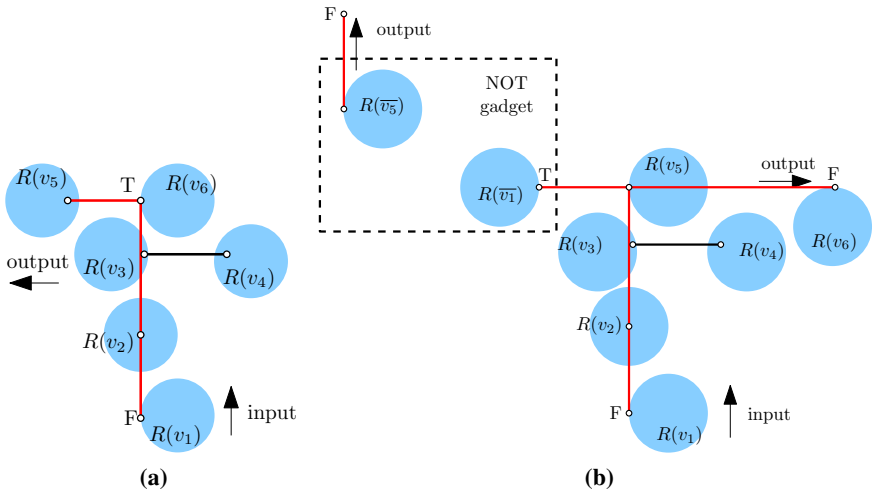


Figure 7.10: (a) *Not* gadget. (b) *Split* gadget in its `true` configuration.

$R(v_3)$, and $R(v_5)$; (c) $R(v_5)$ and $R(v_6)$ are horizontally aligned; and (d) $R(v_1)$ and $R(v_6)$ are vertically aligned. If the input variable is false, edges (v_1, v_2) and (v_2, v_6) are almost tangent to $R(v_1)$ and to $R(v_6)$, and edge (v_3, v_4) is below edge (v_5, v_6). Hence, edge (v_5, v_6) is aligned with the centers of v_5 and v_6, giving raise to a true value. If the input variable is true, edges (v_1, v_2) and (v_2, v_6) are vertically aligned with the centers of $R(v_1)$ and $R(v_6)$, and edge (v_5, v_6) is below edge (v_3, v_4). Hence, edge (v_5, v_6) is almost tangent to $R(v_5)$ and $R(v_6)$, giving raise to a false value.

The *split* gadget (Fig. 7.10b) is composed of vertices v_i, for $i = 1, \ldots, 7$, plus a set of additional vertices $\overline{v_j}$, for $j = 1, \ldots, 6$, belonging to a not gadget, and edges (v_1, v_2), (v_2, v_5), (v_5, v_6), (v_3, v_4), and $(v_5, \overline{v_1})$, plus the set of edges of the not gadget. The regions of the vertices are places in such a way that: (a) their centers appear left-to-right in this order: first all the vertices of the not gadget ending with $\overline{v_1}$, then $R(v_3)$, $R(v_2)$, $R(v_1)$, $R(v_4)$, and $R(v_6)$; (b) their centers appear bottom-up in this order: $R(v_1)$, $R(v_2)$, $R(v_4)$, $R(v_6)$, and $R(v_5)$; (c) the center of vertices $R(v_5)$ and $R(\overline{v_1})$ are horizontally aligned; (d) $R(v_1)$ and $R(v_5)$ are vertically aligned; and (e) $R(v_3)$ and $R(v_4)$ are horizontally aligned.

The *clause* gadget is illustrated in Fig. 7.11. The truth values of the input literals completely determine the drawing of path (v_1, v_3, v_6, v_8). Edge (v_5, v_{10}) can always be drawn as the short segment joining a point in $R(v_5)$ to a point in $R(v_{10})$. Edge

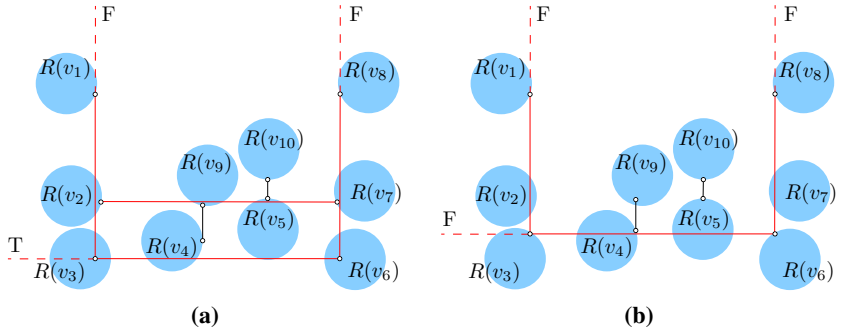


Figure 7.11: *Clause* gadget. The dashed lines represent the three input truth values.

(v_4, v_9) can be drawn as a segment tangent to $R(v_4)$ only if the literal attached to v_3 is true. as in Fig. 7.11a. This allows drawing edge (v_2, v_7) as an horizontal segment inserted between edges (v_4, v_9) and (v_5, v_{10}). Otherwise, if all the three literals are false, the obstruction represented by edges (v_4, v_9) and (v_5, v_{10}) does not allow for a crossing-free drawing of (v_2, v_7). If the literal attached to v_1 is true, then edge (v_2, v_7) can be drawn above edge (v_5, v_{10}). Finally, if the literal attached to v_8 is true, then edge (v_2, v_7) can be drawn below edge (v_4, v_9) which is drawn tangent to $R(v_9)$. □

7.5 Conclusions and Future Work

We considered the ANCHORED GRAPH DRAWING problem in several settings, showing that, provided that the input instance do not have overlaps between vertex regions (Property A), the problem of producing planar drawings is NP-hard in most of the settings. The only exception is for the case with rectilinear drawings and uniform distances (square-shaped regions), for which a polynomial-time algorithm is provided in Section 7.3.

We leave open the following questions: (*i*) Does problem AGD belong to class NP? (*ii*) The instances in our NP-hardness proofs can be augmented to equivalent instances whose graphs are biconnected (we omit details for space reasons). In these instances, different truth values correspond to different embeddings. What is the complexity of AGD when the input graph is triconnected or has at least a fixed embedding? (*iii*) What if we allow the vertex regions to (at least partially) overlap?

Chapter 8

Heuristics for Visualizing Georeferenced Graphs

This chapter describes an approach based on *Topology + Geography* (see Chapter 1) for drawing graphs whose vertices have a geographical position, or *georeferenced graphs*, which is the case of routing graphs. The approach deals with the *Relation To Geography* challenge. The work is based on the intuition that vertices can be moved within a limited span around their given positions to improve the readability of the drawing. Chapter 7 studied a corresponding graph-planarity problem that resulted NP-hard in many cases of interest, therefore this chapter deals with the visualization of georeferenced graphs by means of heuristics. The work introduces a 2.5D visual metaphor for georeferenced graphs, and describes a force-directed graph drawing algorithm that considers geographical constraints. The work ends with the description of a prototype tool, GeoGraph, which was tested through automated experiments with both real and artificial data. A preliminary version of this chapter was published in [DDP+15a].

8.1 Introduction

We address the problem of exploring a georeferenced graph, i.e., a graph with some geographic information associated to it. Typical applications include for example, visualizing Internet routing events, coordinating search and rescue teams, supervising medical evacuation squads, monitoring ad-hoc networks, and, on a more familiar and playful side, exploring location-based social networks.

The requirements of the interface are the following: the area of interest consists of

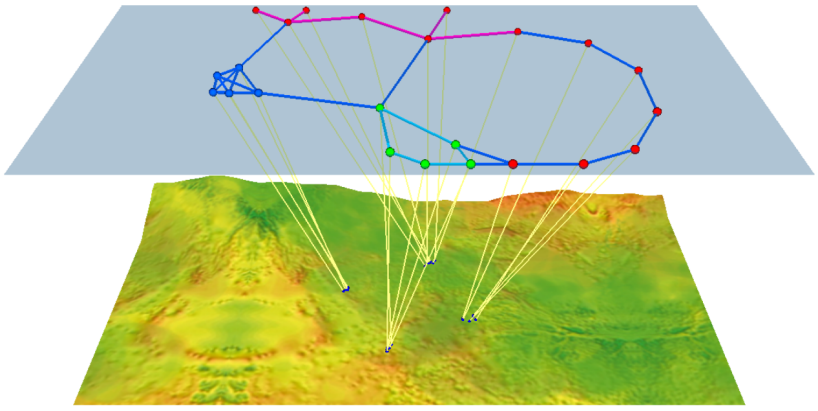


Figure 8.1: A snapshot of the proposed 2.5D interface. The *logical layer*, above, shows the networked data, while the *geographic layer*, below, displays actual locations of the entities contained in the logical layer, whenever available.

a terrain where a number of entities are located, and possibly move. Their geographic position may be declared by the entities themselves, tracked by radar stations, inferred from their transmissions, or, in some cases, completely unknown. Entities have a number of relationships, such as established connections, similarity, reachability, etc. The purpose of the interface is to represent in the most intuitive and unambiguous way both the relationships among the entities and their positions, conveying at the same time the degree of uncertainty associated with the geographic information.

User's tasks involve the analysis of both the networked and the geographic dimensions of the information. Let's suppose, for example, that the data represented come from a social network. Typical queries may be: What is the shortest friendship chain leading from a friend of mine living in London to anyone located in Berlin? Is it possible to find a friendship chain from London to Berlin without involving any person living in Rome? How many friends of my friends are currently in the same location as I am now?

