UiT THE ARCTIC UNIVERSITY OF NORWAY

Department of Computer Science

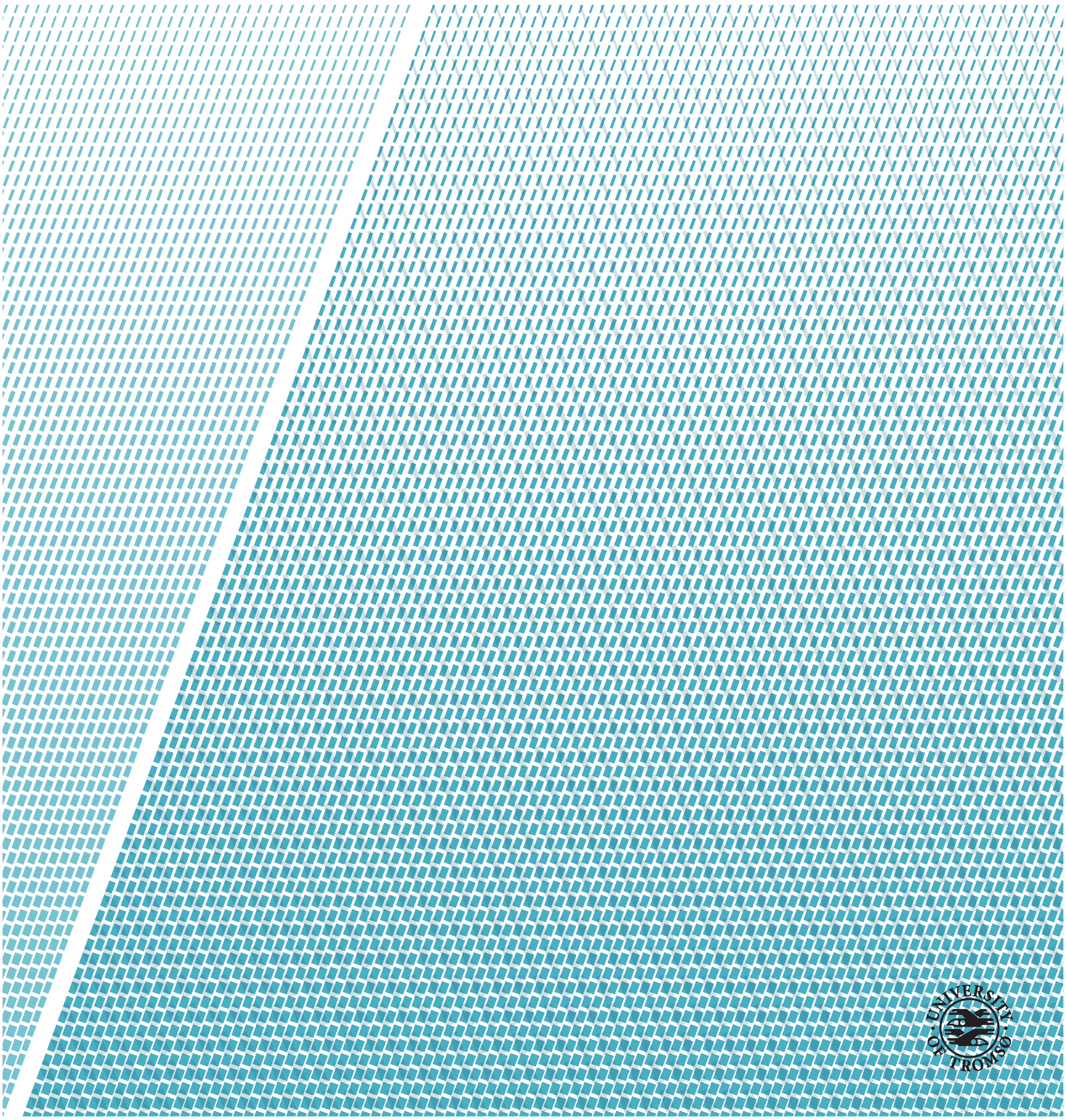
Large Multiples

Exploring the large-scale scattergun approach to visualization and analysis

—

Einar Holsbø

INF3990 — Master's thesis in computer science, May 2014



Abstract

We create 2.5 quintillion bytes of data every day. A whole 90% of the world's data was created in the last two years.[1] One contribution to this massive bulk of data is Twitter: Twitter users create 500 million tweets a day,[2] which fact has greatly impacted social science [24] and journalism [39].

Network analysis is important in social science [6], but with so much data there is a real danger of information overload, and there is a general need for tools that help users navigate and make sense of this.

Data exploration is one way of analyzing a data set. Exploration-based analysis is to let the data suggest hypotheses, as opposed to starting out with a hypothesis to either confirm or refute. Visualization is an important exploration tool.

Given the ready availability of large-scale displays [1], we believe that an ideal visual exploration system would leverage these, and leverage the fact that there are many different ways to visualize something. We propose to use wall-sized displays to provide many different views of the same data set and as such let the user explore the data by exploring visualizations. Our thesis is that a display wall architecture [1, 42] is an ideal platform for such a scheme, providing both the resolution and the compute power required. Proper utilization of this would allow for useful sensemaking and storytelling.

To evaluate our thesis we have built a system for gathering and analyzing Twitter data, and exploring it through multiple visualizations.

Our evaluation of the prototype has provided us with insights that will allow us to create a practicable system, and demonstrations of the prototype has uncovered interesting stories in our case study data set. We find that it is strictly necessary to use clever pre-computation, or pipelining, or streaming to meet the strict latency requirements of providing visualization interactively fast.

Our further experiments with the system have led to new discoveries in streaming graph processing.

1. http://www.ibm.com/software/data/bigdata/
2. http://about.twitter.com/company

Acknowledgements

Naturally thanks to my adviser Professor Otto Anshus and my co-adviser Associate Professor Phuong Hoai Ha for help and guidance. Thanks also to Associate Professor Lars Ailo Bongo for pointing out some serious structural problems in this text—some of which I may have been able to fix—and for general helpfulness and encouragement.

Thanks Jan-Ove “Kuken” Karlberg for our fruitful proofreading symbiosis. Thanks Bjørn Fjukstad, u r the wind beneath my wings. Thanks Erlend Graff for totalitarian typography and various proofreading. If I have gotten any usage correct in this text, it’s due to Erlend or to Garner’s Modern American Usage. I take personal responsibility for all the usage I’ve botched.

Thank you Isabelle Stanton for taking the time to answer some dumb questions about partitioning.

Bestemor & bestefar, pappa, mamma, Ing, Øyst, Steff, øvrig familie.

Everyone else that deserves mention by name: sorry.

Contents

List of Figures

List of Tables

/1 Introduction

According to IBM[1] we create 2.5 quintillion bytes of data every day, and 90% of the data in the world been created in the last two years. This data comes from our using social media, running Large Hadron Colliders, sequencing DNA, uploading cat videos to YouTube, etc., etc. Twitter reports[2] that their users produce 500 million tweets per day. Facebook surpassed one billion users in October 2012.[3] DNA sequencing cost has dropped dramatically in recent past (see Figure 1.1[4]).

An analyst faced with such massive data would soon suffer from information overload, and there is a general need for tools that help users explore and make sense of data sets. The popularity of content curation websites such as Tumblr or Reddit exemplifies this; finding the most interesting cat pictures on the Web — arguably the biggest and messiest data set of all — is a major hassle.

As data have grown in volume, variety, and velocity, so too has our access to compute power and screen real estate. Both the screen resolution and CPU power of a 2014 mobile phone will be far superior to those of the 2006 MacBook that this thesis is written on, at a much lower price. This opens up opportunities for novel approaches to data exploration, storytelling, and sensemaking.

1. http://www.ibm.com/software/data/bigdata/
2. http://about.twitter.com/company
3. http://www.facebook.com/facebook/info
4. Figure adapted from http://www.genome.gov/sequencingcosts/ by Bjørn Fjukstad.

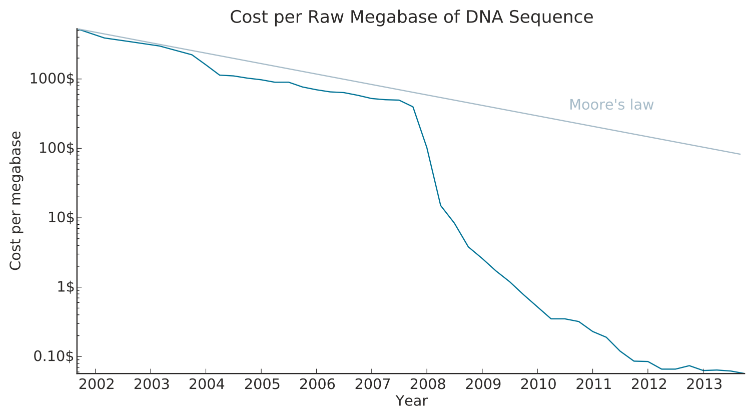


Figure 1.1: Cost per RAW Megabase of DNA Sequence.

1.1 Motivation and Idea

Network analysis is important in social science [6], and the field of computational social science — a marriage between computer science and social science — has emerged as a result of the ready availability of large social network data sets that is now offered by for e.g. Twitter [24].

Data exploration is one way of analyzing a data set. Exploration based analysis is to let the data suggest hypotheses, as opposed to starting out with a hypothesis to either confirm or refute. An important tool in such exploration is visualization.

But which visualization is conductive to hypotheses? And does a visualization that makes sense to user A also make sense to user B? Not necessarily. Shahaf et al. [55] reported, for e.g., quite disparate user comments to their Metro Maps visualization of scientific literature. Although they report that many of the negative comments could be addressed by improvements in UI design, the UI is part of visualization.

An ideal visual exploration system would leverage large-scale displays and the fact that there are many different ways to visualize something in such a way that a user could easily discover the stories and hypotheses hidden in their data.

Current work uses wall-sized displays to compare many data sets visualized in the same way [31, 53], uses wall-scale displays to provide a massively scaled up visualization of a single data set [20], or provides several suggested views of data one at a time on small displays.[5]

We propose to use wall-sized displays to provide many different views of the same data set and to let the user explore the data by exploring the many possible visualizations. Our thesis is that a display wall architecture (see Section

5. http://www.tableausoftware.com/

3.3) is an ideal platform for such a scheme, providing both the resolution and the compute power required.

1.2 Thesis Outline

To evaluate our thesis we have built a system for gathering Twitter data and exploring this data through many visualizations. Instead of working directly with the data set, we propose data exploration through visualization exploration. We are especially interested in timelines and storytelling, which we believe to be of interest to journalists and social scientists. Figure 1.2 shows our prototype in action. This thesis builds on our previous work in gathering and analyzing social network data from Twitter (see Holsbø et al., 2013 [32]).

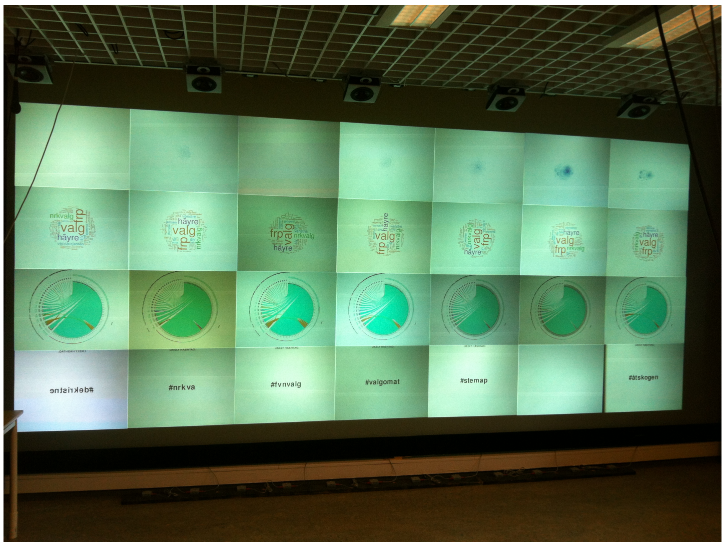


Figure 1.2: Large multiples

Our evaluation of a prototype system has provided us with insights that will allow us to create a practicable system, and demonstrations of the prototype has uncovered interesting stories, which we outline in Chapter 9.

This text has three major branches: we begin with a chapter on graph- and network theory, graph analysis, and visualization. We describe the architecture, design, implementation, and evaluation of our system for large-display visual exploration. Finally we describe and evaluate the groundwork we have laid down towards adapting our system for streaming graph analysis.

1.3 Contributions

The main contributions of this work are as follows:

- We build and evaluate a prototype for the scattergun approach to social network visualization, providing insights into how such a system should be realized.
- We identify a novel problem in streaming graph computation. We propose and evaluate a solution to this problem.
- We provide an overview of the different ways of gathering social media data and evaluate how these impact state of the art trend prediction.

2 Networks, Graphs, and Visualization

This chapter largely describes historical related work and important ground concepts behind our system. We first define basic network- and graph terminology, we also describe social networks, complex graph structures and visual analysis. We finally describe the state of the art of graph processing, and how complex graph structures are problematic to large-scale graph processing.

2.1 Graph Terminology

This section contains basic graph terms variously used in the other sections in this chapter and throughout the thesis.

A *graph*, $G = \{V, E\}$, is a mathematical model for connections and relations. Formally it comprises a set of *vertices*,[1] and a set of *edges*. Vertices, $V = \{v_1, v_2, \ldots, v_n\}$, represent some set of things we wish to model, and edges represent some connection or relation between pairs of vertices: $E = \{(v_i,v_j), \ldots, (v_y,v_z)\}$. Figure 2.1 below shows a node-link diagram of a toy graph with five vertices (the points) and eight edges (the lines).

We use graphs to model different kinds of networks — note that some

1. Also called *nodes*.

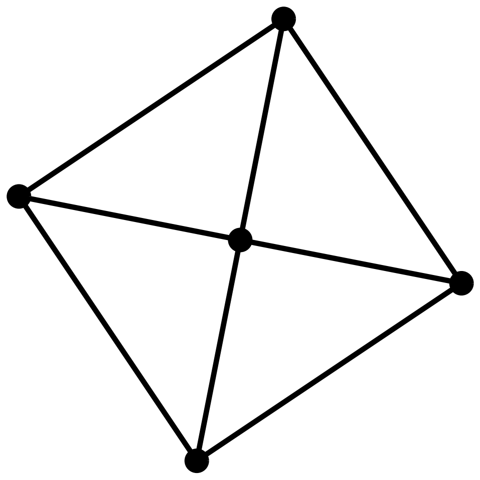


Figure 2.1: A node-link diagram of a toy graph. This is a common way to draw graphs.

authors use 'network' about special graphs. Sedgewick uses the word about directed, weighted graphs "for economy" [54], and Strang uses the word for any weighted graph [58]. We will use it in its broadest dictionary sense — E.g.: computer networks, collaboration networks, biological networks (such as protein interaction networks and a brain's network of neural connections).