We propose an innovative 2.5D paradigm to visually explore data with both a relational and a geolocalized nature. Our proposal is based on first separating and then integrating back again the networked and the geographic information. Namely, the geographic information is shown within a map, called *geographic layer*, while a second *logical layer* is devoted to the relational information. The two equally-sized rectangular layers are placed one above the other, and viewed from a side in a 2.5D

fashion, so that there is no overlap among them, i.e., no ambiguity between the two types of information (see Figure 8.1).

Entities are placed on the logical layer in such a way to reduce cluttering and to make their relationships readable and clear, while leaders are used to relate each entity on the logical layer to its known location on the geographic layer, whenever available.

All the screenshots in this chapter are taken from a JavaScript demonstrative prototype, *GeoGraph*, implemented using the WebGL graphics library [The13], that runs within any compatible web browser.

8.2 Visualizing Networked and Geographic Data

In this section we describe a visualization system, based on a novel graphical metaphor, that overcomes the limitations of traditional solutions with respect to user tasks that are typical of the visualization of georeferenced networks.

Problem Statement

Our visualization problem has two different inputs: the first one is from the user, who controls a rectangular area on a map, which is the current *area of interest* to be monitored and that can be translated, rotated, and zoomed. The second one comes from the outside world and is, essentially, a set of relationships among the entities of the considered domain. Each entity comes equipped with its type, its position, and the degree of uncertainty of such geographic information, which is provided by specifying the size of a geometric shape (usually a circle) that approximates the area where the entity is assumed to be.

The available data defines a network, whose nodes are the entities and whose edges are the relationships among entities. The purpose of the system is to show both the networked and the geographic information, meeting the following high-level requirements.

Effectiveness. The visualization should show in a clear and readable way the number of entities located on the selected area, their current position, and their relationships.

Intuitiveness. The graphic metaphors and the interaction primitives should be natural and intuitive, with low cognitive load.

Robustness. The visualization should support incomplete information, handling, in particular, missing and uncertain geographic data.

Unambiguity. The representation should be accurate and unambiguous. For example, the degree of uncertainty of the geographic information should be clear.

User's Tasks

When exploring a georeferenced graph, relevant user's tasks involve both the networked and the geographic dimensions of the information. Some of these tasks address simple quantitative queries, as estimating what is the amount of entities that share some target location, finding the location that hosts the most interconnected entities, determining whether a specific location hosts unconnected clusters of entities, etc.

We also identify more complex tasks that strongly rely on the analysis of the structure of the networked information. For example, finding the shortest chain of entities leading from a source placed in location A to any target located in B; finding a chain from location A to location B that does not involve any entity located in C; determining how many entities are reachable with two edges and are placed in a specific location; determining how strong are the connections among entities placed in location A and entities placed in location B, etc.

All these high level tasks primarily require the ability of the user to explore the structure of the relationships among the entities on the logical layer. Secondarily, the user needs to quickly grasp the area that hosts a given entity or, conversely, the set of entities that are located in a given area. These basic operations are complicated by the fact that some entities may not have a location associated and that a number of entities may share the same location.

Limitation of 2D Interfaces

The simple approach of placing icons on a 2D map based on the entities' geographic coordinates would not meet the requirements. In particular, since some entities share the same location, any 2D map would fail to unambiguously show both the location of the entities and their relationships. Also, entities on a geographic map are rarely equispaced. Instead, it is often the case that they gather in specific points (e.g., cities). Long and short distances among entities have to be simultaneously represented in the same view. When some entities have very close locations, their icons overlap unless the user zooms in on them. This zooming in and out has a dramatic effect on the usability of the interface due to focus-and-context issues: when some specific details are on focus the whole picture is no longer in sight and vice versa.

To complicate this scenario, some entities may have unknown geographic position. For example, users of a geolocated social network may disallow their applications to take advantage of GPS data; some devices of an ad-hoc network may not host a ground positioning circuit; routers of a computer network may not have an associated administrative site; end-points of a phone or radio conversation may be unknown; etc.

Wherever such entities would be placed on a 2D map, it would result in an ambiguous representation since the user would assume that those positions are the actual positions of the entities.

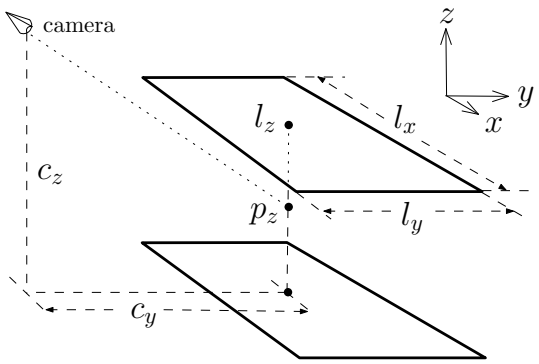
Finally, a 2D map does not convey in a natural way the degree of uncertainty of the geographic information. Placing icons on the map at the center of the area where the entity is supposed to be may result in the user's false confidence about the actual position of the entity. Drawing on the map some shadowed shapes, rectangles or circles, proportional to the degree of uncertainty on the position of the corresponding entities yields a representation that is not self-evident and that is confusing when several entities, with different shapes and different degree of uncertainty, are close one to the other.

Exploiting a 2.5D Visualization

Our strategy is to separate and simultaneously visualize the networked and the geographic information of the input dataset. Namely, the geographic information is represented on the *geographic layer*, which is in the bottom part of the interface, while the networked information is represented on the *logical layer*, which is parallel to the geographic layer and placed in the upper part of the interface. Leaders among the two layers relate nodes with their geographic locations, if any. The interface is shown in Fig. 8.1. In order to avoid overlaps between the two layers, which would give occlusion among the two types of information, we restrict their size to two equally-sized rectangles and suitably place the point of view on the longest side of the rectangles as shown in Fig. 8.2.

It has to be pointed out that the size and the orientation of the rectangles representing the logical and geographic layers are fixed with respect to the screen coordinates. Panning, zooming and rotating will have the effect of changing what is represented in the geographic and logical layers, but will not move the point of view with respect to the layers themselves. This design choice allows for a very simple and intuitive navigation of the scenario, that does not require the user to cope with fully 3D navigation primitives. Hence, in spite of its 3D flavor, our representation is a 2.5D one, both because the graph is actually drawn on the 2D surface offered by the logical plane and because the user interaction is limited to the 2D primitives of panning, zooming, and rotating. From a practical perspective, this is realized with four clipping planes that move together with the point of view and that cut out the world scene lying beyond the prescribed area.

Nodes are placed on the logical layer with the purpose of conveying as effectively as possible the structure of the graph, reducing cluttering and crossings among edges (see Fig. 8.3). To this purpose, we devised a specialized force-directed algorithm,



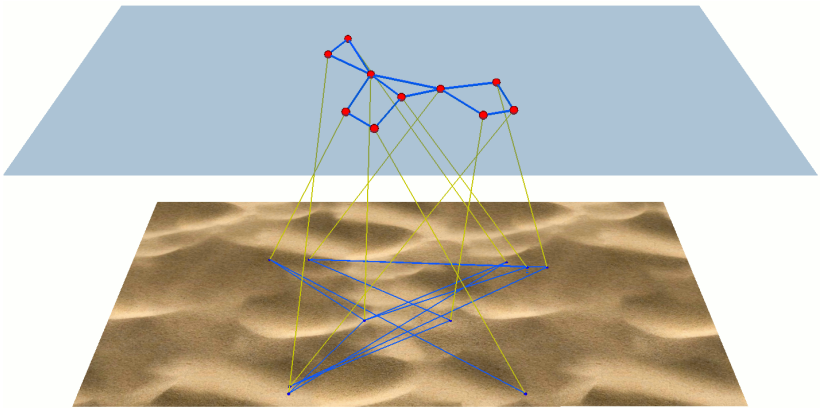
(a)

Figure 8.2: The position of the geographic and logical layers with respect to the point of view (we used $l_x = 1600$, $l_y = 900$, $l_z = 650$, $c_y = 2000$, $c_z = 1600$, $p_z = 260$).

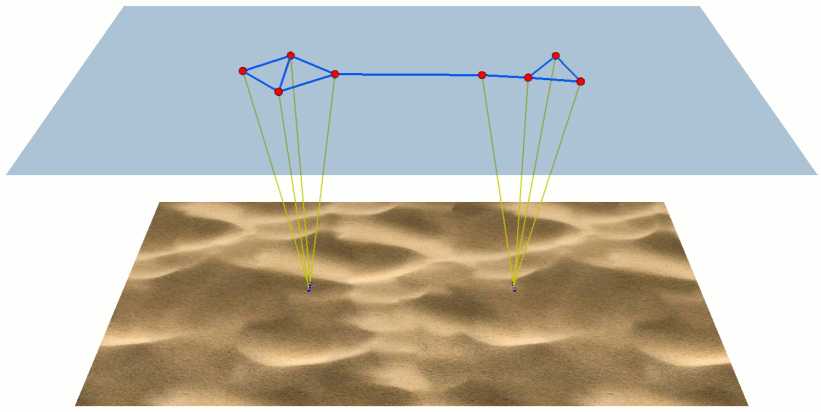
described in Section 8.4. The computed layout tries to achieve both evenly-spaced distribution of nodes and few crossings among edges, while seeking to minimize the distance of each node from the corresponding position on the geographic layer. The goal of the geographic layer, instead, is that of displaying the current position of each node shown on the logical layer. Such a position is represented by means of a marker on the map with a straight-line leader connecting it to the corresponding node on the logical layer. When the position of the node is affected by uncertainty the marker on the map is a geometric shape, usually a circle, enclosing the area where the corresponding object is supposed to be, and the leader consists of a cone with its apex on the node. Nodes with no geographic information associated have no marker on the map (see Fig. 8.4).

Therefore, in our approach, we have two types of links: (i) the edges on the logical layer and (ii) the leaders connecting the two layers. We privilege the readability of the graph induced by the first type of links, by trying to reduce crossings on the logical layer, which severely jeopardize the comprehension of the graph structure [PCJ97, Pur00]. Crossings among leaders have lower impact on readability, since the structure of the graph induced by them is of limited interest to the user. In fact, each leader establishes a connection between an entity and a geographic location, and paths of leaders are never considered by the user. Also, when the mouse hovers a vertex on the logical layer, the interface highlights its incident leader to help the user identify the leader endpoint on the geographic layer.

The graph represented in the logical layer is composed by all the entities inside



(a)



(b)

Figure 8.3: Two snapshots of the interface showing (a) crossings are reduced in the layout in the logical layer (the edges on the geographic layer are drawn for comparison) and (b) entities with common locations are clearly shown in the logical layer.

the area of interest of the user (*in-sight entities*) and all the entities that have links with such entities (*linked entities*). When the user zooms, rotates, or translates the area of interest (for example by pressing keyboard combinations or by dragging the mouse) the system updates the graph. Appearing nodes are placed on the border of the logical

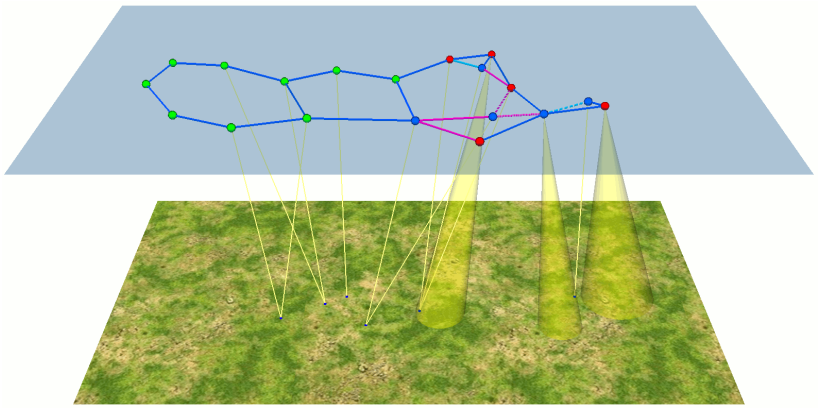


Figure 8.4: A georeferenced graph where the position of some entities is unknown and the position of other entities is known with some approximation.

layer, in the point which is nearer to their actual geographic position or in the point which is nearer to the position of one of the nodes they are linked to (in case of linked entities with no geographic position).

8.3 Related Work

Visual links in 2.5D visualizations. The use of visual links (i.e., edges) to highlight relationships between multiple views has been pioneered in the two-dimensional setting in [Wea05,AS07,SA06] and further explored in [SWS+11,HSS11]. The third dimension has been often used to add extra information to a traditional 2D layout. In particular, the use of inter-plane edges accounting for the relationships between nodes of separate 2D visualizations of the same graph has been proposed by the authors of VisLink [CC07]. The VisLink system allows the user to change the position of the planes hosting the drawings of the graph, stacking them horizontally, placing them side-by-side vertically, viewing them from the top, etc. In [SKKS08,LSKS10] a similar approach has been used for the exploration of interconnected pathways. In this case, the planes are usually more than two, and the system does not rely on the user ability for arranging the planes and for efficiently using the available screen space. Instead, the planes are placed on five sides of a cube and the user looks at them from the remaining side.

Spatial and Non-Spatial Data in Cartography. Our target visualization problem can

be viewed as a particular case of integrated spatial and non-spatial data visualization, where the non-spatial data can be modeled as a graph composed of entities and relationships among pairs of entities. Providing an integrated visualization of spatial and non-spatial data is a traditional topic in cartography where thematic maps are used to visualize the distribution of statistical variables. The values associated with the points on the maps can be represented, for example, by colors (choropleth maps), by the sizes of suitable symbols (proportional symbols maps), by the density of dots (dotted maps), etc. Although thematic maps may go so far as to represent a small chart for each location [WSD11], usually the non-spatial data represented has a very simple structure.

Reducing the Visual Clutter in Spatial Data. The problem of reducing the visual clutter of symbols on interactive maps has been approached with different techniques. Google Earth [Goo13] automatically collapses into a single symbol spatially coincident placemarks, which are exploded again in a cluster when clicked. Spatial dithering and changes in symbology (e.g., colour, opacity, line thickness and size) can be used to reflect the existence of unseen or coincident data [WDSC07].

8.4 The Retina Layout Algorithm

In this section we describe the algorithm that computes the network layout on the logical layer. Spring embedders are natural candidates for our application, since they grant, besides good quality results, the flexibility needed by our interactive system.

Although spring embedders are standard force-directed graph layout algorithms, our visualization problem is somehow special, as the logical layer is viewed from a side by the user, who, therefore, sees a picture distorted by the perspective. This has the effect of increasing the cluttering of the objects in the background with respect to those in the foreground. Hence, we conceived a variant of the spring embedder algorithm that computes the layout directly on the view plane, which is essentially the user's retina. This is why we dubbed it Retina layout algorithm.

A spring embedder boils down to be a very simple iterative process that, given the configuration at iteration i, computes the configuration at iteration $i+1$ by summing up, for each node, the forces acting on it, and then translating the node in the direction of the resulting force and proportionally to its magnitude. Such process then stops as soon as the sum of the forces acting on the nodes drops below a certain minimum threshold. Like (traditional) spring embedders, Algorithm Retina searches for an equilibrium configuration of a physical model obtained by replacing nodes with equally charged particles and edges with springs. On the one hand, since particles have the same charge, the *Coulomb force*, decreasing with the square of their dis-

tance, pushes them apart. On the other hand, as edges are replaced by springs, the *Hooke force* tends to keep adjacent particles close together (our springs have natural length zero and never exert a repulsive force). Further, in order "to keep the node close" to its geographic position, we introduced a *geographic force* that attracts each node to the point on the logical layer corresponding to the node geographic position. The initial placement of each node on the logical layer is given by the vertical projection of its geographic coordinates on the map. Nodes with no geographic information are initially placed in the barycenter of their neighbors, if any, or in a random point inside the boundaries of the logical layer that is not occupied by any other node. The sum of the forces acting on a node at a specific iteration i concides with the displacement vector that is applied to the current position of the node to obtain its position at iteration $i+1$. Boundary constraints are not implemented as forces, but rather as restrictions on the nodes' translations. Namely, suppose that for a node n we computed a translation vector v_n that would bring n outside the logical layer. Let x_n be the intersection between v_n and boundary of the logical layer. If n does not lie already on x_n, we move n to x_n. Otherwise, we replace v_n with the projection of v_n on the boundary of the logical layer.

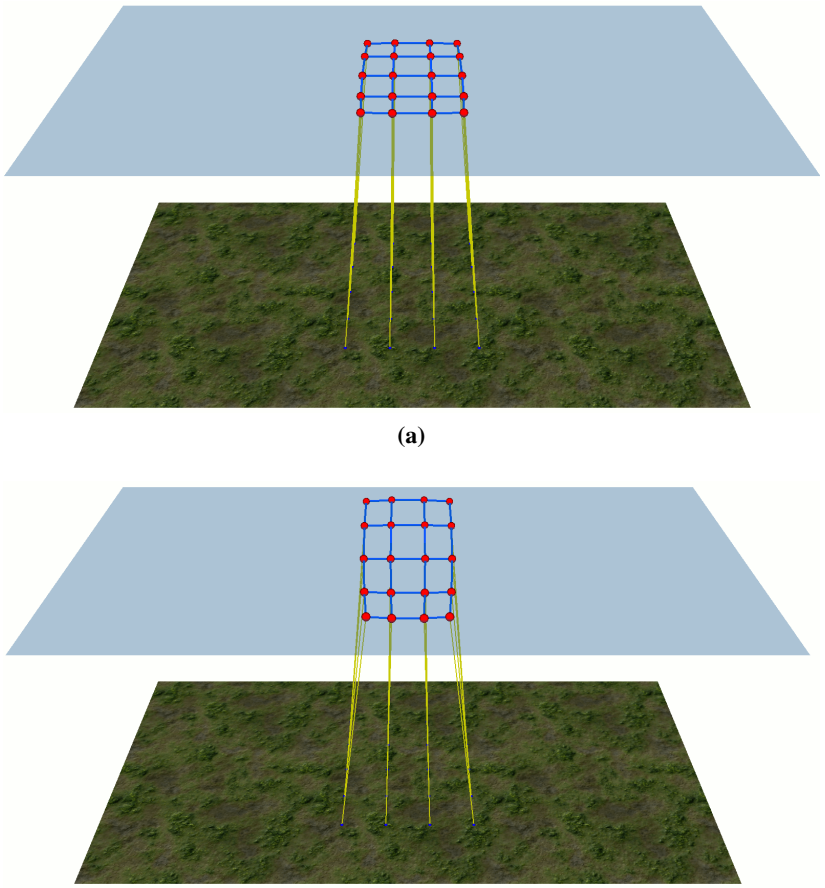
Algorithm `Retina` introduces a major, although conceptually simple, difference with respect to standard spring embedders. The forces and their sums are computed on the projections of the nodes' positions on the view plane, so to avoid the perspective distortion perceived by the user. Once the sum of the forces is computed for each node projection, the translation vectors are unprojected again from screen to world coordinates, yielding the new positions for the nodes.