Undirected graphs model symmetric relations: an undirected edge (v, u) connects v to u and vice versa. A collaboration network contains symmetric relations: if Bob has co-authored a paper with Alice, Alice must necessarily have co-authored this same paper with Bob. *Directed* graphs similarly model asymmetric relations. The edge (v, u) only connects v to u. The relation is reciprocal if and only if both (u, v) and (u, v) exist in E. A road network, where one-way streets might occur, can be modelled as a directed graph.

Edges can be *weighted*, in which case they have some number w associated with them. In a road network example this might be the length of a road.

If it is possible to follow edges from vertex v (respecting their direction) to vertex u, there is a *path* between v and u: they are *connected*. A graph is connected if every pair of its vertices is connected. The longest shortest path between two vertices is a graph's *diameter*.

The number of edges that v participates in is called the *degree* of v. In a directed graph we separate *in-degree* and *out-degree* — number of incoming edges, and number of outgoing edges.

2.2 Social Networks and Complex Graph Structures

The analysis and the visualizations in this work are based on social network theory and graph computation. We outline the more relevant concepts in this section.

Social networks are networks of social interactions and personal relationships.[2] Collaboration networks, for instance, are social networks. These networks have certain common structural characteristics, the most important of which we will highlight by examining a graph of character co-occurrence in the novel *Les Misérables*. The graph (see figure 2.2) was compiled in the 1990s by Donald Knuth [36], and although it models a small, fictional social network, its characteristics are found even in Facebook-scale social network graphs.

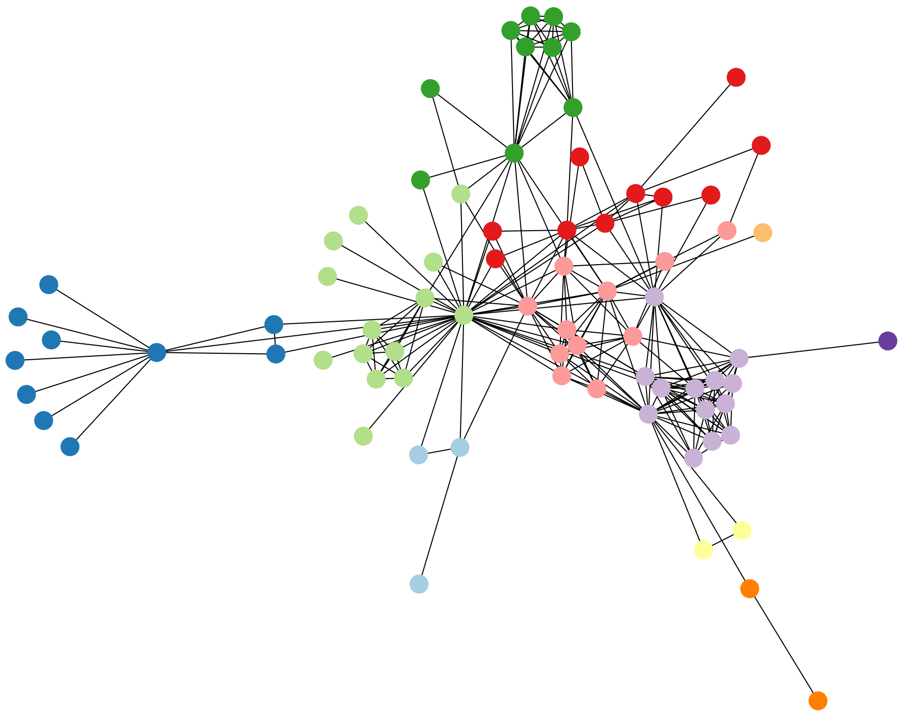


Figure 2.2: Character co-occurrence in Victor Hugo's *Les Misérables*.

2. Citing the New Oxford American Dictionary.

Scale-free

Scale-free networks have power law degree distributions. Formally: the fraction *P(k)* of vertices with k neighbors will, for large values of k:

$$P(k) \sim k^{-\gamma}$$

Where γ is a constant that describes the distribution.

Scale free means that some vertices have degrees that greatly exceed the average. Such vertices with a high k are called *hubs*. Figure 2.2's most obvious hub is towards the middle in light green. This is Jean Valjean, the novel's protagonist, who is bound to run into many other characters. Another slightly smaller hub is somewhat to the right of Valjean in light violet; this is Marius, who is another central character: love interest to Valjean's adoptive daughter and loosely affiliated with a revolutionary student club of some importance. Many nodes have noticeably fewer neighbors, some connected to the graph by only a single edge.

Small-world

In small-world networks, the average distance L between two randomly chosen nodes grows logarithmically with the number of nodes, N in the network:

$$L \propto \log N$$

This implies that most nodes aren't neighbors with one another, but most nodes can reach any other node by traversing a small number of edges. The graph in figure 2.2 has an average shortest distance of 2.6, which is to say that on average the shortest path between any two nodes is 2.6 edges long.

Travers and Milgram [62] performed a famous early small-world experiment in 1969. They asked 296 randomly chosen people to forward a letter to a Boston stockbroker via people they knew on a first-name basis; these people were in turn given the same instructions. In the end, 64 of the letters reached the intended target, and the median path length of these traces in the global social network was 6. This is where the concept "six degrees of separation" first appeared.

Homophily

The idea that similar people are more likely to form social connections: a notion at least as old as Plato [45].

We see evidence of homophily in figure 2.2. As mentioned, the figure shows Marius as a hub in light violet. Most of his neighbors in the graph are also light violet, and most of their neighbors again are light violet. The vertices thus colored are rich, young, revolutionary students that call themselves the Friends of the ABC. Most of Marius' friends are revolutionaries, hence (social homophily suggests) Marius is likely to be a revolutionary himself.

Community Structures

Similar to the idea of homophily: some characters are more closely tied to one another and will occur together more often, such as the above mentioned Friends of the ABC.

Giant Connected Component

A connected component is a subset *CC* of vertices such that (i) all pairs of vertices in *CC* are connected, and (ii) *CC* isn't a true subset of another set where (i) is true.

Social networks and other complex networks tend to have one giant connected component where most vertices participate. Figure 2.2 shows a single GCC.

2.3 Visual Analytics

The Figure 2.2–exercise was an example of *visual analysis*. The figure projected a novel of about 1200 pages onto about half a page worth of node-link diagram that held a lot of information about the socio-communal structures of the novel. This section will provide some insight in the problems and concepts that pertain to our visualizations.

Zhang et al. [66], succinctly outlines the visual analysis pipeline as follows: data loading, integration, transformation, data mining, and data interpretation.[3]

2.3.1 Graph Visualization

A graph can be visualized in a myriad of ways, and a reasonable study of the field would probably fill a master's thesis in its own right. We don't intend to provide anything close to such a study, our goal is simply to illustrate some common ways to draw graphs and to establish that there are always several

3. Paraphrasing the knowledge discovery pipeline of Fayyad & al. [18].

ways to visualize the same thing. Hence, in this section we will limit ourselves force-directed node-link diagrams (NLDS) and adjacency matrices.

Node-Link and Force-Direction

There are no rules for how a NLD should be laid out, but it's generally held that edges crossings should be minimized, vertices distributed evenly, and graph symmetry clearly shown [3, 22, 34, 50]. Battista et al. note that "In general, the optimization problems associated with these aesthetics are NP-hard" [3].

Figures 2.1 and 2.2 were both force-directed NLDS. Force-directed layouts try to emphasize community structure while keeping clarity so as to fulfill the aesthetics outlined above. They work by assigning attractive forces between adjacent vertices (i.e., vertices that are neighbors), and repulsive forces between all vertices. Fruchterman and Reingold [22] did important work in this area, building on the ideas of Eades [15] and Quinn & Breuer [51].

Force-directed methods are generally acknowledged to produce pleasing layouts, but they suffer from long run times as the general concept is similar to the n-body problem, and as such has a computational complexity of $O(|V|^2)$. Later work has gone towards improving on these long run times, Hu [34], whose algorithm is included with Mathematica, reduces the complexity by working on multiple levels of coarsened versions of the graph — i.e. simplified versions of the graph where $|V|$ and $|E|$ have been reduced by coalescing pairs of adjacent vertices into a single vertex.

Adjacency Matrix

An adjacency matrix is a graph data structure where the graph is represented as a $|V|^2$ matrix M, and entry $M_{u,v}$ is 1 if the edge (u,v) exists, 0 otherwise.

This can easily be translated to a visualization: figure 2.3 shows the *Les Misérables* graph from section 2.2. A 1 in the adjacency matrix is a colored square in the figure, a 0 is the absence of same. Adjacency matrices, like node-link diagrams can be presented in more or less helpful ways; the matrix in Sub-figure 2.3a has its rows and columns ordered by character name, while Sub-figure 2.3b has them ordered by community affiliation. The former looks rather noisy and unhelpful, while the latter shows clear social structures and signs of homophily.

A clear advantage the adjacency matrix has over the force-directed NLD is that it is computationally inexpensive to layout.

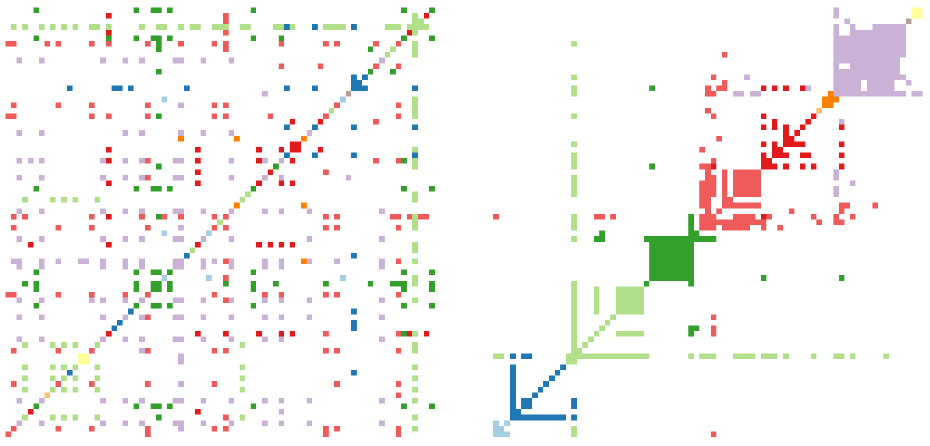

(a) Ordered by name **(b)** Ordered by cluster

Figure 2.3: Adjacency matrices for the *Les Misérables* graph.

2.3.2 Text Visualization

Text visualization is something of a new field, where the goal is to present an abstract, succinct view of a longer text.

The *wordcloud* [35] is by far the most common way to visualize text. Figure 2.4 shows a wordcloud generated from Edsger Dijkstra's 1968 article *Go To Statement Considered Harmful* [14]. Basically the wordcloud is a way to visualize a word count with more frequent words rendered larger than less frequent ones. It is common to remove stop words, punctuation, capitalization, etc., before counting the words.

Another text visualization is *theme river* [30], which is basically a wordcloud over time. Themes are represented as bands in a river that flows in a timeline, and the width of the band reflects the theme's relative prominence. Figure 2.5 shows a theme river constructed from Associate Press news wire stories in mid-1990.

2.4 Graph Processing

This section briefly outlines advances in graph processing at scale, some of which is relevant to our partitioning work.

The MapReduce model has been a corner stone in large scale data processing since Dean and Ghemawat's seminal 2008 paper [13]. In time, though, people realized how the model is ill-suited for graph problems: while MapReduce is excellent for IO-bound data parallel processing, graph computations aren't strictly speaking data parallel. The problem specifically is that a graph

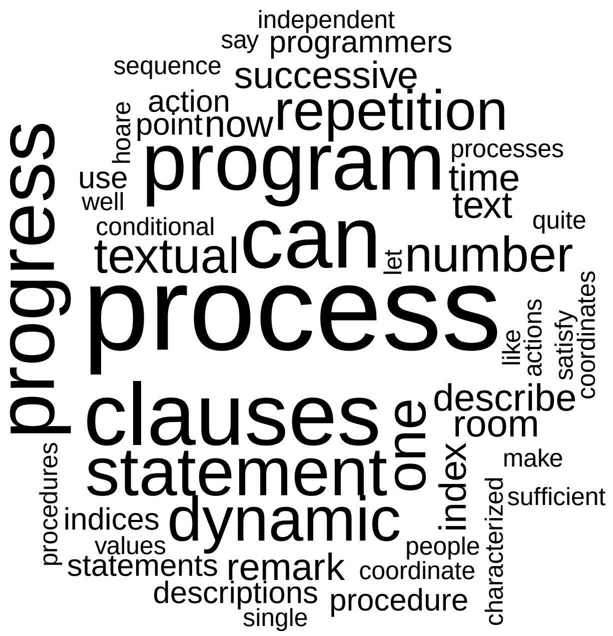


Figure 2.4: A wordcloud

computation is very often iterative, which entails many short runs of MapReduce, which again entails lots of disk IO and wasted time. Spark [65], which doesn't have this prohibitive startup cost, was introduced in 2010.

Google recognized the unsuitability of MapReduce for graph computing, and published Pregel[4] [44] in 2010. Pregel pretty much did for graph processing what MapReduce did for data parallel processing, and Apache Giraph[5] is to Pregel what Apache Hadoop[6] is to MapReduce.

Pregel — a cluster system like MapReduce is — introduced a novel *think like a vertex* programming model. Execution runs in bulk-synchronous parallel [63]: A vertex is given some task, e.g. *what is the weighted average of the PageRank of your neighbors*, performs this task, communicates the result with its neighbors, receives results from its neighbors, repeats until convergence. The communication stage crucially serves as a barrier synchronization, preventing tiresome races.