Figure 8.5 shows a drawing of a 5×4 grid graph with and without the `Retina` algorithm. The perspective distortion of the traditional spring embedder that computes the layout on the logical plane is apparent in Fig. 8.5a. Such distortion is reduced in Fig. 8.5b.

8.5 Experimental Evaluation

We evaluated the effectiveness of the proposed 2.5D visualization and of Algorithm `Retina` by contrasting them with "traditional 2D visualization" where entities are place in their geographic position and the screen is fully devoted to a 2D representation of the area of interest. With respect to the problem requirements, we claim the following statements.

(i) The interface allows us to unambiguously represent networked data enriched with geographic information which may be missing or uncertain for some entity (Req. *Unambiguity*, *Robustness*).



(a)

(b)

Figure 8.5: The effect of the `Retina` algorithm is apparent when grid graphs are represented. (a) A grid graph drawn by a traditional spring embedder. The perspective distortion is apparent. (b) A grid graph with the `Retina` algorithm.

(*ii*) The interface allows us to clearly represent entities that have coincident or very close locations (Req. *Unambiguity*).

(*iii*) The logical and the geographic layers allow us to represent the networked information in a readable way (Req. *Effectiveness*, *Unambiguity*).

(*iv*) Algorithm `Retina` improves the readability of the drawing with respect to a traditional spring embedder run on the logical layer (Req. *Effectiveness*, *Unambiguity*).

The first claim, in our opinion, is self-evident, as we are not aware of alternative visualization techniques to represent in an unambiguous way geolocated networks where part of the entities have missing or uncertain geographic information. Therefore, we will give evidence of the other three claims by assuming that all the entities have a known position. To this end, we set up the experimental setting described in the following subsections. In all the experiments we allowed the on-line layout algorithm to reach an equilibrium configuration.

Quality Measures

To assess the effectiveness of the interface we adopted the following readability measures.

Crossings percent reduction ($\mathfrak{cpr}$). There is strong evidence in the literature that reducing the number of edge crossings is by far the most important aesthetic to improve the readability of a drawing [PCJ97,Pur00]. This metric estimates the ability of the interface to reduce the number of edge crossings in the representation of the networked data. In particular, we measured the average percent reduction of the number of crossings on the logical layer with respect to the number of crossings that would occur if the nodes were placed at their actual location, as in a traditional 2D visualization.

Homogeneous edge length ($\mathfrak{hel}$). This measure is based on the average percent deviation of edge lengths using a mean central tendency [FK13]. We compute this metric on the projections of the edges on the view plane:

$$\mathfrak{hel} = 1 - \frac{1}{m}\sum_{j=1}^{m}\left|\frac{|e_j| - |e|_{avg}}{\max\{|e|_{avg}, |e|_{max} - |e|_{avg}\}}\right|$$

where m is the size of the edge-set of the graph, $|e_j|$ is the length of the jth edge, $|e|_{avg}$ is the average edge length, and $|e|_{max}$ is the maximum edge length. Observe that $0 \leq \mathfrak{hel} \leq 1$. A value of $\mathfrak{hel} = 0$ could indicate that half of the edges have length zero while the other half have length $2|e|_{avg}$. A value of $\mathfrak{hel} = 1$ indicates all the edges have the same length.

Node separation ($\mathfrak{ns}$). Our purpose is to measure how well the interface is able to separate close nodes. Metric $\mathfrak{ns}$ is the minimum distance between the projections of the nodes on the view plane divided by the length of the diagonal of the viewport. In [FK13] a more sophisticated node separation measure has been proposed, which is

based on the perimeter of the largest empty rectangle enclosing each node. We chose a simpler and rougher measure because the one proposed in [FK13] is unpractical for our purposes, where 2D and 2.5D visualizations have to be contrasted, and would give us very limited additional benefits (see the appendix at the end of this chapter for more details).

We remark that our 2.5D visualization is unfavored by measures $\mathfrak{hcl}$ and $\mathfrak{ns}$ as it uses only a portion of the viewport to distribute nodes, whereas in a 2D visualization the whole viewport is used by the map. We also observe that our readability measures do not take into account the crossings among the edges that link the entities on the logical layer to their geographic positions. Such crossings are due to the 3D perspective and remain in the background when the user explores the network on the logical layer. Hence, they may have a reduced effect on the comprehension of the structure of such a network.

Test Suite

For our experiments we adopted a mix of real-life and synthetic data. In particular, we obtained real-life data about geolocalized entities from the maritime data collected by the Automatic Identification System (AIS), which is an onboard navigation safety device that broadcasts time-labeled messages with the location and characteristics of vessels in real time. We used a historic dataset of AIS data collected by the U.S. Bureau of Ocean Energy Management and by the U.S. National Oceanic and Atmospheric Administration [NOA]. Precisely, we used the dataset relative to the *Zone 14* of the Gulf of Mexico area, collected during January 2009. Starting from this dataset, we produced 300 test instances of increasing size and density.

First, we chose an arbitrary rectangular region whose aspect ratio is 16:9. We selected ten time instants uniformly distributed in the available time window. For each time instant, we selected in the area of interest the last s vessels broadcasting their position and considered their last update, with s ranging from 10 to 100 with a step of 10. This produced 100 sets of geolocated entities with distinct geographic positions.

AIS data are not provided with information about relationships between the vessels. Therefore, we created graph instances with different edge densities as follows. We randomly added edges to each set of n vertices, among all the possible $\frac{n*(n-1)}{2}$ edges, until we reached the density of 5%, 10%, and 15%. This produced 300 geolocated graphs, which compose the testsuite for our experiments.

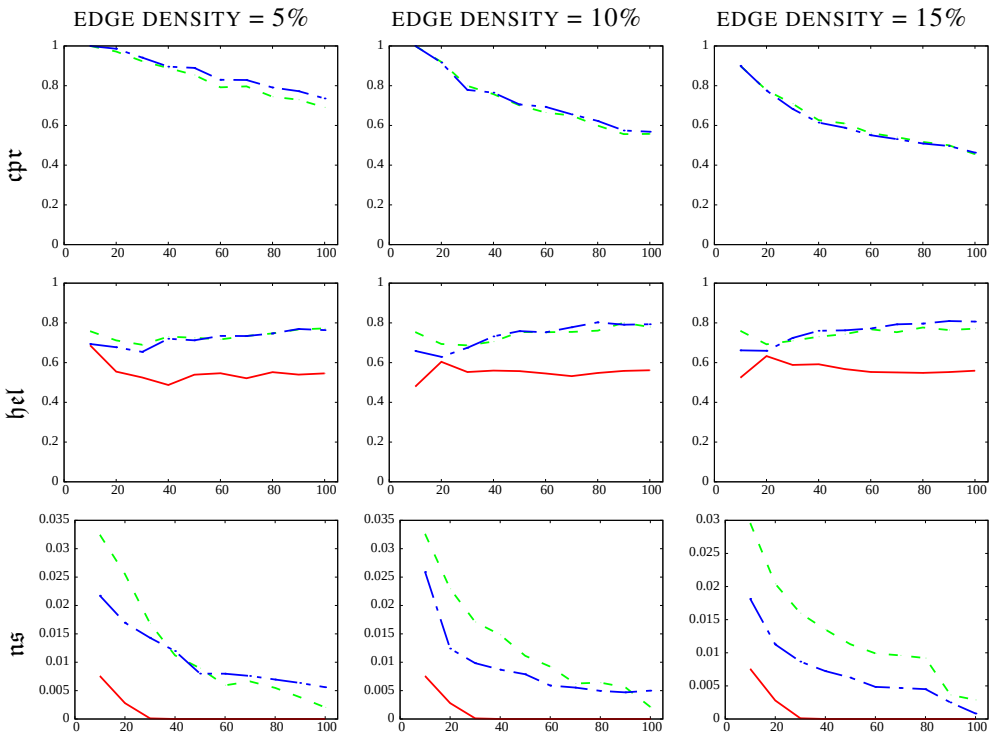


Figure 8.6: Results of the experiments. Charts in the first row report cpr measure; the second row is devoted to hel; and the third row to ns. The density of the graphs varies over the columns. The *x*-axis shows the size of the graphs, while the *y*-axis reports the measure of interest. The solid red line is the measure for the 2D visualization. The dashed green line is the measure for Algorithm `Retina`. The dash-dotted blue line is the measure for the traditional spring embedder with no `Retina` distorition.

Results and Discussion

Figure 8.6 shows the results of the experiments. Each dot corresponds to the average over ten values. Regarding measure cpr (Figs. 8.6, first row), crossings are completely removed for sparser and smaller graphs and are greatly reduced both by the traditional spring embedder and by Algorithm `Retina`, with the latter performing negligibly worse for sparse graphs (see Fig. 8.6, first row, first column). Even for bigger and denser instances of our dataset crossings are reduced of more than 40%.

We remark that the number of crossings on the geographic map for the denser graphs is huge (for example, the last point of Fig. 8.6, first row, last column, corresponds to more than $6,000$ crossings). The second row of Fig. 8.6 is devoted to metric $\mathfrak{hel}$. For all the tested densities and sizes the readability of the layout is steadily improved both by the traditional spring embedder and by Algorithm `Retina`, which appears to behave better for small graphs. The advantage of using a 2.5D interface is confirmed by these charts. It should be also noted that georefenced graphs coming from many real-life applications have the property that adjacencies are more likely among nodes that are geographically close. However, in order to prove the effectiveness of Algorithm `Retina` with respect to metric $\mathfrak{hel}$, we decided to perform our tests against unfavorable instances that do not exhibit this property. Algorithm `Retina` performs better than the traditional spring embedder with respect to metric $\mathfrak{ns}$ (see Fig. 8.6, last row), while the 2D visualization has very unsatisfactory results. Denser graphs are more effectively handled by Algorithm `Retina`. Overall, our experiments confirm that the 2.5D interface meets the system requirements. In particular, it allows us to represent the networked information in way that is considerably more readable than a traditional 2D interface. The adoption of Algorithm `Retina` is justified by its improvement on measures $\mathfrak{hel}$ and $\mathfrak{ns}$ at the expense of a negligible worsening of the $\mathfrak{cpr}$ measure. When the user focuses on smaller instances, the superiority of Algorithm `Retina` is apparent.

8.6 Conclusions and Future Work

We described a 2.5D visualization technique for exploring networked data enriched with geographic information, where the latter may have uncertain or missing values. Adapting, to our knowledge, for the first time a force-directed algorithm to a 2.5D setting, we conceived a variant of a spring embedder algorithm that directly computes the layout on the view plane (i.e., on the user's screen).

We measured the effectiveness of the proposed visualization and layout algorithm by contrasting them with a traditional 2D visualization with respect to three relevant readability measures. Both the experimentation and our experiences with the interface support our confidence about the effectiveness of the proposed techniques for small instances of geolocalized graphs. In fact, when the entities are more than a few dozens the readability measures show very poor performances and the drawing on the logical layer becomes too cluttered to be clearly readable (see Fig. 8.7).

Although the results are promising, our experiments only evaluate the static setting and do not account for the dynamic scenario, where changes occur both in the area of interest selected by the user and in the environment. An evaluation of the effectiveness

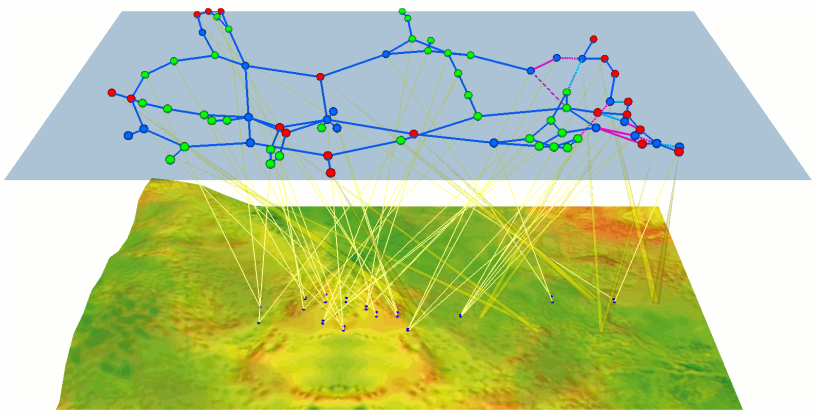


Figure 8.7: A georeferenced graph with 76 entities and 96 edges. The size of the graph makes cluttering hard to avoid.

of the dynamic scenario would be much more complex and could not leave aside a thorough user study. An interesting evolution of the `Retina` algorithm could consider additional forces to take into account crossings among leaders. One line of further investigation is given by the possibility of representing on the logical layer further information. A simple idea is to show a network that is wider than the area of interest (we call it *neighborhood visualization*), so to enhance the situational awareness of the user. Our preliminary experiments in this direction are encouraging.

Appendix

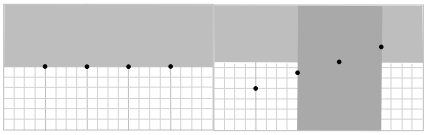
A measure for node separation alternative to measure $\mathfrak{ns}$ adopted in this chapter was proposed in [FK13]. Such a measure is based on the average percent deviation of separating rectangles. Given a node v_i on a plane, its *separating rectangle* is the maximum-perimeter axis-aligned rectangle that is contained within the bounding box of all nodes and that does not properly contain any other node in addition, possibly, to v_i (which may lay on the rectangle's boundary).

A node separation metric $\mathfrak{ns}'$ based on separating rectangles could be defined as follows ([FK13]):

$$\mathfrak{ns}' = 1 - \frac{1}{n}\sum_{i=1}^{n}\left|\frac{|v_i| - |v|_{avg}}{\max\{|v|_{avg}, |v_{max}| - |v|_{avg}\}}\right|$$

where n is the size of the vertex-set of the graph, $|v_i|$ is the perimeter of the separating rectangle of the ith node, $|v|_{avg}$ is the average over all nodes, and $|v|_{max}$ is the maximum over all nodes.

Measure $\mathfrak{ns}'$ is more sophisticated than measure $\mathfrak{ns}$ adopted in this chapter. However, we remark that it is difficult to efficiently compute the separating rectangles, unless the algorithm in [AS87] is adapted to this purpose. Also, there are configurations where a better node separation yields a worse $\mathfrak{ns}'$ measure. Consider, for example, Figure 8.8. Nodes in configuration (b) are better separated than nodes in configuration (a), as they are further apart from each other. In configuration (a) the separating rectangle is the same for all the four nodes (shaded area). This gives $\mathfrak{ns}' = 1$, the highest possible value. Two of the separating rectangles of configuration (b) are shown with light and dark shaded areas (the other two are symmetrical). The perimeter of these two separating rectangles is 40 and 51, respectively, which gives a worse $\mathfrak{ns}'$ metric ($\mathfrak{ns}' = 0.88$).



(a) **(b)**

Figure 8.8: The configuration in (a) has $\mathfrak{ns}' = 1$, the highest possible value. The configuration in (b) has $\mathfrak{ns}' = 0.88$. The shaded areas represent the separating rectangles for some of the nodes.

For the above reasons, we decided to use the metric $\mathfrak{ns}$ based on the minimum node distance on the view plane, which is easy to implement and allows us to contrast 2D and 2.5D visualizations.

Appendices

Appendix: List of Publications

The following papers were published or submitted with the results of this thesis.

- Marco Di Bartolomeo, Yifan Hu. *There is More to Streamgraph Than Movies: Better Aesthetics via Ordering and Lassoing*. In Proc. 18th EG/VGTC Conference on Visualization (EuroVis '16). Eurographics, 2016. To appear.
- Davide Ceneda, Marco Di Bartolomeo, Valentino Di Donato, Maurizio Patrignani, Maurizio Pizzonia, Massimo Rimondini. *RoutingWatch: Visual Exploration and Analysis of Routing Events*. In Proc. 15th IEEE/IFIP Network Operations and Management Symposium (NOMS '16). IEEE, 2016. To appear.
- Marco Di Bartolomeo, Valentino Di Donato, Maurizio Pizzonia, Claudio Squarcella, Massimo Rimondini. *Discovering High-Impact Routing Events Using Traceroutes*. In Proc. 20th IEEE Symposium on Computers and Communications (ISCC '15). IEEE, 2015. Best International Paper Award.
- Giordano Da Lozzo, Marco Di Bartolomeo, Maurizio Patrignani, Giuseppe Di Battista, Davide Cannone, Sergio Tortora. *Drawing Georeferenced Graphs - Combining Graph Drawing and Geographic Data*. In Proc. 6th International Conference on Information Visualization Theory and Applications (IVAPP '15). SciTePress, 2015.
- Patrizio Angelini, Giordano Da Lozzo, Marco Di Bartolomeo, Giuseppe Di Battista, Seok-Hee Hong, Maurizio Patrignani, Vincenzo Roselli. *Anchored Drawings of Planar Graphs*. In Proc. 22nd International Symposium on Graph Drawing (GD '14). Springer-Verlag, 2014.
- Massimo Candela, Marco Di Bartolomeo, Giuseppe Di Battista, Claudio Squarcella. *Dynamic Traceroute Visualization at Multiple Abstraction Levels*. In Proc. 21st International Symposium on Graph Drawing (GD '13). Springer-Verlag, 2013.

Bibliography

[AAHS05] M. Abellanas, A. Aiello, G. Hernández, and R. I. Silveira. Network drawing with geographical constraints on vertices. In *Actas XI Encuentros de Geom. Comput.*, pages 111–118, 2005.

[ABHR+13] Basak Alper, Benjamin Bach, Nathalie Henry Riche, Tobias Isenberg, and Jean-Daniel Fekete. Weighted graph comparison techniques for brain connectivity analysis. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 483–492, New York, NY, USA, 2013. ACM.

[ACO+06a] Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clémence Magnien, and Renata Teixeira. Avoiding traceroute anomalies with paris traceroute. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pages 153–158, New York, NY, USA, 2006. ACM.

[ACO+06b] Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clémence Magnien, and Renata Teixeira. Avoiding traceroute anomalies with paris traceroute. In *Proc. IMC*, 2006.

[ADD+14] Patrizio Angelini, Giordano Da Lozzo, Marco Di Bartolomeo, Giuseppe Di Battista, Seok-Hee Hong, Maurizio Patrignani, and Vincenzo Roselli. *Graph Drawing: 22nd International Symposium, GD 2014, Würzburg, Germany, September 24-26, 2014, Revised Selected Papers*, chapter Anchored Drawings of Planar Graphs, pages 404–415. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[ADDF13] Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, and Fabrizio Frati. Strip planarity testing. In Stephen K. Wismath and Alexan-

der Wolff, editors, *Graph Drawing*, volume 8242 of *LNCS*, pages 37–48. Springer, 2013.