Parallel to Pregel from Malewicz et al., Bader et al. [2] were working STINGER for streaming graph processing. Pregel's graph model is essentially static, you

4. Named for the river in Köningsberg over which spanned the bridges of Euler's seven bridges problem.
5. http://giraph.apache.org/
6. http://hadoop.apache.org/

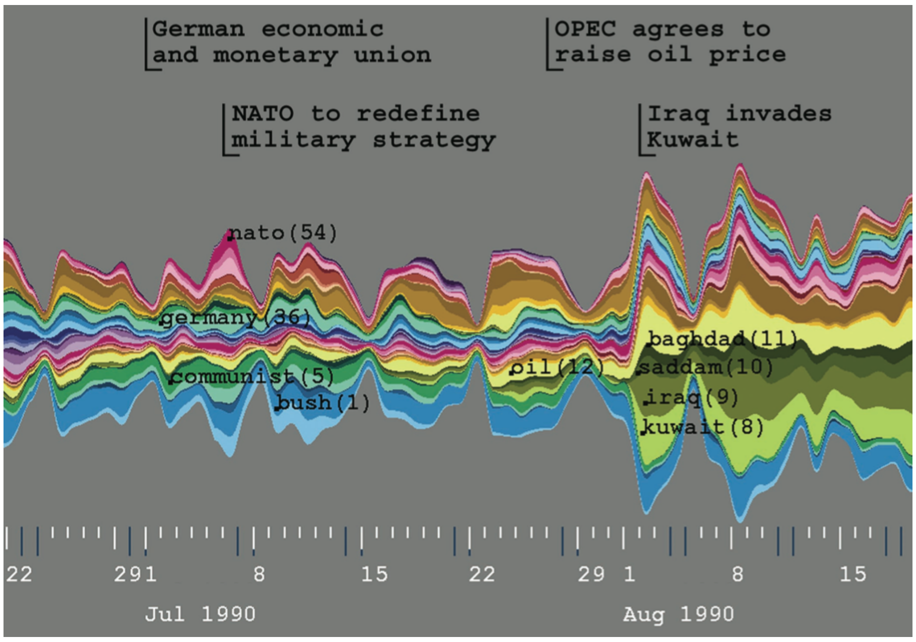


Figure 2.5: A theme river. Figure from Havre et al. [30].

load a graph, run some huge computation, get some output. STINGER's graph is modelled as an infinite stream of edge updates and deletions. This is a much more complicated model to work with, and the field is quite new. The infinite edge stream model requires new algorithms, data structures, and programming models, where STINGER primarily provides data structures. Ediger et al. [17] published a paper in 2011 presenting a framework for tracking connected components in streaming graphs. For comparison, the basic algorithm for finding connected components in a static graph was already well known in 1973 [33].

Interestingly, Google employs at least two special-purpose graph processing systems for the training of deep networks [11, 12].

2.5 The Problem of Distributing Complex Graphs

Our attempted move towards streaming graph processing on compute clusters (see Chapter 8) sparked an investigation into graph partitioning, we briefly outline the main points and problems here.

Large scale distributed graph processing systems, e.g. Pregel or GraphLab [43], partition the graph and distribute it to several machines, much as MapReduce would partition data to several machines. The problem with this is that real-world graphs are complex (most important here is the scale-free property described in section 2.2) and hence notoriously difficult to partition cleverly.

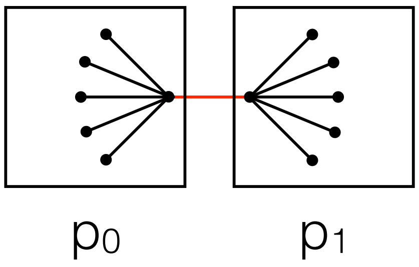
This problem is known as *balanced k-partitioning*: how to split a graph in k parts of equal size cutting[7] as few edges as possible. Researchers have worked on this problem since the 1970s at least [19], and the finding of an optimal solution has long since been shown to be NP-complete [23].

This isn't just an interesting academical problem. As most graph computations involve graph traversal, an edge cut is a message that has to be sent between machines. It is tempting to ignore the issue (as indeed Pregel does, encouraging the user to roll their own partitioner) and simply assign vertices to machines randomly, but it can be shown that the expected cut[8] size of a random k-partitioning is

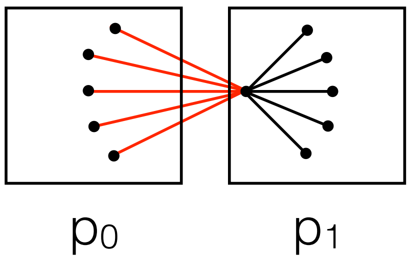
$$E[\frac{\#edges\ cut}{\#edges}] = 1 - \frac{1}{k}$$

which is quite expensive. It is, in fact, a very good way to approximate a maximal cut for any interesting value of k.

Figure 2.6 shows two graphs, where 2.6a has a cheap balanced 2-partition with a cut size of a single edge, shown in red. The graph in 2.6b shows the star motif that characterizes the hubs that are typical of scale-free networks. A hub has no cheap cut; half the edges would have to be cut no matter which vertices were put in which of the two partitions.



(a) Cheap



(b) Expensive

Figure 2.6: Two graph partitioning examples

7. **cut** (an edge), v.t.: to assign two adjacent vertices to different partitions so that the edge between them spans partitions.
8. **cut**, n.: the set of edges that span partitions.

3 Architecture and Design

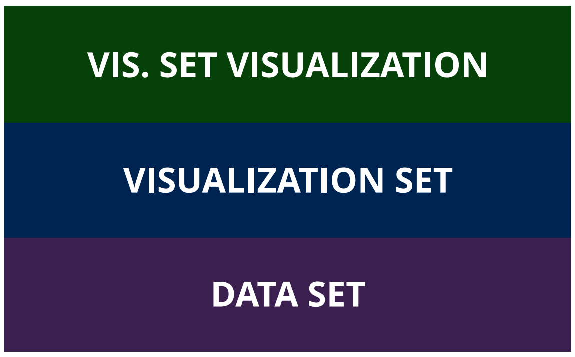


Figure 3.1: System architecture

Figure 3.1 shows our architecture's main abstractions. There are many ways of visualizaing a data set, in principle there is a visualization set that comprises all these different visualizations. A subset of this could be presented to the user; presenting a selection of visualizations to the user could be seen as a kind of meta-visualization itself.

In principle the data could be anything from log files to cat videos, in our prototype implementation we focus on Twitter data. Data could either be stored or generated on-site, or feched from elsewhere.

A view into the visualization set must be created and presented; we imagine a system where the user is presented a handful of visualizations, and can quickly exchange these for others if they don't show anything useful.

3.1 Requirements and Design Considerations

The visualization pipeline mentioned in 2.3 defines the requirements related to creating visualizations:

Data loading: The user needs to be able to load data into the system.
Integration: If the data comes from different sources, in different formats, or have to be integrated in some other manner, the system should support this.
Transformation: Data may have to be transformed or cleaned to be workable. E.g. the handling of missing values, normalization, outlier removal, noise reduction, reformatting. Different visualizations may require different transformations.
Data Mining: Fitting models to-, or inferring patterns from the data.
Visualization: The data-mined results presented to the user as a visualization.

It is also necessary for the user to be able to interact with the system and the different visualizations.

Lastly and importantly: low latency. The user should be able to quickly change views of the data.

3.2 System Design

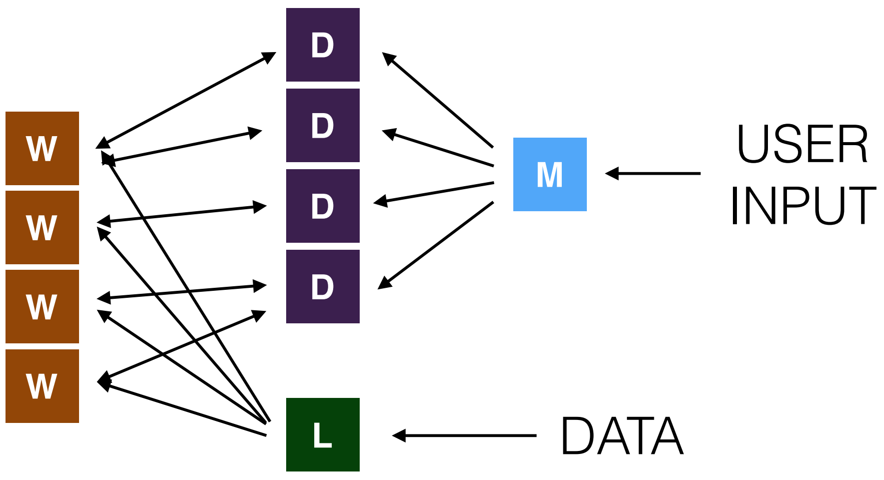


Figure 3.2: System design: workers (W), drawers (D), loader (L), and master (M)

Figure 3.2 shows our design. The user loads data into the system by means of a Loader (L), which performs data loading and integration. User requests for new visualizations go through the Master (M), which forwards instructions to the Drawers (D). The Drawers request visualizations from Workers (W); Workers are responsible for most of the heavy lifting, being in charge of transformation and data mining — both tasks that can be computationally expensive. When the data mined results are ready, they are returned to the Drawers, which then display the visualization. Note the similarity between the Drawer-Worker interaction here and the NAD-NAC interaction of WallScope described below.

Workers have a cache where they can store results from both transformation and data mining. This first because the user might want to see a visualization again, but also because some transformation and data mining results can be re-used for different visualizations. For e.g. does three of our implemented visualizations rely on an extracted communication graph to show different aspects of the data set.

3.3 Display Wall Technology

This section will briefly describe the Tromsø Large Display Wall, which was important to the design and development of our system. This section will be heavily based on the 2013 nine-year retrospective of Anshus et al. [1].

Display walls are wall-sized tiled displays composed of several smaller displays and computers. Displays on such large scales often offer screen resolution orders of magnitude larger than those of commodity desktop computers or laptops. Among the main benefits from this is the multi-level view that large-scale displays provide — far away from the display the user gets a top-level total view of whatever is displayed, close up they're provided high detail — and the collaboration that the large display encourages over that of standing several people stooped over a laptop [21].

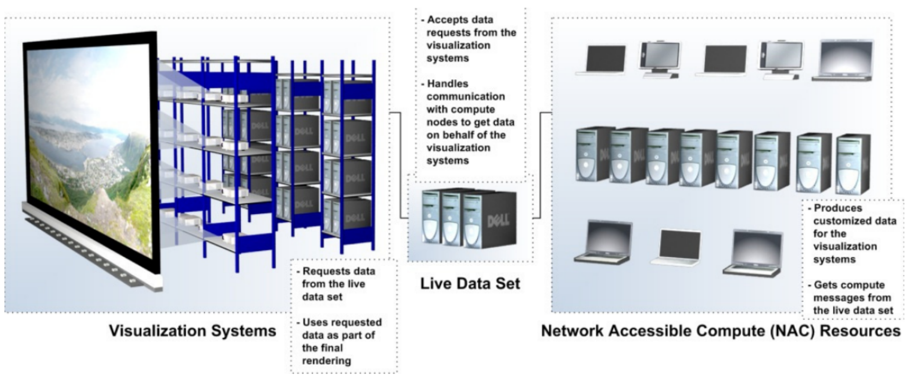

- Accepts data requests from the visualization systems
- Handles communication with compute nodes to get data on behalf of the visualization systems
- Requests data from the live data set
- Uses requested data as part of the final rendering
Live Data Set
- Produces customized data for the visualization systems
- Gets compute messages from the live data set
Visualization Systems
Network Accessible Compute (NAC) Resources

Figure 3.3: WallScope, with the display wall to the left of the figure.

A display wall is constructed by tiling several displays and computers [42]; the Tromsø Large Display Wall (TLDW) is a 22 megapixel rear-projected display comprising a 7 by 4 grid of projectors. These projectors are driven by a 28-node display cluster connected by a gigabit Ethernet switch.

Figure 3.3[1] shows an illustration of the TLDW to the left. The figure as a whole shows the WallScope architecture: a visualization system comprising Network-accessible Display resources and Network-accessible Compute resources. The pull-based NADs request visualizations from the NACs.

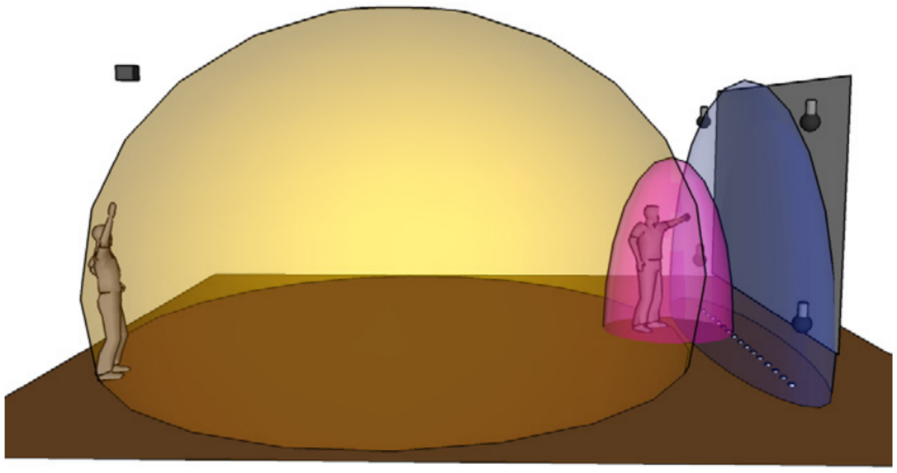


Figure 3.4: Interaction spaces

There are several interfaces for interaction with the TLDW, most notable of which are the Interaction Spaces of Stødle et al. [60]. Figure 3.4[2] shows three interaction spaces: the blue sphere is *Camera-sense*, which tracks objects in three dimensions by means of commodity web cameras; the pink sphere shows *Arm-angle*, which recognizes the angle at which a user points their arm by means of a ceiling-mounted pan-tilt-zoom camera; and finally the yellow sphere shows *Snap-detect*, which tracks a users location by way of four microphones picking up sharp sounds such as the snapping of fingers or the clapping of hands.

1. Figure from Hagen's 2010 Ph.D. dissertation [28].
2. Figure from Stødle's 2009 Ph.D. dissertation [59].

4 Implemented Visualizations

This chapter describes the visualizations we have implemented for- or fitted to our system.

Our prototype was built for Twitter type data, and our choice of visualizations is necessarily influenced by this. Graph analysis is central to Twitter analysis, so too are topics such as opinion mining, trend analysis, lexical analysis, etc.

4.1 Longitudinal Studies

It was clear to us from the start that it would be impractical to either implement or integrate enough different visualizations to fit, say, one on each tile of the 28-tile display wall, let alone enough of them to test our idea of allowing the user to quickly switch between sets of visualizations on a large-scale display. Given the temporal nature of Twitter data, it seems natural to present it as a time series. A handful of different visualizations at many different times provide many different views of the data.

Hence we arrange our visualizations as in figure 4.1. The x-axis is a time series of our data in which the data is separated into snapshots of equal length (e.g. all activity in a single day). The y-axis is different visualizations. Such

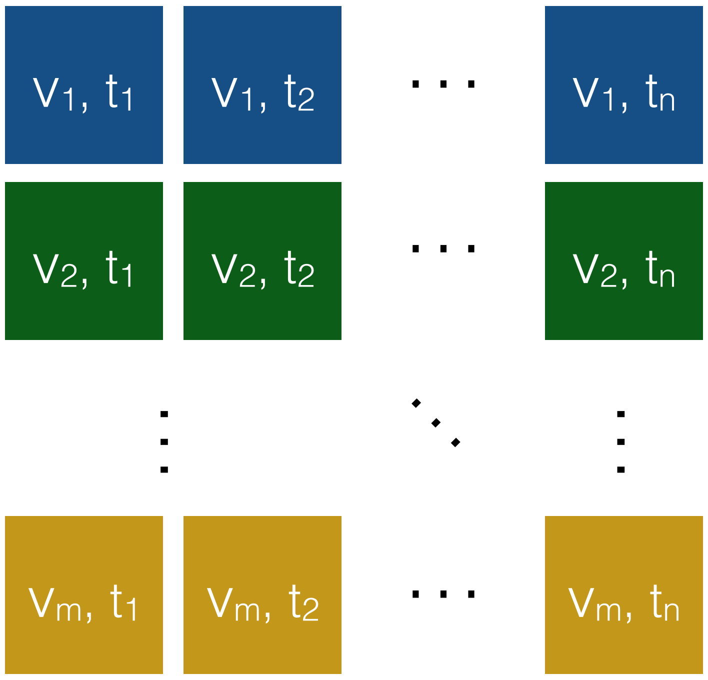


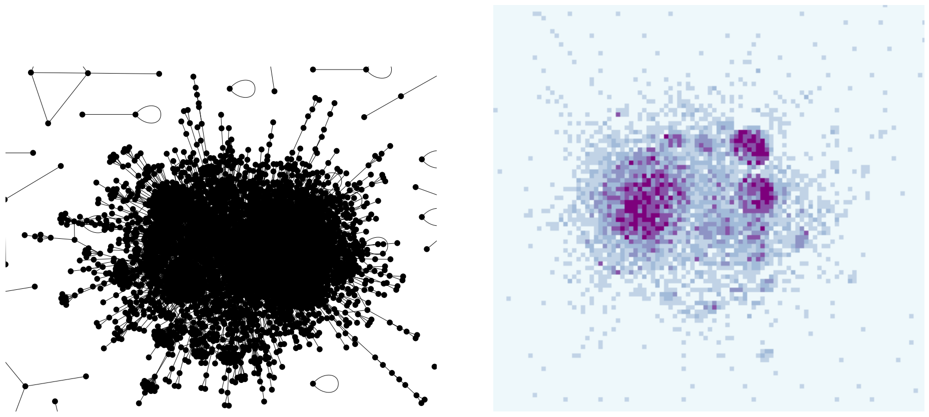
Figure 4.1: How we arrange visualizations on the TLDW. Different visualizations $\{v_1, v_2, \ldots, v_m\}$ show different steps in a time series $[t_1, t_n]$.

analysis over time is called longitudinal study: observe some phenomenon over time, e.g. politics-related Tweets and interactions; ascertain what changes and how it changes.

4.2 Force Direction and the Hairball Problem

As discussed in chapter 2, force-directed node-link diagrams and adjacency matrices can show community structure in a graph. We wish to visualize how the users in our data set communicate; Smith et al. [56] recently published a study in the conversational archetypes of Twitter such as *polarized crowd*, which shows how partisan groups of users largely ignore one another; *tight crowd*, which shows how tight-knit communities participate gather around e.g. a conference or hobby, etc. (six in total). Signs of these archetypes may provide the analyst with valuable insight.

A problem with visualizing graphs once they grow past a few hundred vertices and edges is that the complexity of the graph completely dominates the visualization. Figure 4.2 shows the @-graph of a snapshot from our data set; the graph contains roughly 5200 vertices and 7000 edges. A force-directed



(a) Node-link diagram (b) Heatmap

Figure 4.2: Hairball problems

NLD (Sub-figure 4.2a) appears as an intractable mess of edges with no clear structure at this size. We choose to use only the coordinates that result from a force-directed layout and heatmap these, as in Sub-figure 4.2b. This provides topographical information (e.g. clustering) that is impossible to discern in Sub-figure 4.2a.

We use the sfdp program from graphviz[1] to calculate our force-directed vertex positions. The sfdp implements the multi-level force-directed layout algorithm of Hu [34], which we briefly discussed at the end of Section 2.3.1.

4.3 Lexical Analysis

We use the R text mining[2] and wordcloud[3] packages to create a wordcloud for each snapshot in our data set. Section 2.3.2 showed an example wordcloud.

4.4 Circos

Circos [4] is a collection of Perl scripts used to create visualizations of connections and interactions in a data set. It was originally designed for visualizing genomic data and has been used in for e.g. Nature Reviews [8]. From Krzywinski et

1. http://www.graphviz.org/
2. http://cran.r-project.org/web/packages/tm/index.html
3. http://cran.r-project.org/web/packages/wordcloud/index.html
4. http://circos.ca/

al. [38]:

> Circos uses a circular ideogram layout to facilitate the display of relationships between pairs of positions by the use of ribbons, which encode the position, size, and orientation of related genomic elements.

We use Circos to create another view of the communication graphs of our data set. We first extract the 25 most active users in a snapshot and coalesce all other users into one fat 26th user. We then count the number of times these 26 users send @-messages mentioning one another and enter this data into the Circos table viewer.[5] We wish to show how the data set's most important actors interact with one another and how well they interact with the community.

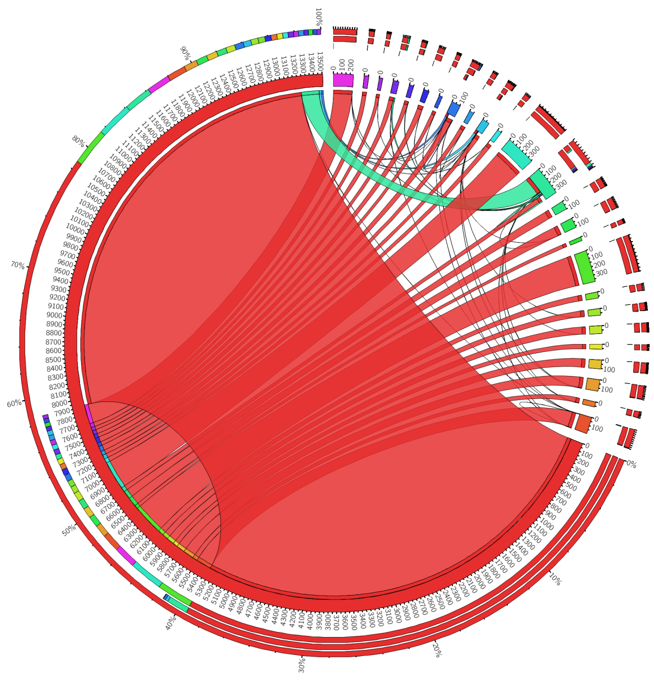


Figure 4.3: A circos visualization from a snapshot in our dataset

Figure 4.3 shows such a visualization from the same snapshot that was visualized in Figure 4.2. We have removed user names to maintain privacy. As a Circos visualization takes some explanation and such an explanation isn't relevant to the thesis, we will omit the intricacies of the figure and simply observe that it benefits from a display wall setting as there is an overall high-level view in the bands that criss-cross its diameter, and there is a lot of fine

5. http://circos.ca/intro/tabular_visualization/

detail in the little ticks and numbers that run along its circumference.

4.5 Virality predictor

The final visualization is very simple graphically and quite complex computationally. The visualization simply shows a random hashtag[6] out of a list of hashtags that are likely to go viral.[7] The computation of this list is the challenge, and we devote the rest of the visualization chapter to outline the process.

Our implementation of virality prediction follows recent work by Weng et al. [64], whose main contribution is a method for translating data about community structure into features for predictive knowledge about what info will spread widely.

4.5.1 Labeling

First it is useful to define rigorously what constitutes “viral.” Weng & al. propose the thresholds θ_T and θ_U: if a hashtag is mentioned in more tweets than θ_T% of the others, it is viral; if a hashtag is adopted by more users than θ_U% of the others, it is viral.

4.5.2 Features

The “community structure” mentioned above is the social network in the data set. This can either be follower–followee relationships or the graph induced by communication (i.e. @-messages). We have tried different variants of both, the results of which are provided in Chapter 7. Here we will limit ourselves to an overview of the features proposed by Weng & al. The features are generated from the first snapshot in which the hashtag occurs:

Number of early adopters: how many distinct users are observed using the tag.

N.o. uninfected neighbors: how many users are neighbors to early adopters, but not early adopters themselves.

N.o. infected communities: how many communities contain early adopters (see below for a description of community detection).

Usage entropy: let $r_c(h)$ be the fraction of tweets in community c that contain

6. **hashtag**, n.: a little label that users attach to their messages to indicate topic.
7. **viral** (of information diffusion), a.: of the nature of a virus, i.e. spreads widely rapidly.

hashtag h. The usage entropy $H^t(h)$ of h is defined as follows:

$$H^t(h) = -\sum_{c \in C} r_c(h) \log r_c(h)$$

Adoption entropy: similarly to usage entropy, let $g_c(h)$ be the fraction of users in community c that are early adopters of h:

$$H^u(h) = -\sum_{c \in C} g_c(h) \log g_c(h)$$

Fraction of intra-community interactions: How many out of all pairwise user interactions (@-messages or retweets) in a given community contain the tag.

4.5.3 Community detection

We rely on the Louvain fast community unfolding method of Blondel et al. [4] to detect communities for use in feature generation. The method is a heuristic approach that aims to minimize the *modularity* of a partitioning of the graph. Modularity is a measure for the density of edges within communities vs. that of edges between communities, for which see Newman and Girvan [48].

We use a python implementation of the Louvain method.[8]

4.5.4 Training and Classification

We do supervised learning. To generate training data, gather a twitter set over time and divide into snapshots as described in Section 4.1, gather a social network to provide community structure as described in Chapter 6. To generate labels, simply iterate over all the data and count hashtag uses and unique adopters, and label according to θ_U and θ_T. This results in a binary classification problem of "likely to go viral" or "unlikely to go viral." Features are generated as described above from the first snapshot in which a hashtag appears.

We use a Go-implemented random forest (see Breiman [7]) classifier.[9] Random forest is an ensemble classifier of multiple decision trees, where the idea behind ensemble classifiers is that many weak learners may make a strong one together. Each tree in the forest gets a vote on which class a feature belongs to, and the majority vote wins.

Decision trees comprise a large class of non-linear classifiers where classification is performed by means of several questions about the features of a

8. http://perso.crans.org/aynaud/communities/
9. https://github.com/sajari/random-forest

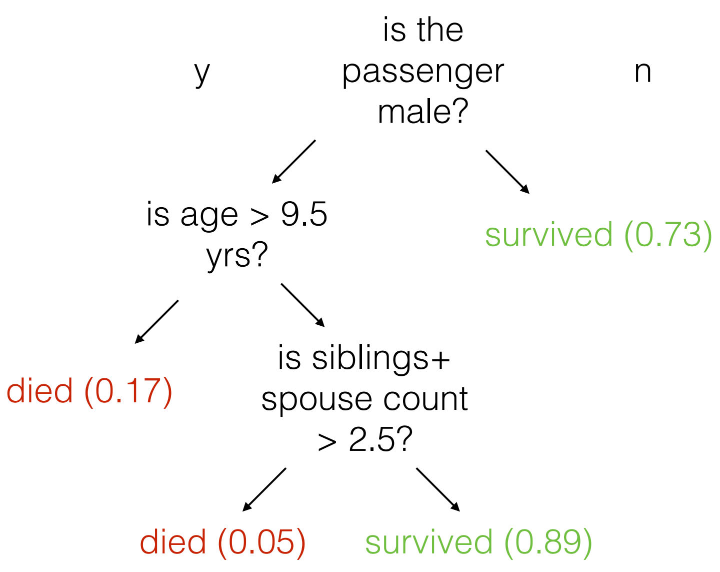


Figure 4.4: A decision tree

feature vector, such as "is $x_i \leq \alpha$?" where this threshold α will determine either which class the feature should be classified as, or whether more questions should be asked [61].

Figure 4.4 shows a decision tree over survival on the Titanic. The parenthesized numbers show survival probability.[10]

10. Figure adapted from http://en.wikipedia.org/wiki/Decision_tree_learning

5 System Implementation

This chapter describes our prototype implementation. The prototype consolidates different visualization programs and scripts — some our own, some by others — by means of various wrapper/mediator Python code.

5.1 Communication

Workers, Drawers, and the Master communicate by means of the rpcHugs python module from previous work of ours.[1] The module offers a simple remote procedure call interface where the user creates a dummy object and simply uses this as though it were a local object:

```
stub = RPC()
drawer = stub.getDummy((drawerIP, drawerPort))
drawer.draw(snapshotNumber)
```

1. http://github.com/transmetro/rpcHugs

5.2 Master

The prototype Master is a simple shell where the user can start and stop the system, and select a range of snapshots to visualize.

The system is initialized by launching a Worker and a Drawer on each tile on the Tromsø Large Display Wall. Once the Workers and Drawers are running, the master sets each Drawer's mode (see below) according to the arrangement described in Section 4.1.

The user enters visualization requests into the master's terminal command line interface such as

```
$ window 5
```

which will cause the first column of drawers to visualize snapshot 5, the next one to visualize snapshot 6, etc.

5.3 Drawers

A Drawer accepts two different remote calls: draw snapshot s, and change mode to m.

Drawers start up with the default *drawer mode* of heatmapping. This means that a Drawer will always return a heatmap visualization if this mode isn't changed. Four different modes correspond to the visualizations described in Chapter 4. If a Drawer is meant to draw something other than a heatmap, it must be explicitly set to different mode.

Drawers are programmed to know what work is required for a given visualization. They have a list of all the Workers from which they can request work. When a Drawer receives a visualization request, it locates a Worker in its Worker list by calculating the following index: let s be the requested snapshot, let m be the Drawer's current mode, and let w be the number of Workers:

$$i = s^m \bmod w$$

This ensures that (i) the same Worker is always responsible for generating m from s, which gives us cache utilization; (ii) a single Worker is unlikely to be responsible for all different visualization work related to m for the same s or vice versa, which would not be the case by simply $i = s \bmod w$, or $i = m \bmod w$, and which would cause workload hotspots; lastly, (iii), Worker lookup is quick and cheap.

5.3.1 Visualization hints

Drawers are by design independent of one another, but they do sometimes have to share information in the prototype. This is specifically a problem with the heatmaps: heatmaps drawn completely independently of one another may falsely appear to be of a similar size and density. To solve this, we allow the drawers to provide one another with hints about how large the largest heatmap generated thus far is and how dense the densest heatmap is. This allows the drawers to scale down and lighten the appearance of their heatmaps if bigger ones exist, which allows the user to relate heatmaps more easily.

The hints are delivered in a best-effort all-to-all manner within a row of similar visualizations. As it is difficult to imagine very long rows of many displays (i.e. enough that all-to-all communication would be punitively costly), this seems like a fair choice.