[AFT07] Brice Augustin, Timur Friedman, and Renata Teixeira. Measuring load-balanced paths in the Internet. In *Proc. IMC*, 2007.

[agc] MisuraInternet. https://www.misurainternet.it/.

[AP12] D. Archambault and H.C. Purchase. The mental map and memorability in dynamic graphs. In *Visualization Symposium (PacificVis), 2012 IEEE Pacific*, pages 89–96, Feb 2012.

[AP13a] Daniel Archambault and Helen C. Purchase. "the "map" in the mental map: Experimental results in dynamic graph drawing". *International Journal of Human-Computer Studies*, 71(11):1044 – 1055, 2013.

[AP13b] Daniel Archambault and HelenC. Purchase. Mental map preservation helps user orientation in dynamic graphs. In Walter Didimo and Maurizio Patrignani, editors, *Graph Drawing*, volume 7704 of *Lecture Notes in Computer Science*, pages 475–486. Springer Berlin Heidelberg, 2013.

[APP11a] D. Archambault, H.C. Purchase, and B. Pinaud. Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *Visualization and Computer Graphics, IEEE Transactions on*, 17(4):539–552, April 2011.

[APP11b] Daniel Archambault, HelenC. Purchase, and Bruno Pinaud. Difference map readability for dynamic graphs. In Ulrik Brandes and Sabine Cornelsen, editors, *Graph Drawing*, volume 6502 of *Lecture Notes in Computer Science*, pages 50–61. Springer Berlin Heidelberg, 2011.

[ark15] Ark. http://www.caida.org/projects/ark, 2015.

[AS87] A. Aggarwal and S. Suri. Fast algorithms for computing the largest empty rectangle. SCG '87, pages 278–290, New York, NY, USA, 1987. ACM.

[AS07] Aleks Aris and Ben Shneiderman. Designing semantic substrates for visual network exploration. *Information Visualization*, 6(4):281–300, 2007.

[atl15] RIPE Atlas. http://atlas.ripe.net, 2015.

[Aug] Bjorn Augustsson. Xtraceroute. `http://www.dtek.chalmers.se/~d3august/xt/index.html`.

[AWW09] K. Andrews, M. Wohlfahrt, and G. Wurzinger. Visual graph comparison. In *Information Visualisation, 2009 13th International Conference*, pages 62–67, July 2009.

[Bac07] C. Bachmaier. A radial adaptation of the sugiyama framework for visualizing hierarchical information. *IEEE Trans. on Visualization and Computer Graphics*, 13(3):583–594, 2007.

[BAM07] R. Bourqui, D. Auber, and P. Mary. How to draw clusteredweighted graphs using a multilevel force-directed graph drawing algorithm. In *Information Visualization, 2007. IV '07. 11th International Conference*, pages 757–764, July 2007.

[BBDW14] Fabian Beck, Michael Burch, Stephan Diehl, and Daniel Weiskopf. The State of the Art in Visualizing Dynamic Graphs. In R. Borgo, R. Maciejewski, and I. Viola, editors, *EuroVis - STARs*. The Eurographics Association, 2014.

[BBL12] Ilya Boyandin, Enrico Bertini, and Denis Lalanne. A qualitative study on the exploration of temporal changes in flow maps with animation and small-multiples. *Computer Graphics Forum*, 31(3pt2):1005–1014, 2012.

[BDM02] Giuseppe Battista, Walter Didimo, and A. Marcandalli. Planarization of clustered graphs. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *Graph Drawing*, volume 2265 of *LNCS*, pages 60–74. Springer Berlin Heidelberg, 2002.

[BEB+13] Marcelo Bagnulo, Philip Eardley, Trevor Burbridge, Brian Trammell, and Rolf Winter. Standardizing large-scale measurement platforms. *ACM SIGCOMM Computer Communication Review*, 43(2):58–63, 2013.

[BHJ+09] Mathieu Bastian, Sebastien Heymann, Mathieu Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *ICWSM*, 8:361–362, 2009.

[BL76] Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *JCSS*, 13(3):335 – 379, 1976.

[BLC12] Dominikus Baur, Bongshin Lee, and Sheelagh Carpendale. Touchwave: Kinetic multi-touch manipulation for hierarchical stacked graphs. In *Proceedings of the 2012 ACM International Conference on Interactive Tabletops and Surfaces*, ITS '12, pages 255–264, New York, NY, USA, 2012. ACM.

[BM99] François Bertault and Mirka Miller. *Graph Drawing: 7th International Symposium, GD'99 Štiřín Castle, Czech Republic September 15–19, 1999 Proceedings*, chapter An Algorithm for Drawing Compound Graphs, pages 197–204. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[BOH11] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3 data-driven documents. *IEEE Trans. Vis. Comput. Graphics*, 17(12):2301–2309, December 2011.

[Bos] Mike Bostock. Data-driven documents. http://d3js.org/.

[BPF14] Benjamin Bach, Emmanuel Pietriga, and Jean-Daniel Fekete. Graphdiaries: Animated transitions and temporal navigation for dynamic networks. *IEEE Transactions on Visualization and Computer Graphics*, 20(5):740–754, May 2014.

[Bro14] Nevil Brownlee. On searching for patterns in traceroute responses. In *Proc. PAM*, 2014.

[BS08] Michael Baur and Thomas Schank. Dynamic graph drawing in visone. Technical report, Fakultät für Informatik, Universität Karlsruhe, 2008.

[BW04] Ulrik Brandes and Dorothea Wagner. Visone - analysis and visualization of social networks. In Michael Jünger and Petra Mutzel, editors, *Graph Drawing Software*, Mathematics and Visualization, pages 321–340. Springer Berlin Heidelberg, 2004.

[BW08] Lee Byron and Martin Wattenberg. Stacked graphs – geometry & aesthetics. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1245–1252, November 2008.

[CC07] C. Collins and S. Carpendale. VisLink: Revealing relationships amongst visualizations. *IEEE Trans. on Visual. and Comp. Graph.*, 13(6):1192–1199, 2007.

[CC14] Giovanni Comarela and Mark Crovella. Identifying and analyzing high impact routing events with PathMiner. In *Proc. IMC*, 2014.

[CD05] P. F. Cortese and G. Di Battista. Clustered planarity. In *SCG '05: Proceedings of the twenty-first annual symposium on Computational geometry*, pages 32–34, New York, NY, USA, 2005. ACM Press.

[CDD+16] D. Ceneda, M. Di Bartolomeo, V. Di Donato, M. Patrignani, M. Pizzonia, and M. Rimondini. Routingwatch: Visual exploration and analysis of routing events. In *Network Operations and Management Symposium (NOMS), 2016 IEEE*, 2016.

[CDDS13] Massimo Candela, Marco Di Bartolomeo, Giuseppe Di Battista, and Claudio Squarcella. Dynamic traceroute visualization at multiple abstraction levels. In Stephen Wismath and Alexander Wolff, editors, *Graph Drawing (Proc. GD '13)*, volume 8242 of *Lecture Notes in Computer Science*, pages 500–511, 2013.

[CDM+05a] Lorenzo Colitti, Giuseppe Di Battista, Federico Mariani, Maurizio Patrignani, and Maurizio Pizzonia. Visualizing interdomain routing with BGPlay. *Journal of Graph Algorithms and Applications, Special Issue on the 2003 Symposium on Graph Drawing, GD '03*, 9(1):117–148, 2005.

[CDM+05b] Lorenzo Colitti, Giuseppe Di Battista, Federico Mariani, Maurizio Patrignani, and Maurizio Pizzonia. Visualizing interdomain routing with BGPlay. *J. Graph Alg. and App.*, 9(1):117–148, 2005.

[CGC13] Giovanni Comarela, Gonca Gürsun, and Mark Crovella. Studying interdomain routing over long timescales. In *Proc. IMC*, 2013.

[Cis15] Cisco Systems, Inc. IOS Embedded Event Manager. `http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-embedded-event-manager-eem/`, 2015.

[CKN+03] Christian Collberg, Stephen Kobourov, Jasvir Nagra, Jacob Pitts, and Kevin Wampler. A system for graph-based visualization of the evolution of software. In *Proceedings of the 2003 ACM Symposium on Software Visualization*, SoftVis '03, pages 77–ff, New York, NY, USA, 2003. ACM.

[CLT+11] Weiwei Cui, Shixia Liu, Li Tan, Conglei Shi, Yangqiu Song, Zekai Gao, Huamin Qu, and Xin Tong. Textflow: Towards better understanding of evolving topics in text. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2412–2421, Dec 2011.

[CM13] Sergio Cabello and Bojan Mohar. Adding one edge to planar graphs makes crossing number and 1-planarity hard. *SIAM J. Comput.*, 42(5):1803–1829, 2013.

[COZ08] Ying-Ju Chi, Ricardo Oliveira, and Lixia Zhang. Cyclops: The AS-level connectivity observatory. *ACM SIGCOMM Comput. Commun. Rev.*, 38(5):5–16, September 2008.

[CTFD09] Ítalo Cunha, Renata Teixeira, Nick Feamster, and Christophe Diot. Measurement methods for fast and accurate blackhole identification with binary tomography. In *Proc. IMC*, 2009.

[data] Box office mojo. `http://www.boxofficemojo.com`.

[datb] Eurostat. `http://ec.europa.eu/eurostat/en/web/lfs`.