5.4 Workers

Workers, as mentioned in the design chapter, do most of the heavy lifting. They are responsible for co-ordinating the various data mining programs we use, and for performing any data transformations required. They keep all previously performed work cached so that partial results can be reused in other work and full results can be provided if the same visualization is requested later.

We set no restrictions about number of workers, there should at least be one. For simplicity we run one worker per machine in the Display Wall cluster.

5.5 Loaders

Loaders receive data and decide where best to place them. As our data set is fairly small, we simply duplicate it on all nodes in the display cluster to minimize communication-incurred latency.

The loader generates snapshots from a large pile of Twitter data by taking all the tweets that occurred within some time frame and placing them in the same file, this is done sequentially for all of the data and the snapshots stored by number:

```
/tmp/vis/data/0.twt
/tmp/vis/data/1.twt
    ...
/tmp/vis/data/n.twt
```

The data is only ever read, never modified. If a worker needs to generate a temporary file as part of some work (e.g. a table for the Circos visualization), this is created in a separate temporary directory.

6 Case Study Data Set

This chapter describes the data set we use as a case study in our work — how it was gathered and its general characteristics. The work and observations in this chapter largely build on previous work of ours [32], which in turn was based on work by Tor Kreutzer [37], who was probably the first to seriously consider how to systematically harvest Twitter data.

6.1 Tweet Anatomy

A tweet is a JSON object with many dimensions, we will only mention the fields we used for our analyses here. For a complete enumeration see Twitter's developer resources.[1]

tweet['text']: The tweeted message.
tweet['entities']['hashtags']: Hashtags used in the message.
tweet['entities']['user_mentions']: User mentions.
tweet['retweeted_status']['user']: The retweeted_status field only exists if the tweet is a retweet of someone else's message. This field provides information about the originator of the tweet.
tweet['user']: Information about the user who tweeted this message.

1. https://dev.twitter.com/docs/platform-objects/tweets

The user fields have several sub-fields, we are only interested in the *id* and *screen_name* fields.

6.2 Tweet Harvest

We gathered our dataset in the months leading up to the Norwegian 2013 parliamentary election through Twitter's stream API.[2] Our search parameters were largely party names and the names of prominent politicians, which see appendix A for the complete list.

The stream API provides at most a 1% sample of Twitter's total activity (they call this the streaming cap). If the volume of matching tweets is under the streaming cap, all activity will be returned. If tweets are dropped due to rate limiting, the stream provides messages informing the user about how many tweets were dropped. As such we can be certain that our gathered data is the complete set — we never received rate limit messages.

6.3 Data Set Volume

The data set consists of roughly 400 MB of JSON data; table 6.1 provides a more detailed view of the set's volume. For one example about the dimensionality this might entail, the graph induced by @-messages comprises the 19 K users as vertices and the 75 K @-messages as edges: user A's mention of user B results in the directed edge *(a, b)*.

Table 6.1: Quantifying the Twitter data set

Number of tweets	112 196
N.o. unique users	18 974
N.o. @-messages[a]	75 419
@-messages within data set	21 192
Reciprocal @-messages	1 739

[a] An @-message is a user mentioning another user.

While the set is somewhat sparse, it is real data of exactly the kind that an analyst can gather during election times, which is a real advantage.

2. https://dev.twitter.com/docs/api/streaming

6.4 Social Network Harvest

The Twitter stream doesn't provide anything other than raw tweet data. This means that if you're interested in the community structures that connect the users occurring in the stream, this must come from some other source. A few options are open:

i. Use the induced @-graph as described in Section 6.3.
ii. Use the graph that Kwak et al. [39] crawled in 2010.
iii. Gather the graph yourself through Twitter's REST API.[3]
iv. Apply for research access to Twitter's data.
v. Purchase access to Twitter's data.

Option (i) is tempting in its simplicity, but the graph will necessarily lack a lot of information. Option (ii) requires only a very little extra work over (i), but the graph is four years old at the time of writing and will necessarily lack a lot of information. Option (iii) will provide the full graph, but it is a painstaking and slow process of incrementally querying Twitter's REST API while adhering to their draconian rate limiting. Options (iv) and (v) will both provide the full graph, but (v) is costly and the terms that govern (iv) "read like they [i.e. Twitter] want to own [your] ideas" [5], which might deter researchers.

We have performed our experiments with i–iii, and we evaluate their coverage of our data set in section 7.4.2.

6.5 Words of Caution About Twitter, Big Data, and Sampling Bias

The analyst working on found data data sets — whether they be Twitter data, search trends, or otherwise — should proceed with caution. For e.g. has Morstatter et al. [46] recently shown that there is evidence of bias in how the Twitter 1% sample is chosen. They did this by purchasing access to the full 100% firehose[4] and comparing this with the 1% sample.

In early 2013 Nature reported how Google's much-lauded Flu Trends[5] had over-predicted flu levels by nearly two times that of US Center for Disease Control reports [9]. Study has shown that GFT has in fact consistently over-predicted for a long time [40], and that the fit of simple forward projection of already available CDC data is very nearly as good as that of GFT [25].

Our data set is necessarily biased: it only contains tweets that fall within

3. https://dev.twitter.com/docs/api/1.1
4. This is what they call it.
5. http://www.google.org/flutrends/

the search defined in appendix A. It is not, however, biased from Twitter’s end; as already mentioned in section 6.2, our harvest never hit the 1% streaming cap.

There are other caveats to be made and cautionary tales to be told about using found data, suffice it to say that we will be making no bold claims about the Norwegian electorate in this report.

7 Evaluation

The economist reported in October, 2013, how science's self-correction mechanism may not correct all that well;[1] negative results are seldom reported, and replication of results — though supposed to be the keystone in modern science — is not generally encouraged.

In the spirit of thoroughness and in light of the above, we will (in addition to our prototype) evaluate the classification system of Weng et al. [64] that we implemented as one of our visualizations. This will also show how the method works on a smaller, more specialized dataset.

7.1 Metrics

As specified in Chapter 3, the main requirement for our system is response time. We measure this through **roundtrip latency**, which we define as the time from the moment that a drawer receives a visualization request to its having updated the display with the requested visualization. This times the path from drawer to worker and back again. We measure roundtrip latency in seconds (s).

There are two main ways of evaluating a classification system: through error probability estimates or through measuring the system's tendency for confusion (of classes). We take the lead from Weng & al. [64] and measure

1. http://www.economist.com/news/briefing/21588057-scientists-think-science-self-correcting-alarming-degree-it-not-trouble

confusion by way of precision, accuracy, and recall.

A classifier's *confusion matrix* is the matrix $A = [A(i, j)]$ where the element $A(i, j)$ is the number of feature vectors whose true label was i that were classified as belonging to class j. As we have implemented a binary classifier (viral or not viral), we have a four-entry confusion matrix, where we will focus on the precision and recall with regard to positives (i.e. "viral"). Let class 1 be positives, and class 2 be negatives.

Precision: The fraction of all vectors classified as true that were true positives:

$$P = \frac{A(1,1)}{A(1,1) + A(2,1)}$$

Recall: The fraction of vectors with true class label positive that were classified as positive:

$$R = \frac{A(1,1)}{A(1,1) + A(1,2)}$$

Accuracy : The fraction of vectors that were correctly classified:

$$Ac = \frac{A(1,1) + A(2,2)}{A(1,1) + A(1,2) + A(2,2) + A(2,1)}$$

These measures can be expanded for more classes, they can also be calculated w.r.t. different classes.

Signal to noise–ratio is the fraction of a data set that carries useful information.

7.2 Experimental platform

We performed all experiments on the Tromsø Large Display Wall cluster, which comprises 28 HP Z400 Workstations with the following hardware configuration:

- Intel Xeon processor W3550, 3.06 GHz, 8 MB cache, 1066 MHz memory, 4.8GT/s QPI, Quad-Core, HT, Turbo
- 12 GB RAM (4 x 3) DDR3 1333 MHz
- Intel® X58 Express Chipset
- Integrated Broadcom 5764 LAN
- 1TB SATA Hitachi HDS72101
- 600 W power

All nodes in the cluster run Rocks 5.4 (Maverick), Linux version 2.6.18-194.17.4.el5.

7.3 Experiments

7.3.1 Roundtrip Latency

This experiment is to evaluate the responsiveness of our system. We performed the experiment by first simplifying our system by disabling the Master, and modifying the Drawers to automatically access the entire data set from Chapter 6 sequentially — in total 36 snapshots of varying size (see results). We time the path from Drawer to Worker and back again until the display is updated at the Drawer. We performed this experiment twice, once with caching enabled and once with caching disabled.

7.3.2 Social Network Coverage

This experiment measures the extent to which the different ways of collecting a social network allows us to create feature vectors from our data set. As laid out in section 6.4, there is a handful of ways to collect community structure about a set of Twitter users. We evaluate the use of (i) the induced communication graph ("@-graph"); (ii) the 2010 graph collected by Kwak et al. [39] ("Kwak2010"); and (iii) a graph gathered by requesting all friend lists of all users in our data set through the Twitter REST API ("following").

These are all directed graphs; we convert them to undirected graphs in two ways: by removing the direction of all edges ("full"), and by having undirected edges only where the original graph has an edge in both directions ("reciprocal").

The above gives us six graphs in total: the three different graphs in full and reciprocal forms. Now consider the set of all hashtags used in our data set. We wish to convert these to feature vectors by means of the community structure provided by the different graphs. The signal in this set are the hashtags where it is possible to find communal structure around the early adopters; our signal-to-noise ratio is the fraction of the hashtag set that can usefully be converted to feature vectors.

7.3.3 Social Network's Effect on Confusion Tendency

This is to evaluate the confusion tendency of the virality prediction classifier on our data set. We measure confusion tendency by means of 10-fold cross validation using each of the above graphs to generate feature vectors.

In k-fold cross validation, the training set is divided into k subsets (folds) of equal size. One of these is used as validation data (from which the confusion matrix is calculated) and the $k - 1$ others are used for training data. This is repeated k times so that each subset has been used once for validation data, after which the precision, etc. are averaged across the folds. As there is some randomness to how random forests [7] are generated, we perform 100 such tests for each graph.

7.4 Results

This section describes and discusses our experimental results. Our box-and-whisker plots are of the standard (sometimes called Tukey) style: the box encapsulates the data between Q_1 and Q_3, the crossbar shows the sample median, and the whiskers extend to the farthest points that are within 1.5 times the interquartile range $IQR = Q_3 - Q_1$ of the box.

We generate the box plots from $n = 2400$ samples per visualization per snapshot. We omit outliers (i.e. points that fall outside the whiskers) to avoid overplotting; the most extreme outliers fall somwehere close to 300s.

7.4.1 Roundtrip latency

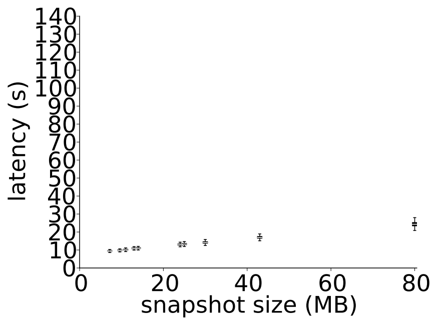
Uncached

Figure 7.1 shows the roundtrip latency of different visualizations for different sizes of snapshots in our data set, with caching disabled. The largest snapshot — the days around election night — is roughly 80 MB large.

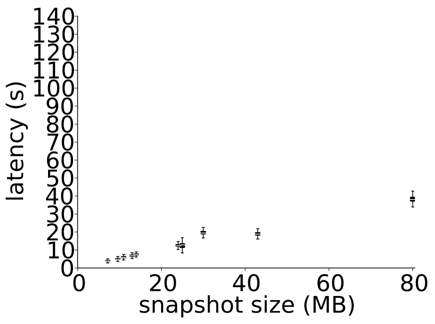
The most obvious result here is that on-demand computation of visualizations is out of the question: the latency for even small snapshots is unreasonable for all visualizations.

Virality prediction, shown in Sub-figure 7.1c, is the most computationally intensive and as such has the longest latencies. It is also the most variable in latency.

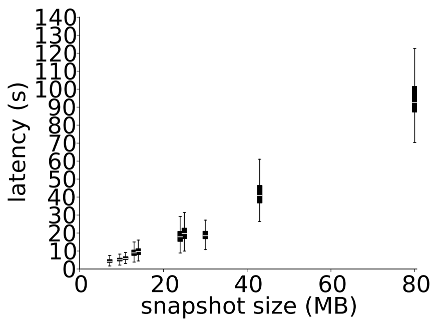
Wordcloud (Sub-figure 7.1d) has the tightest latency distributions, and doesn't seem to take longer time as snapshot size grows at this scale. A wordcloud is basically a word count, which can be performed with one pass of the data. The snapshots are mostly within the same order of magnitude in size, so it's not surprising that the wordcloud performs consistently.



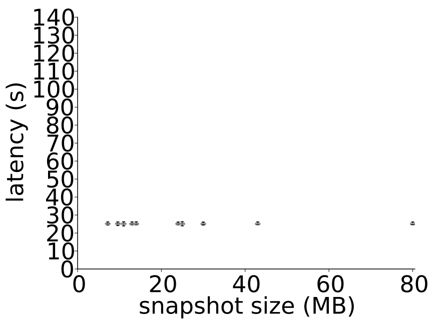
(a) Circos



(b) Heatmap



(c) Virality prediction



(d) Wordcloud

Figure 7.1: Latency experiments without caching.

Cached

Figure 7.2 shows the effect of caching. Caching allows sub-second delivery of all visualizations for all snapshot sizes.

The heatmap is the only visualization that takes noticeably longer to deliver for larger snapshots. This is simply because it is the only one whose worker-provided analytic grows with data size: wordcloud and Circos are sent as images, virality prediction is sent as a fixed-size list of likely hashtags. Heatmap requests the force-directed positions of all vertices in the @-graph. The logarithm-like growth of the heatmap latency suggests that the number of vertices in the @-graph doesn't grow as fast as data volume, which in turn suggests that the users already participating generate more content.

7.4.2 Social Network Coverage

Table 7.1 shows the signal-to-noise ratio of the different graphs enumerated in section 7.3.2 — i.e. how many of the 7842 hashtags that occur in our data set have to be thrown away because the feature vectors generated with a given

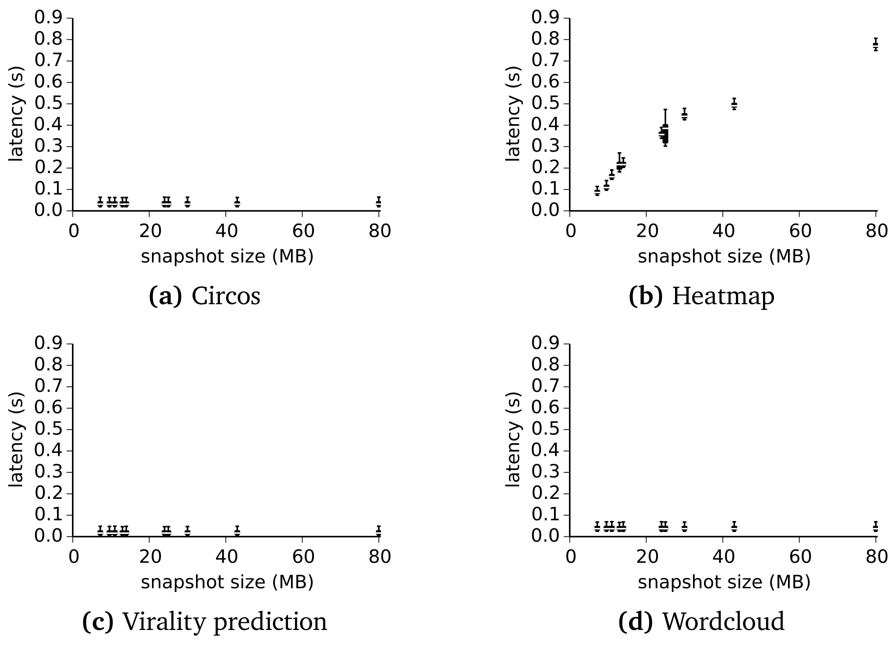


Figure 7.2: Latency experiments with caching.

TRAINING SET	SIGNAL TO NOISE
Following, full	0.99
@-graph, full	0.91
Following, reciprocal	0.90
@-graph, reciprocal	0.49
Kwak2010, full	0.27
Kwak2010, reciprocal	0.24

Table 7.1: Social network signal-to-noise ratios

graph turns out to be a handful of early adopters and then all zeroes: a direct result of there being no community structure around these early adopters.

The take-home message of table 7.1 is that the Kwak2010 graph, while valuable as a research tool, is impractical to use on a real-world dataset. Simply too many users have signed up since the graph was harvested.

The hand-harvested full follower–followee graph naturally has the best coverage. The reason there is still some data missing is that some users have privacy settings that prevents public access of their friend lists.

Reciprocal graphs generally have poorer coverage because of the asymmetrical nature of twitter: an @-message isn't a conversation, it is simply a direction that the message is intended to go, and there is no reciprocity tied to a user following another user.

7.4.3 Social Network's Effect on Confusion Tendency

These experiments are both classification system evaluations and social network experiments. They show how our implementation of the virality predictor works, but they also show how different social network graphs affect prediction.

The tables in this section show how the use of different social networks to create training data affects accuracy, precision, and recall in the virality prediction. We include two naive approaches that ignore features entirely — random choice: uniformly output either *true* or *false*; and prior probability-based choice: 90% of the time output *true*, 10% of the time output *false*.

This is where it is interesting to look at the reciprocal graphs; it's tempting to conjecture that reciprocal graphs would cause better predictions, as a reciprocal relationship reflects closer, more stable social connections [64].

It is important to keep in mind that the below tables have been computed on the necessarily curtailed training sets that result from the different graphs. Kwak2010-full, for example, had its precision, recall, and accuracy computed with the 27% of the original 7842 training vectors that weren't all-zeros.

TRAINING SET	MEAN	SD
Following, full	0.92	0.00028
@-graph, full	0.91	0.00030
Following, reciprocal	0.91	0.00033
@-graph, reciprocal	0.86	0.00060
Kwak2010, reciprocal	0.83	0.0018
Kwak2010, full	0.82	0.0012
Prior probability	0.82	N/A
Random Guess	0.50	N/A

Table 7.2: Classifier accuracy w.r.t. different graphs

As mentioned, we use a $\theta_U = 90\%$ threshold for virality, By definition 90% of all training vectors will be *false*, and 10% of all training vectors will be *true*. Prior probability–based classification (PP) will, based on this knowledge, have an accuracy of:

$$Ac_{PP} = \frac{0.1^2 + 0.9^2}{1} = 0.82$$

Random choice (RC) will have an accuracy of:

$$Ac_{RC} = \frac{0.5 \cdot 0.1 + 0.5 \cdot 0.9}{1} = 0.5$$

The full follower–followee graph works best, but only by a small margin over the full @-graph, which is much more readily available. Accuracy isn't a very good measure for how well the classifier performs, however. For an example, it is very simple to design a classifier with 90% accuracy in this case: always output *false*.

TRAINING SET	MEAN	SD
Kwak2010, reciprocal	0.89	0.0047
Kwak2010, full	0.89	0.0061
Following, reciprocal	0.88	0.0075
Following, full	0.87	0.0075
@-graph, full	0.86	0.0056
@-graph, reciprocal	0.86	0.0072
Random choice	0.10	N/A
Prior probability	0.10	N/A

Table 7.3: Classifier precision w.r.t. different graphs

Table 7.3 shows the precision of virality prediction in terms of different graphs used for training data. We compute the precision of our naive approaches as follows:

$$P_{PP} = \frac{0.1^2}{0.1^2 + 0.9 \cdot 0.1} = 0.1$$
$$P_{RC} = \frac{0.5 \cdot 0.1}{0.5 \cdot 0.1 + 0.5 \cdot 0.9} = 0.1$$

So while PP performed tolerable in terms of accuracy, 90% of the positives it returns will be false positives. The Kwak2010 graphs performs the best, but only by a small margin, and only on a small fraction of the original training set.

TRAINING SET	MEAN	SD
Kwak2010, reciprocal	0.52	0.0047
Random choice	0.50	N/A
Kwak2010, full	0.47	0.0033
@-graph, reciprocal	0.31	0.0022
Following, reciprocal	0.26	0.0018
@-graph, full	0.26	0.0023
Following, full	0.25	0.0019
Prior probability	0.10	N/A

Table 7.4: Classifier recall w.r.t. different graphs

Table 7.4 shows the recall of virality prediction w.r.t. the different graphs. We compute the recall of our naive approaches as follows:

$$\begin{aligned} P_{PP} &= \frac{0.1^2}{0.1^2 + 0.1 \cdot 0.9} = 0.1 \\ P_{RC} &= \frac{0.5 \cdot 0.1}{0.5 \cdot 0.1 + 0.5 \cdot 0.1} = 0.5 \end{aligned}$$

Only the training set created from the reciprocal Kwak2010 graph gets over the seemingly low bar of RC.

8 Infinite Graph Partitioning

Streaming graph computation is a new and exciting field, and we would like to move our system in the streaming direction. We have noticed that there is no comparable system to STINGER [16], a multicore server type system, in cluster computing; the closest contender is perhaps Microsoft's Naiad [47], which enables the embedding of iterative steps into a dataflow pipeline. This chapter describes our first steps towards a streaming graph system for clusters. We identify a novel graph partitioning problem, and propose and evaluate a solution to this problem.

8.1 Partitioning Revisited

A first, important step towards a cluster streaming graph system is that of loading the graph and partitioning the data. The state-of-the-art graph processing systems mostly ignore graph structure and randomly assign vertices to computers by a fast, random hash partitioning. As discussed in Section 2.5, a random partitioning will result in a very large edge cut and, as a consequence, cause a punitive communication overhead.

To sidestep the NP-completeness of optimal graph partitioning, heuristic methods have been developed. A recent paper by Stanton and Kliot [57] presents a thorough comparison of online partitioning heuristics that partition the graph as it is loaded onto the cluster. Their model is that the graph arrives as a stream of edge lists: a vertex arrives along with all its edges.

8.2 Problem Description

The streaming model we're interested in is the edge tuple model of STINGER. The edge tuple model sees graph as an infinite stream of edge insertions, updates, and deletions. The stream could be ordered any way at all, and there is no hint about the topology of the graph. This makes a partitioning based on a priori knowledge very difficult, as we describe in more detail below.

We initially assumed that the heuristics of Stanton and Kliot could be adapted directly to a massive streaming setting, but early experiments with their best heuristics produced very poor partitions. This is due to their model; in an edge tuple stream situation (as opposed to edge list), the first time a vertex appears in the stream, which is when it needs to be placed in a bin, exactly one neighbor is known — the other vertex in the edge tuple. Hence it is useless to assign the vertex to a bin based on neighbor-counting heuristics.

The problem is broader than the lack of useful neighbor information: there is no a priori way of knowing how large the graph is either, so it is difficult to know how many partitions are needed. Ideally a graph would be partitioned only if it is large enough that a single machine can't fit it in memory. Twitter choose to run their Who To Follow service in such a way that the graph computation can be performed out of memory on one fat server [27].

There are three implied levels of a priori graph knowledge in this chapter so far. First, with full knowledge of the entire graph, any graph partitioning scheme can be used; second, with the one-pass edge list streaming model of Stanton and Kliot — a much stricter model — the several neighbor-counting heuristics that they propose and evaluate can be used; third, and stricter still, is the no-knowledge level of edge tuple streams. This is to our knowledge unexplored territory, and it is what we focus on here:

> How to partition an infinite and unknowable graph such that (i) the cut is usefully smaller than that of random partitioning, and (ii) the throughput of the partitioner is not too far from that of random partitioning.

8.3 A Solution

We begin with some design considerations:

- It is neither necessary nor desirable to partition a graph that can fit in the memory of a single machine.
- The expected cut size of random partitioning is $1 - \frac{1}{k}$, where k is the number of partitions. If two or three partitions is enough, random may be good enough. E.g., in natural graphs, where there is no cheap cut, a

cut of 50%–60% of all edges may be tolerably close to the best offline heuristics. For large values of k, the cut will be massive.
- Randomly assigning some number of vertices to bins that already contain large, good partitions will probably not affect the size of the edge cut significantly.
- A partitioner should support several loaders.

The above suggests that if there is no knowledge about a vertex, it is just as well to do optimistic, best-effort partitioning. As edge tuples accumulate into a larger graph, however, there will be more information available, and it will be possible to make better partitioning decisions.

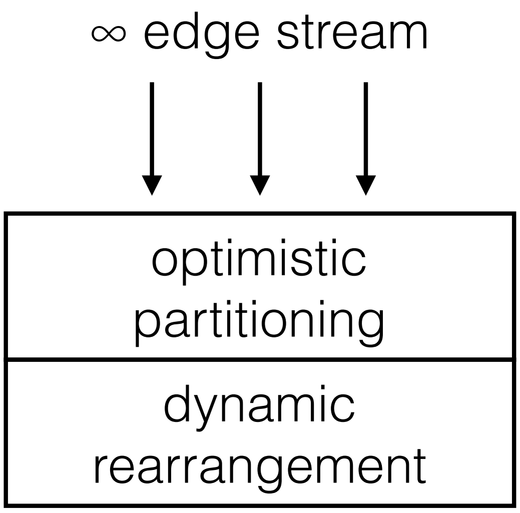


Figure 8.1: Graph partitioner high-level view: optimistic initial partitioning, dynamic rearrangement of partitions as knowledge accumulates.

Our solution is similar to array resizing schemes. When using an array as a dynamic data structure (e.g., when implementing a linked list ADT), it is usual to start with a conservatively small array and then resize it if a larger array is needed. This is done by first allocating a larger array, then copying the data from the small array into the larger one and swapping pointers.

Hence, when working with an infinite edge stream, start with a conservatively small number of workers — one, for instance — and repartition once it is apparent that more machines are needed. And as nothing is known about the node apart from a single neighbor, it is just as well to randomly assign it. On repartitioning, more will be known about vertices, and more informed partitioning can be performed.

Figure 8.2 illustrates this: an optimal 2-partition is shown in red and black. Inactive workers are shaded grey. In step (i), there is not enough graph to warrant more than one worker, and the edge cut is exactly $\emptyset$. In step (ii),

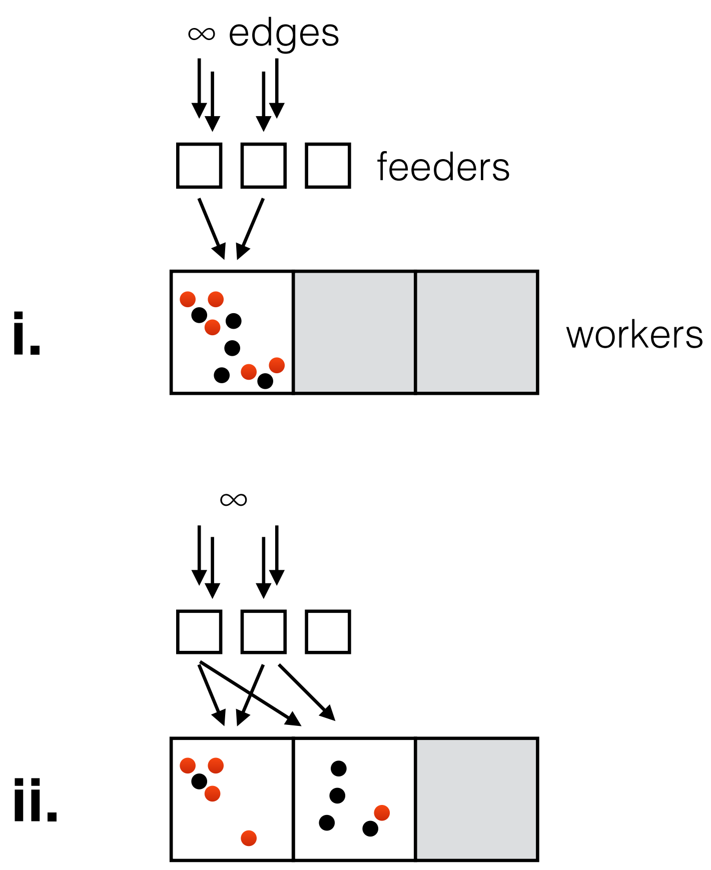


Figure 8.2: A possible infinite graph partitioning scheme: (i) shows one worker in commission, (ii) shows another brought in and the graph repartitioned.

another worker has been commissioned, and the graph has been split into a tolerable 2-partition.

If the graph shrinks sufficiently (through edge deletions) that it is unreasonable to keep all workers in commission, the worker with the smallest load (i.e. the fewest vertices) is decommissioned and its load is distributed to the remaining workers according to the repartitioning heuristic.

8.4 Evaluation

Time constraints prevented us from creating a functioning prototype, but we have performed offline simulations to show that our idea has merit.

Let E be the set of edges in a graph, and let $C \subset E$ be the set of edges that span partitions (the cut). We measure cut severity by $\frac{|C|}{|E|}$.

We create test graphs with a Kronecker generator [41] that we have implemented for an earlier project.[1] The reason we use these synthetic graphs is that the Kronecker generator can make graphs of arbitrary sizes, and it simulates power law edge distribution fairly well, which is the property that makes real life graph partitioning so difficult. Also, the Graph500 benchmarks for graph supercomputing[2] use a Kronecker generator, so it has potential for becoming a standard benchmarking tool with consistent and well-understood properties.