[datc] Google domestic trends. `https://www.google.com/finance/domestic_trends`.

[datd] Linux kernel. `https://github.com/torvalds/linux`.

[date] Nyc opendata. `https://nycopendata.socrata.com/Social-Services/311-Service-Requests-from-2010-to-Present/erm2-nwe9`.

[datf] Yahoo! finance. `http://finance.yahoo.com`.

[datg] Ycharts. `https://www.ycharts.com`.

[DBN88] G. Di Battista and E. Nardelli. Hierarchies and planarity theory. *Systems, Man and Cybernetics, IEEE Transactions on*, 18(6):1035–1046, 1988.

[DDP+14] Marco Di Bartolomeo, Valentino Di Donato, Maurizio Pizzonia, Claudio Squarcella, and Massimo Rimondini. Mining network events using traceroute empathy. Technical Report arXiv:1412.4074, Cornell University, Dec 2014.

[DDP+15a] Giordano Da Lozzo, Marco Di Bartolomeo, Maurizio Patrignani, Giuseppe Di Battista, Davide Cannone, and Sergio Tortora. Drawing georeferenced graphs - combining graph drawing and geographic data. In Lars Linsen, Andreas Kerren, and José Braz, editors, *Proceedings of the 6th International Conference on Information Visualization Theory and Applications, IVAPP 2015, Berlin, Germany, 11-14 March, 2015.*, pages 109–116, 2015.

[DDP+15b] Marco Di Bartolomeo, Valentino Di Donato, Maurizio Pizzonia, Claudio Squarcella, and Massimo Rimondini. Discovering high-impact routing events using traceroutes. In *Computers and Communication (ISCC), 2015 IEEE Symposium on*, 2015.

[DE02] T. Dwyer and P. Eades. Visualising a fund manager flow graph with columns and worms. In *Information Visualisation, 2002. Proceedings. Sixth International Conference on*, pages 147–152, 2002.

[DG02] Stephan Diehl and Carsten Görg. Graphs, they are changing. In MichaelT. Goodrich and StephenG. Kobourov, editors, *Graph Drawing*, volume 2528 of *Lecture Notes in Computer Science*, pages 23–31. Springer Berlin Heidelberg, 2002.

[DGWC10] M. Dork, D. Gruen, C. Williamson, and S. Carpendale. A visual backchannel for large-scale events. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1129–1138, Nov 2010.

[DH] Marco Di Bartolomeo and Yifan Hu. There is more to streamgraph than movies: Better aesthetics via ordering and lassoing. In *Proc. 18th EG/VGTC Conference on Visualization (EuroVis '16). To appear.*

[DM03] Adrian Dumitrescu and Joseph S. B. Mitchell. Approximation algorithms for TSP with neighborhoods in the plane. *J. Algorithms*, 48(1):135–159, 2003.

[DTDD07] Amogh Dhamdhere, Renata Teixeira, Constantine Dovrolis, and Christophe Diot. Netdiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data. In *Proc. CoNEXT*, 2007.

[Duf06] Nick Duffield. Network tomography of binary network performance characteristics. *IEEE Transactions on Information Theory*, 52(12):5373–5388, 2006.

[EH00a] Peter Eades and Mao Lin Huang. Navigating clustered graphs using force-directed methods. *J. Graph Algorithms Appl.*, 4(3):157–181, 2000.

[EH00b] Peter Eades and Mao Lin Huang. Navigating clustered graphs using force-directed methods. *J. Graph Algorithms Appl.*, 4(3):157–181, 2000.

[FAM+11] Paolo Federico, Wolfgang Aigner, Silvia Miksch, Florian Windhager, and Lukas Zenk. A visual analytics approach to dynamic social networks. In *Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies*, i-KNOW '11, pages 47:1–47:8, New York, NY, USA, 2011. ACM.

[FB04] Michael Forster and Christian Bachmaier. Clustered level planarity. In Peter Emde Boas, Jaroslav Pokorný, Máriá Bieliková, and Július Štuller, editors, *SOFSEM 2004: Theory and Practice of Computer Science*, volume 2932 of *LNCS*, pages 218–228. Springer Berlin Heidelberg, 2004.

[FE01] Carsten Friedrich and Peter Eades. The marey graph animation tool demo. In Joe Marks, editor, *Graph Drawing*, volume 1984 of *Lecture Notes in Computer Science*, pages 396–406. Springer Berlin Heidelberg, 2001.

[FE02] Carsten Friedrich and Peter Eades. Graph drawing in motion. *Journal of Graph Algorithms and Applications*, 6(3):353–370, 2002.

[FHQ11] Michael Farrugia, Neil Hurley, and Aaron Quigley. Exploring temporal ego networks using small multiples and tree-ring layouts. *Proc. 4th International Conference on Advances in Computer-Human Interactions (ACHI 2011)*, 2011:23–28, 2011.

[FK13] J. Joseph Fowler and StephenG. Kobourov. Planar preprocessing for spring embedders. In *Graph Drawing GD '12*, volume 7704 of *LNCS*, pages 388–399. Springer, 2013.

[FMM+04] Anja Feldmann, Olaf Maennel, Z. Morley Mao, Arthur Berger, and Bruce Maggs. Locating Internet routing instabilities. *ACM SIGCOMM Computer Communication Review*, 34(4):205–218, August 2004.

[FQ11] Michael Farrugia and Aaron Quigley. Effective temporal graph layout: A comparative study of animation versus static display methods. *Information Visualization*, 10(1):47–64, 2011.

[FT04] Y. Frishman and A. Tal. Dynamic drawing of clustered graphs. In *IEEE Symposium on Information Visualization, 2004. INFOVIS 2004*, pages 191–198, 2004.

[GEY12] S. Ghani, N. Elmqvist, and J. S. Yi. Perception of animated node-link diagrams for dynamic graphs. *Computer Graphics Forum*, 31(3pt3):1205–1214, 2012.

[GKNpV93] Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Kiem phong Vo. A technique for drawing directed graphs. *IEEE Transactions On Software Engineering*, 19(3):214–230, 1993.

[God95] Michael Godau. On the difficulty of embedding planar graphs with inaccuracies. In Roberto Tamassia and Ioannis Tollis, editors, *Graph Drawing (GD '94)*, volume 894 of *LNCS*, pages 254–261. Springer, 1995.

[Goo13] Google Inc. Google Earth, 2013. `http://earth.google.com`.

[Goo15] Google Inc. Google Maps. `https://maps.google.com/`, 2015.

[GT01] Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.*, 31(2):601–625, 2001.

[GWY+11] Hanqi Guo, Zuchao Wang, Bowen Yu, Huijing Zhao, and Xiaoru Yuan. Tripvista: Triple perspective visual trajectory analytics and its application on microscopic traffic data at a road intersection. In *Visualization Symposium (PacificVis), 2011 IEEE Pacific*, pages 163–170, March 2011.

[HD12] Mountaz Hascoët and Pierre Dragicevic. Interactive graph matching and visual comparison of graphs and clustered graphs. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '12, pages 522–529, New York, NY, USA, 2012. ACM.

[HFT08] Yiyi Huang, Nick Feamster, and Renata Teixeira. Practical issues with using network tomography for fault diagnosis. *ACM SIGCOMM Computer Communication Review*, 38(5):53–58, 2008.

[HHWN02] Susan Havre, Elizabeth Hetzler, Paul Whitney, and Lucy Nowell. Themeriver: Visualizing thematic changes in large document collections. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):9–20, January 2002.

[HSS11] S. Hadlak, H. Schulz, and H. Schumann. In situ exploration of large dynamic networks. *IEEE Trans. on Visual. and Comp. Graph.*, 17(12):2334–2343, Dec 2011.

[Int16] International Data Corporation. Mobile Internet Users to Top 2 Billion Worldwide in 2016. `https://www.idc.com/getdoc.jsp?containerId=prUS40855515`, 2016.

[JCC+13] Umar Javed, Italo Cunha, David Choffnes, Ethan Katz-Bassett, Thomas Anderson, and Arvind Krishnamurthy. Poiroot: Investigating the root cause of interdomain path changes. *ACM SIGCOMM Computer Communication Review*, 43(4):183–194, August 2013.

[KAF+08] Daniel Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Görg, Jörn Kohlhammer, and Guy Melançon. *Information Visualization: Human-Centered Issues and Perspectives*, chapter Visual Analytics: Definition, Process, and Challenges, pages 154–175. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[KBMJ+08] Ethan Katz-Bassett, Harsha V Madhyastha, John P John, Arvind Krishnamurthy, David Wetherall, and Thomas E Anderson. Studying black holes in the Internet with Hubble. In *Proc. NSDI*, 2008.

[KBSC+12] Ethan Katz-Bassett, Colin Scott, David R Choffnes, Ítalo Cunha, Vytautas Valancius, Nick Feamster, Harsha V Madhyastha, Thomas Anderson, and Arvind Krishnamurthy. Lifeguard: Practical repair of persistent route failures. *ACM SIGCOMM Comput. Commun. Review*, 42(4):395–406, 2012.

[KYGS] Ramana Rao Kompella, Jennifer Yates, Albert G Greenberg, and Alex C Snoeren. Detection and localization of network black holes. In *INFOCOM '07*.

[Lic82] David Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11:185–225, 1982.

[LMR98] Kelly A. Lyons, Henk Meijer, and David Rappaport. Algorithms for cluster busting in anchored graph drawing. *J. Graph Algorithms Appl.*, 2(1), 1998.