The Kronecker generator takes two parameters *scale* and *edge factor*, and will use these to create 2^{scale} vertices with an average neighbor count equal to the edge factor. For our experiment, we use a scale of 19 and an edge factor of 15. The generator produces edge tuples one by one in a random ordering, and as such is a perfect simulation of an edge stream.

Our baseline is random hashing—it's the only real contender. We use Stanton and Kliot's Linear Deterministic Greedy (LDG) partitioner to repartition on commissioning a new worker: let k be the number of available bins; let $P^t(i)$ refer to bin i at time t; let v be the vertex we wish to assign to a bin, and $\Gamma^t(v)$ be all of its known neighbors at time t. Finally, let c be the capacity constraint of a bin (i.e. how many nodes can the worker hold in memory). The index of the bin to which v should be assigned is determined by

$$ind = \arg\max_{i \in [k]} \left[|P^t(i) \cap \Gamma^t(v)| w(t,i) \right],$$

where $w(t,i) = 1 - \frac{|P^t(i)|}{c}$ is a linear weight function to encourage the assignment of vertices to bins with lower load. In other words: assign v to the bin in which it currenly has the most neighbors, weighted by this linear weight function.

Figure 8.3 shows our offline evaluation with partitioning badness as a function of received vertices in the edge stream. We use the repartitioning-at-need scheme described above in Section 8.3. The black line shows repartitioning by random hashing and the red line shows repartitioning by LDG. We have set the capacity constraint to $\frac{2^{19}}{10} \approx 52000$ to simulate ten available workers. The abrupt steps are when new workers are commissioned and the graph repartitioned.

Our evaluation confirms all of our conjectures: (i) a single worker has an empty edge cut, (ii) the second and third worker steps are comparable to hash partitioning, and (iii) adding random-partitioned vertices to already cleaned-up partitions does not cause a very steep rise in partition badness.

Point (i) is self-evident. Point (ii) stems from the observation of the expected cut size of a random cut, which is tolerable for few partitions. Point (iii) can be seen in a signal-to-noise light: if the signal is very strong, a comparatively small amount of noise won't make that much of a difference (i.e. won't push the signal-to-noise ratio very far).

1. http://github.com/transmetro/krongen
2. http://www.graph500.org/

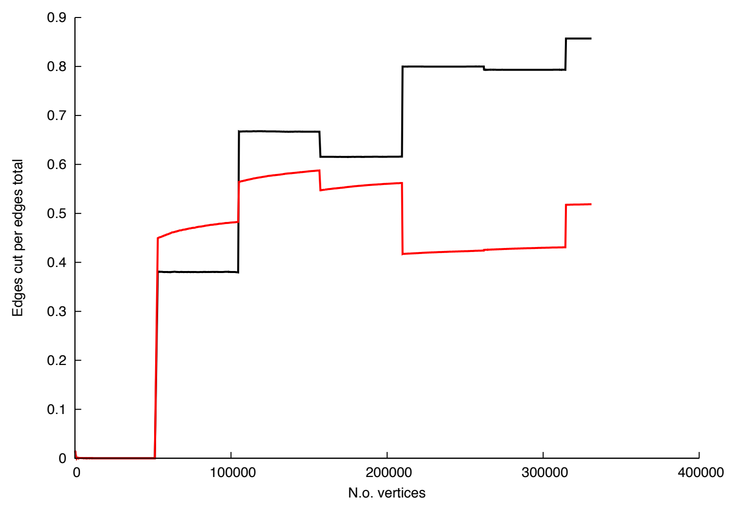


Figure 8.3: Offline simulation of the proposed solution to measure the edge cut of pure hashing (black) vs. hashing with rebalancing (red).

As a final remark, the use of random partitioning at load time makes the task of node assignment embarrassingly parallel, so that arbitrarily many loaders can work in parallel. This is useful for large-scale systems; it's impractical to crawl the web from a single machine.

9 Discussion and Related Work

9.1 Our System as a Whole

We have built and evaluated a system for exploratory analysis through visualization. Visual analysis, as all analysis, could either be hypothesis-based or exploratory. Hypothesis-based analysis is in the safe realm of Popper's "falsifiability, or refutability, or testability" [49], where the researcher has a specific hypothesis, and seeks to either confirm or refute it. Exploratory analysis is on shakier philosophical ground; the approach is more about having a data set known to be in some way relevant, but letting the data suggest hypotheses to test. This is necessarily more risky (and indeed receives fewer research grants than does hypothesis testing [29]). We harvest data from Twitter and visualize it without really knowing that there is anything about the data to recommend its exploration; hence we fall within the more extreme realms of exploratory analysis.

Our experiments show that it is in most cases strictly necessary to do either partial or full pre-computation of visualizations to provide the responsiveness required for the scattergun-approach to visualization. This is especially clear in the cases where we deal with graph computation, which is notoriously complex and difficult to parallelize (cf. Section 2.5). As we have already mentioned, the aesthetics of graph drawing are mostly NP-hard [3], and although we use some of the fastest algorithms available for graph layouts [34] and community detection [4], we are unable to provide visualizations on-demand at anything near acceptable times without pre-computation and caching, even at the small scale that we work with here. However, a display wall is an ideal setting for such problems, as it has not only the resolution to display visualizations at scale, but also the compute power to perform the necessary pre-computation.

It is clear that we either have to refine our implementation to pre-compute more aggressively (i.e. at all), or change our design to either (i) employ a pipelining scheme such as that of Naiad [47], or (ii) go full streaming as discussed below. This latter point (ii) would necessitate serious novel work in both systems and algorithms.

Display walls have only recently reached the point where people are starting to figure out how and why they are useful. Ruddle et al. [53] provide visual comparative genomics analysis of copy number variation. Their work can be said to leverage the full potential of display walls (in their words "WHirDs") in that they use the compute power as well as the screen real-estate that a display wall provides. Fjukstad et al. [20] provide high-resolution weather forecast visualization, where the display wall is used purely as a very large display; Hibbs et al. [31] provide comparative analysis of microarray datasets on a display wall. These systems provide either a comparison of several data sets or a large-scale view of one data set. Our system provides many smaller parallel views of the same data set.

The commercial business intelligence software Tableau[1] has a similar "multiple views of the data" philosophy to ours, and can suggest visualizations for your data. In this sense, our system is similar to having many Tableaux showing different things.

As a final note, the Norwegian Broadcasting Company (NRK) used both a display wall and interaction spaces in their 2013 election night coverage.[2]

9.2 Towards Streaming: Infinite Partitioning

We have identified the infinite edge stream partitioning problem and provided what is to our knowledge the only work on this exact problem. We propose and evaluate a partitioner for the infinite edge stream that compares favorably to its only real contender: the industry standard of random hash partitioning.

The field of graph partitioning is too broad to provide anything near a complete overview, we will restrict ourselves to observing that other work either ignores graph structure entirely [43, 44], or works with different assumptions and models than we do [26, 57].

1. http://www.tableausoftware.com/
2. http://nrkbeta.no/2013/08/08/spillteknologi-i-nrks-valgstudio/

9.3 Virality prediction

Our virality prediction system is an implementation of the work of Weng et al. [64] and as such provides nothing new in itself. It is however an application of the method to a specific real-world data set, and as such an interesting exercise.

We kept to a $\theta_U = 90\%$ threshold (as opposed to several thresholds) for simplicity. Weng & al. report the following $\theta_U = 90\%$ numbers:

Precision	0.62
Recall	0.42

Table 9.1: Results reported in Weng et al.

We achieved vastly better precision in all our cases and better recall in two cases — cf. tables 7.3 and 7.4. Our data set is much smaller than theirs ($\sim$ 7800 vs. $\sim$ 10 000 000) and more specialized than theirs (Norwegian politics vs the random sample provided by Twitter). We conjecture that our results are due to the specialized nature of the data set, but we can't rule out over-fitting. We use the 10-fold cross-validation of the original authors.

It is unclear how Weng & al. gather their social network (the raw tweet data were gathered from the Twitter random sample stream), or how heavy their graph computations were from a systems perspective — we were forced to use only the immediate social network around the users in our streamed data (a one-step breadth first search) to make the computations faster.

The main result that came out of our investigations of virality prediction and social networks was the impracticability of using the Kwak2010 data set for recent Twitter data, as it only covers a small part of the users that are likely to tweet today.

The work of Weng & al. is very similar to that of Romero et al. [52]; the two papers came out around the same time (2013) and seem to have been conceived and written in parallel with each other. Romero et al. use a simpler predictive model. Cheng et al. [10] published a 2014 paper on predicting the growth of an information cascade, but rather than specifying a binary viral/not viral type of classifier, they look at the prediction of whether a cascade will double in size, which seems like a more sophisticated and realistic approach.

9.4 Our Prototype and Case Study

We have held several informal demonstrations of our system for armchair analysts, this section outlines reactions to- and experiences with the system and the data set. Figure 9.1 shows an excited audience.

Figure 9.1: Armchair analyst.

First, the interface is cumbersome. Using a terminal to interface with a display wall that has the interfaces described in Section 3.3 feels unnatural. It is not immediately apparent to people that the visualizations are arranged in a time series ("needs labels"), and people would like the virality predictor to show several candidates instead of the single one it shows now ("could the hash tag predictor also be a wordcloud?").

On the other hand the system is very responsive when caches warm, as our experimental evaluation also shows, and people enjoy digging around in the different visualizations.

Figure 9.2 shows the heatmap of the communication graph in the weeks around the election (a snapshot is three days). We see activity increasing, peaking at the snapshot that contains election night, and dropping off slightly after election night — these latter also showing signs of community clustering that might warrant closer investigation. A problem with the heatmaps is that early in the election run-up, where activity was orders of magnitude lower, the heatmaps look very small and sparse as a result of having been scaled relative to the densest periods. Finally the maps could benefit from some clearer indication of original graph size in terms of $|V|$ and $|E|$.

The Circos visualizations ended up showing a surprising analytic. We made the circos plot so that it would show the 25 most active users of the communication graph, their interactions among one another, and their interactions with the community (somewhat arbitrarily defined as "the users that aren't the 25 most active"). We had expected the top communicators to be catch-all political party accounts, media outlet accounts, and similar, but one account that was consistently in the top 25 (see figure 9.3) was simply a private citizen. What's more, this user was consistently the most active in interfacing with the

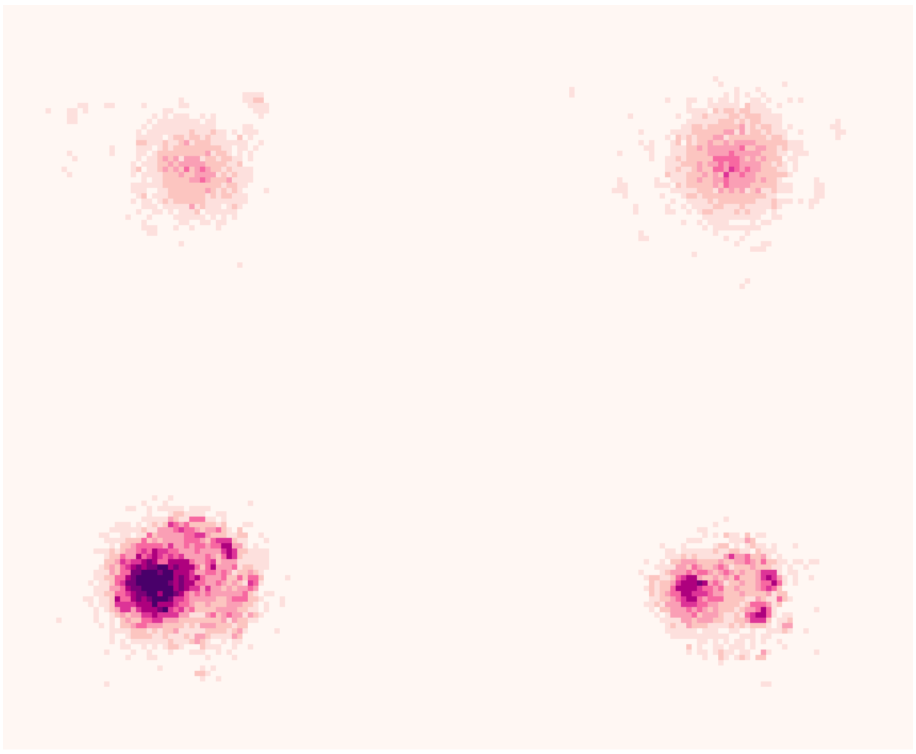


Figure 9.2: The time around election week

community (indicated by the greenish band that crosses over toward the huge red field off-figure, and its being much thicker than the other non-red bands). This top player in Twitter politics describes herself as “mother of two and a supporter of the Progress Party”. The other accounts in the top 25 seem to communicate mostly among themselves. This is the kind of storytelling, we imagine, that a tabloid journalist would love.

Figure 9.4 shows wordclouds generated from the tweets in the election night snapshot and the three snapshots following election night. Common and central to all is “frp”: the acronym for the Progress Party. This isn’t particular to these snapshots, it’s a trend throughout the data set. We conjecture that this is because the Norwegian Progress Party, if not universally well-liked, universally stirs emotions. More interesting are the words frequently occurring post election night such as “throwback,” “class inequality,” “right-wing populism,” etc.

Finally, the wordclouds provided a handy way to check whether the virality predictor was doing its job: if a hashtag is projected as potentially viral and later occurs in a word cloud, this should be counted as a success.