[LMZ06] Mohit Lad, Dan Massey, and Lixia Zhang. Visualizing internet routing changes. *IEEE Trans. Vis. Comput. Graphics*, 12(6):1–11, 2006.

[LSKS10] A. Lex, M. Streit, E. Kruijff, and D. Schmalstieg. Caleydo: Design and evaluation of a visual analysis framework for gene expression data in its biological context. In *PacificVis 2010*, 2010.

[LvK10] Maarten Löffler and Marc J. van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235–269, 2010.

[LWW+13] Shixia Liu, Yingcai Wu, Enxun Wei, Mengchen Liu, and Yang Liu. Storyflow: Tracking the evolution of stories. *IEEE Trans. Vis. Comput. Graph.*, 19(12):2436–2445, 2013.

[Lyo] Gordon Lyon. Zenmap. https://nmap.org/zenmap.

[LZP+12] Shixia Liu, Michelle X. Zhou, Shimei Pan, Yangqiu Song, Weihong Qian, Weijia Cai, and Xiaoxiao Lian. Tiara: Interactive, topic-based visual text summarization and analysis. *ACM Trans. Intell. Syst. Technol.*, 3(2):25:1–25:28, February 2012.

[Max15] MaxMind, Inc. Geoip2. `https://www.maxmind.com/`, 2015.

[MELS95] Kazuo Misue, Peter Eades, Wei Lai, and Kozo Sugiyama. Layout adjustment and the mental map. *"Journal of Visual Languages & Computing"*, 6(2):183 – 210, 1995.

[MG-15] MG-SOFT. Net Inspector. `http://www.mg-soft.com/`, 2015.

[MHS+14] Liang Ma, Ting He, Ananthram Swami, Don Towsley, Kin K Leung, and Jessica Lowe. Node failure localization via network tomography. In *Proc. IMC*, 2014.

[mla15] MLab. http://www.measurementlab.net, 2015.

[MPP13] Pietro Marchetta, Valerio Persico, and Antonio Pescapè. Pythia: yet another active probing technique for alias resolution. In *Proc. CoNEXT*, 2013.

[Nag15] Nagios Enterprises, LLC. Nagios. `https://www.nagios.org/`, 2015.

[NOA] NOAA Coastal Services Center. `http://www.marinecadastre.gov` (acc. 2014).

[NS00] Chris North and Ben Shneiderman. Snap-together visualization: can users construct and operate coordinated visualizations? *International Journal of Human-Computer Studies*, 53(5):715–739, 2000.

[Pat06] Maurizio Patrignani. On extending a partial straight-line drawing. *International Journal of Foundations of Computer Science (IJFCS)*, 17(5):1061–1069, 2006.

[PB08] Mathias Pohl and Peter Birke. *Visual Information Systems. Web-Based Visual Information Search and Management: 10th International Conference, VISUAL 2008, Salerno, Italy, September 11-12, 2008. Proceedings*, chapter Interactive Exploration of Large Dynamic Networks, pages 56–67. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[PCJ97] H. C. Purchase, R. F. Cohen, and M. I. James. An experimental study of the basis for graph drawing algorithms. *J. Exp. Algorithmics*, 2, 1997.

[PHG07] HelenC. Purchase, Eve Hoggan, and Carsten Görg. How important is the "mental map"? – an empirical investigation of a dynamic graph layout algorithm. In Michael Kaufmann and Dorothea Wagner, editors, *Graph Drawing*, volume 4372 of *Lecture Notes in Computer Science*, pages 184–195. Springer Berlin Heidelberg, 2007.

[PN99] Ram Periakaruppan and Evi Nemeth. Gtrace - a graphical traceroute tool. In *Proc. 13th USENIX conference on System administration*, pages 69–78. USENIX Association, 1999.

[PRB08] Mathias Pohl, Florian Reitz, and Peter Birke. As time goes by: Integrated visualization and analysis of dynamic networks. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '08, pages 372–375, New York, NY, USA, 2008. ACM.

[PS08] HelenC. Purchase and Amanjit Samra. Extremes are better: Investigating mental map preservation in dynamic graphs. In Gem Stapleton, John Howse, and John Lee, editors, *Diagrammatic Representation and Inference*, volume 5223 of *Lecture Notes in Computer Science*, pages 60–73. Springer Berlin Heidelberg, 2008.

[Pur00] Helen C. Purchase. Effective information visualisation: a study of graph drawing aesthetics and algorithms. *Interacting with Computers*, 13(2):147–162, 2000.

[QoS] QoSient LLC. Argus. http://qosient.com/argus.

[Rai05] Marcus Raitner. Visual navigation of compound graphs. In János Pach, editor, *Graph Drawing*, volume 3383 of *LNCS*, pages 403–413. Springer Berlin Heidelberg, 2005.

[rfc] RFC 1918. address allocation for private internets. http://www.ietf.org/rfc/rfc1918.txt.

[RIP] RIPE NCC. RIPEstat. https://stat.ripe.net/.

[RM13] S. Rufiange and M.J. McGuffin. Diffani: Visualizing dynamic graphs with a hybrid of difference maps and animation. *Visualization and Computer Graphics, IEEE Transactions on*, 19(12):2556–2565, Dec 2013.

[Rob07] Jonathan C. Roberts. State of the art: Coordinated multiple views in exploratory visualization. In *Proc. CMV*, July 2007.

[Rob12] Maxwell J. Roberts. *Underground Maps Unravelled - Explorations in Information Design*. 2012.

[Rom] Roma Tre University. Radian: Traceroute visualization. `http://www.dia.uniroma3.it/~compunet/projects/radian/`.

[RPD09] F. Reitz, M. Pohl, and S. Diehl. Focused animation of dynamic compound graphs. In *Information Visualisation, 2009 13th International Conference*, pages 679–684, July 2009.

[SA06] B. Shneiderman and A. Aris. Network visualization by semantic substrates. *IEEE Trans. on Visual. and Comp. Graph.*, 12(5):733–740, 2006.

[sam15] SamKnows. https://www.samknows.com, 2015.

[San96] Georg Sander. Layout of compound directed graphs. Technical report, FB Informatik, Universitat Des Saarlandes, 1996.

[San99] G. Sander. Graph layout for applications in compiler construction. *Theoretical Computer Science*, 217(2):175 – 214, 1999.

[SdDF+11] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè. Broadband internet performance: A view from the gateway. In *Proc. SIGCOMM*, 2011.

[SKKS08] Marc Streit, Michael Kalkusch, Karl Kashofer, and Dieter Schmalstieg. Navigation and exploration of interconnected pathways. *Comput. Graph. Forum*, 27(3):951–958, 2008.

[SLN05] P. Saraiya, P. Lee, and C. North. Visualization of graphs with associated timeseries data. In *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*, pages 225–232, Oct 2005.

[SM91] K. Sugiyama and K. Misue. Visualization of structural information: automatic drawing of compound digraphs. *IEEE Trans. on Systems, Man and Cybernetics*, 21(4):876–892, 1991.

[SP08] Peter Saffrey and Helen Purchase. The "mental map" versus "static aesthetic" compromise in dynamic graphs: A user study. In *Proceedings of the Ninth Conference on Australasian User Interface - Volume 76*, AUIC '08, pages 85–93, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.

[SWS+11] M. Steinberger, M. Waldner, M. Streit, A Lex, and D. Schmalstieg. Context-preserving visual links. *IEEE Trans. on Visual. and Comp. Graph.*, 17(12):2249–2258, Dec 2011.

[Tam07] Roberto Tamassia. *Handbook of Graph Drawing and Visualization (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC, 2007.

[TDBET98] Ioannis G. Tollis, Giuseppe Di Battista, Peter Eades, and Roberto Tamassia. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1998.

[The13] The Khronos Group. WebGL, Web Graphic Library – OpenGL ES 2.0 for the Web, 2013. `http://www.khronos.org/webgl/` (acc. 2014).

[Tho] ThousandEyes Inc. Network monitoring software. `http://www.thousandeyes.com/`.

[Tib94] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.

[TM12] Yuzuru Tanahashi and Kwan-Liu Ma. Design considerations for optimizing storyline visualizations. *IEEE Trans. Vis. Comput. Graph.*, 18(12):2679–2688, 2012.

[TMW03] Soon Tee Teoh, Kwan-Liu Ma, and S. Felix Wu. A visual exploration process for the analysis of internet routing data. In *Proc. IEEE Visualization Conference (VIS '03)*, 2003.

[Vis] Visualware. VisualRoute. `http://www.visualroute.com`.

[WDSC07] Jo Wood, Jason Dykes, Aidan Slingsby, and Keith Clarke. Interactive visual exploration of a large spatio-temporal dataset: Reflections on a geovisualization mashup. *IEEE Trans. Vis. and C. Graph.*, 13(6):1176–1183, 2007.

[Wea05] C. Weaver. Visualizing coordination in situ. In *INFOVIS 2005*, Oct 2005.

[WM96] Xiaobo Wang and Isao Miyamoto. *Graph Drawing: Symposium on Graph Drawing, GD '95 Passau, Germany, September 20–22, 1995 Proceedings*, chapter Generating customized layouts, pages 504–515. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[WSD11] J. Wood, A. Slingsby, and J. Dykes. Visualizing the dynamics of London's bicycle hire scheme. *Cartographica*, 46(4):239–251, 2011.

[YFDH01] Ka-Ping Yee, Danyel Fisher, Rachna Dhamija, and Marti Hearst. Animated exploration of dynamic graphs with radial layout. In *Proc. INFOVIS'01*. IEEE Computer Society, 2001.