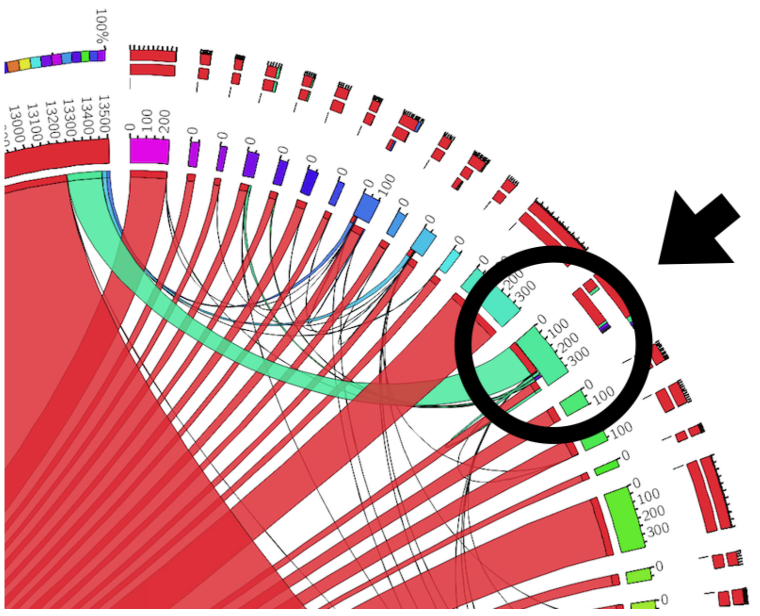


Figure 9.3: A mom that is very interested in progress

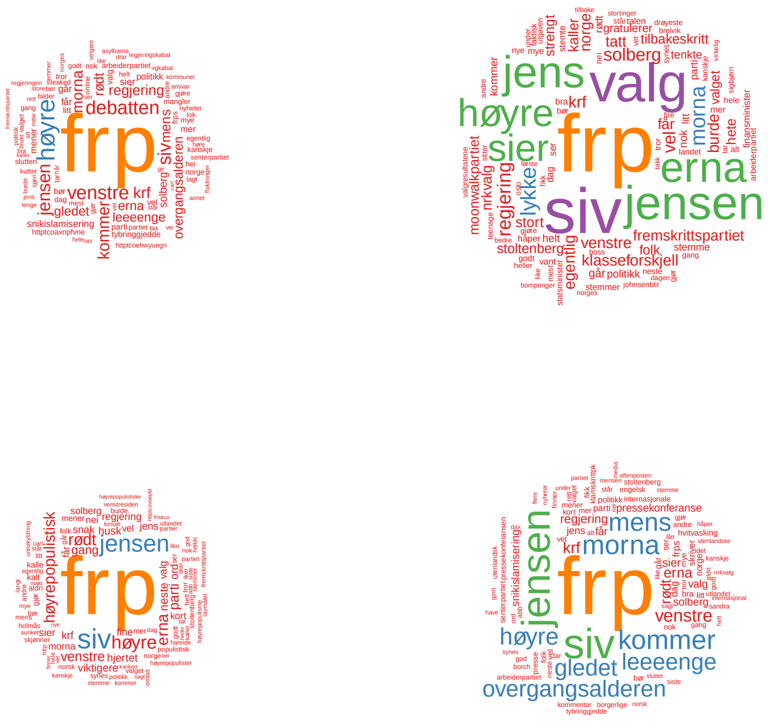


Figure 9.4: Post-election disillusionment

10 Concluding Remarks

This work has been a study in social network analysis in general, and Twitter visual analysis in particular. Our thesis that display wall architectures are well-suited to multi-faceted visualization and storytelling largely holds up. The evaluation shows that it is strictly necessary to use clever pre-computation, or pipelining, or streaming to meet the strict latency requirements of providing visualization interactively fast. In general, what seems up front to be a straightforward task often isn't: creating visualizations for a data set comprising about 400 MB of data — a truly tiny data set by today's standards — might incur worst-case latencies of hundreds of seconds.

Our contributions:

- We have designed, built, and evaluated a prototype for the scattergun approach to social network visualization, providing insights into how such a system should be realized (Chapters 3, 5, 7, and 9).
- We have found that our prototype does indeed uncover interesting stories in our case study data set (Section 9.4).
- We provide an overview of the different ways of gathering social media (Chapter 6) data and evaluate how these impact state-of-the-art trend prediction (Section 7.4).
- We identify the infinite edge stream partitioning problem. We propose and evaluate a solution to this problem (Chapter 8), showing that it compares favorably with the state of the art.

10.1 Future Work

On the visualization side, our prototype could benefit from more meta-information. Instead of scaling all heatmaps relative to the densest and largest among them, it would be a good idea to do relative scaling within orders of magnitude, and color-code different orders of magnitude. To more clearly show that visualizations are shown in a time-series, the snapshot time range should be displayed somewhere. Graph sizes in terms of number of vertices and edges would also be handy.

For smoother interaction, the prototype should be hooked up to the interaction spaces that the Tromsø Large Display Wall offers to replace the cumbersome command line interface. It would also be interesting to provide several levels of detail, for e.g. that the user could select a particular visualization for a particular snapshot and blow it up on the entire wall, or show a finer-grained view. This especially applies to the heatmaps, which hide a lot of information.

The prototype, though in many ways quite unstable, works well enough that it would be beneficial to present it to experts such as social scientists or journalists to get some concrete feedback on what would be a sensible direction to expand the system.

The system as it is only works on small scales, we should either branch out in streaming, and build a working prototype with functionality similar to that of STINGER [2]; or use clever pipelining schemes similar to those provided in Naiad [47].

Finally, we would like to expand our trend prediction evaluation and look at feature selection experiments. Weng et al. [64] indirectly provide a feature selection conjecture in observing that viral cascades seem to spread as simple contagions, which suggests that the features we use may be overkill, and perhaps even that we only need the simplest features such as “number of early adopters” and “number of uninfected neighbors.” We also believe that our system would be well-suited for online learning instead of the offline learning we do now.

Bibliography

[1] O Anshus, Daniel Stødle, T Hagen, Bård Fjukstad, J Bjørndalen, L Bongo, Yong Liu, and Lars Tiede. Nine years of the tromsø display wall, 2013.

[2] David A Bader, Jonathan Berry, Adam Amos-Binks, Daniel Chavarría-Miranda, Charles Hastings, Kamesh Madduri, and Steven C Poulos. Stinger: Spatio-temporal interaction networks and graphs (sting) extensible representation. *Georgia Institute of Technology, Tech. Rep*, 2009.

[3] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry*, 4(5):235–282, 1994.

[4] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[5] John Bohannon. Twitter offers entire data pool, but some wary of diving in. *Science*, 343(6174):958, 2014.

[6] Stephen P. Borgatti, Ajay Mehra, Daniel J. Brass, and Giuseppe Labianca. Network analysis in the social sciences. *Science*, 323(5916):892–895, 2009.

[7] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[8] Samuel F Bunting and Andre Nussenzweig. End-joining, translocations and cancer. *Nature Reviews Cancer*, 13(7):443–454, 2013.

[9] Declan Butler. When google got flu wrong. *Nature*, 494(7436):155, 2013.

[10] Justin Cheng, Lada Adamic, P Alex Dow, Jon Michael Kleinberg, and Jure Leskovec. Can cascades be predicted? In *Proceedings of the 23rd international conference on World wide web*, pages 925–936. International World Wide Web Conferences Steering Committee, 2014.

[11] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew. Deep learning with cots hpc systems. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1337–1345, 2013.

[12] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V Le, Mark Z Mao, Marc'Aurelio Ranzato, Andrew W Senior, Paul A Tucker, et al. Large scale distributed deep networks. In *NIPS*, pages 1232–1240, 2012.

[13] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[14] Edsger W Dijkstra. Letters to the editor: go to statement considered

harmful. *Communications of the ACM*, 11(3):147–148, 1968.

[15] Peter Eades. A heuristics for graph drawing. *Congressus numerantium*, 42:146–160, 1984.

[16] David Ediger, Robert McColl, Jason Riedy, and David A Bader. Stinger: High performance data structure for streaming graphs. In *High Performance Extreme Computing (HPEC), 2012 IEEE Conference on*, pages 1–5. IEEE, 2012.

[17] David Ediger, Jason Riedy, David A Bader, and Henning Meyerhenke. Tracking structure of streaming social networks. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 1691–1699. IEEE, 2011.

[18] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37, 1996.

[19] Miroslav Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25(4):619–633, 1975.

[20] B Fjukstad, O Anshus, and J Bjørndalen. High resolution numerical models on a display wall. In *The 7th Annual Meeting of the European Meteorological Society (EMS) and the 8th European Conference on Applications of Meteorology*. Citeseer, 2007.

[21] Bjørn Fjukstad. NOWAC Data Exploration. http://bdps.cs.uit.no/papers/capstone-bjorn.pdf, 2013.

[22] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[23] Michael R Garey, David S. Johnson, and Larry Stockmeyer. Some simplified NP-complete graph problems. *Theoretical computer science*, 1(3):237–267, 1976.

[24] Jim Giles. Making the links. *Nature*, 488(7412):448–450, 2012.

[25] Sharad Goel, Jake M Hofman, Sébastien Lahaie, David M Pennock, and Duncan J Watts. Predicting consumer behavior with web search. *Proceedings of the National Academy of Sciences*, 107(41):17486–17490, 2010.

[26] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 17–30, 2012.

[27] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. Wtf: The who to follow service at twitter. In *Proceedings of the 22nd international conference on World Wide Web*, pages 505–514. International World Wide Web Conferences Steering Committee, 2013.

[28] Tor-Magne Stien Hagen. Interactive visualization on high-resolution tiled display walls with network accessible compute-and display-resources. 2011.

[29] Chris Haufe. Why do funding agencies favor hypothesis testing? *Studies in History and Philosophy of Science Part A*, 44(3):363–374, 2013.

[30] Susan Havre, Beth Hetzler, and Lucy Nowell. Themeriver: Visualizing theme changes over time. In *Information Visualization, 2000. InfoVis 2000. IEEE Symposium on*, pages 115–123. IEEE, 2000.

[31] Matthew Hibbs, Grant Wallace, Maitreya Dunham, Kai Li, and Olga Troyanskaya. Viewing the larger context of genomic data through horizontal integration. In *Information Visualization, 2007. IV'07. 11th International Conference*, pages 326–334. IEEE, 2007.

[32] Einar J Holsbø, Phuong H Ha, and Otto J Anshus. The big digger & puzzler system for harvesting & analyzing data from social networks. *Norsk informatikkonferanse (NIK)*, 2013, 2014.

[33] John Hopcroft and Robert Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June 1973.

[34] Yifan Hu. Efficient, high-quality force-directed graph drawing. *Mathematica Journal*, 10(1):37–71, 2005.

[35] Owen Kaser and Daniel Lemire. Tag-cloud drawing: Algorithms for cloud visualization. *arXiv preprint cs/0703109*, 2007.

[36] Donald E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing*. ACM, New York, NY, USA, 1993.

[37] Tor Kreutzer. Harvest: a collaborative system for distributed retrieval of social data. 2012.

[38] Martin I Krzywinski, Jacqueline E Schein, Inanc Birol, Joseph Connors, Randy Gascoyne, Doug Horsman, Steven J Jones, and Marco A Marra. Circos: An information aesthetic for comparative genomics. *Genome Research*, 2009.

[39] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, pages 591–600. ACM, 2010.

[40] David Lazer, Ryan Kennedy, Gary King, and Alessandro Vespignani. The parable of google flu: Traps in big data analysis. *Science*, 343(6176):1203–1205, 2014.

[41] Jurij Leskovec, Deepayan Chakrabarti, Jon Kleinberg, and Christos Faloutsos. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In *Knowledge Discovery in Databases: PKDD 2005*, pages 133–145. Springer, 2005.

[42] Kai Li, Han Chen, Yuqun Chen, Douglas W Clark, Perry Cook, Stefanos Damianakis, Georg Essl, Adam Finkelstein, Thomas Funkhouser, Timothy Housel, et al. Building and using a scalable display wall system. *Computer Graphics and Applications, IEEE*, 20(4):29–37, 2000.

[43] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727, 2012.

[44] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

[45] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, pages 415–444, 2001.

[46] Fred Morstatter, Jürgen Pfeffer, Huan Liu, and Kathleen M Carley. Is the sample good enough? comparing data from twitter's streaming api with twitter's firehose. *Proceedings of ICWSM*, 2013.

[47] Derek G Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martin Abadi. Naiad: a timely dataflow system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 439–455. ACM, 2013.

[48] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69:026113, Feb 2004.

[49] Karl Popper. Conjectures and refutations. the growth of scientific knowledge. london and new york, 1963.

[50] Helen C Purchase, Robert F Cohen, and Murray James. Validating graph drawing aesthetics. In *Graph Drawing*, pages 435–446. Springer, 1996.

[51] N Quinn and Melvin A Breuer. A forced directed component placement procedure for printed circuit boards. *Circuits and Systems, IEEE Transactions on*, 26(6):377–388, 1979.

[52] Daniel M Romero, Chenhao Tan, and Johan Ugander. On the interplay between social and topical structure. In *Proc. AAAI Intl. Conf. on Weblogs and Social Media*, 2013.

[53] Roy A Ruddle, Waleed Fateen, Darren Treanor, Peter Sondergeld, and Phil Ouirke. Leveraging wall-sized high-resolution displays for comparative genomics analyses of copy number variation. In *Biological Data Visualization (BioVis), 2013 IEEE Symposium on*, pages 89–96. IEEE, 2013.

[54] Robert Sedgewick. Algorithms in c, part 5: Graph algorithms, 2002.

[55] Dafna Shahaf, Carlos Guestrin, and Eric Horvitz. Metro maps of science. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1122–1130. ACM, 2012.

[56] Marc A Smith, Lee Rainie, Itai Himelboim, and Ben Shneiderman. Mapping Twitter Topic Networks: From Polarized Crowds to Community Clusters. *The Pew Research Center*, pages 1–57, 2014.

[57] Isabelle Stanton and Gabriel Kliot. Streaming graph partitioning for large distributed graphs. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1222–1230. ACM, 2012.

[58] G. Strang. *Linear Algebra and Its Applications*. Thomson, Brooks/Cole, 2006.

[59] Daniel Stødle. *Device-Free Interaction and Cross-Platform Pixel Based Output to Display Walls*. PhD thesis, Ph. d. thesis, Uni. of Tromsø, 2009.

[60] Daniel Stødle, Olga Troyanskaya, Kai Li, and Otto J Anshus. Tech-note: Device-free interaction spaces. In *3D User Interfaces, 2009. 3DUI 2009. IEEE Symposium on*, pages 39–42. IEEE, 2009.

[61] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Fourth Edition*. Academic Press, 4th edition, 2008.

[62] Jeffrey Travers and Stanley Milgram. An experimental study of the small world problem. *Sociometry*, 32(4):425–443, 1969.

[63] Leslie G Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[64] Lilian Weng, Filippo Menczer, and Yong-Yeol Ahn. Virality prediction and community structure in social networks. *Scientific reports*, 3, 2013.

[65] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.

[66] Leishi Zhang, A. Stoffel, M. Behrisch, S. Mittelstadt, T. Schreck, R. Pompl, S. Weber, H. Last, and D. Keim. Visual analytics for the big data era — a comparative review of state-of-the-art commercial systems. In *Visual Analytics Science and Technology (VAST), 2012 IEEE Conference on*, pages 173–182, Oct 2012.

A

Streaming search terms

```
keywords = [
    "valg 2013",
    "valg2013",
    "arbeiderpartiet",
    "kristelig folkeparti",
    "krf",
    "frp",
    "fremskrittspartiet",
    "høyre",
    "sosialistisk venstreparti",
    "senterpartiet",
    "venstre",
    "rødt",
    "stortingsvalg",
    "nrkvalg",
    "stortingsvalget",
    "erna solberg",
    "jens stoltenberg",
    "siv jensen"
]
```