Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Laura Peer

Visualizing dynamics in frequent itemset mining timeseries

Semester Thesis SA-2013-43
April 2013 to July 2013

Tutors: Dr. Bernhard Ager, Dr. Xenofontas Dimitropoulos, Prof. Eduard Glatz
Supervisor: Prof. Dr. Bernhard Plattner

Abstract

Computer networks are evergrowing structures in throughput and complexity. Their increasing integration into essential and sensitive daily operations stresses the need for robustness and swift and reliable troubleshooting.
In a recent paper titled "Visualizing Big Network Traffic Data using Frequent Pattern Mining and Hypergraphs" by E. Glatz et al. [5], frequent itemset mining (FIM) is used to reduce massive network traces to a set of relevant network flows. They further display the extracted flows using hypergraphs. Glatz et al.'s approach works well for visualizing clear snapshots of a time interval. However, due to the very dynamic nature of the generated data, building perspicuous time semantics using hypergraphs proves to be difficult.
This thesis investigates an alternative visualization approach to improve the understanding and interpretation of network flow development in a timeseries. The result of this thesis is an application that generates a timeseries visualization of a selection of relevant traffic flows.

Contents

List of Figures

Chapter 1

Introduction

Computer networks are evergrowing structures in throughput and complexity. They are more and more intertwined with essential and sensitive global operations. Most users take computer networks for granted. That is until something unexpected happens that causes them to go down. This could potentially cause devastating economic results.
Such failures must be prevented and the risk thereof minimized through adequate security measures. Troubleshooting the state is one of these important measures and can be achieved by first collecting network traces, and then making sense of the gathered data, which can be massive in size.

In a recent paper titled "Visualizing Big Network Traffic Data using Frequent Pattern Mining and Hypergraphs", E. Glatz et al. [5], use frequent itemset mining (FIM) to reduce massive network traces to a set of relevant network flows. These flows are visualized using hypergraphs which nicely and adequately illustrate a clear snapshot of a time interval.
Figure 1.1 shows one example hypergraph. The blue circles represent frequent flows. These frequent flows are associated with a number of attributes shown in the red boxes. The connected attributes entirely define the flow characteristics. Let us look at the upper right frequent flow in Figure 1.1. Its label states that this flow makes up 6.0% of all the measured traffic. It has three outgoing arrows. These arrows specify that 6% of all traffic is directed from a single IP address towards one single destination IP address using the source port 80.
These hypergraphs are quite informative for static visualizations and provide a good insight into the current state of the network. Glatz et al. additionally visualize the evolution across multiple data aggregate measurements in time by generating a slide-show of evolving hypergraphs. When using data aggregates from a maximal frequent itemset miner, see Section 3, consecutive hypergraphs show many changes in attribute associations, which renders the slideshow too dynamic to effortlessly understand.
In this Thesis we develop a new scheme that focuses on visualizing the evolution of a time-series of maximal FIM measurements. We have been inspired by Randall Munroe's Congress comic [13] seen in Figure 1.2 to visualize flow frequencies as arc plots and attribute association changes as branches. Our goal is to to create a graphical representation of pre-analyzed network flow data by building upon the findings of Glatz et al. and using their aggregated data to plot an appealing visualization. The resulting graphic should clearly display frequency changes in flows and flow compositions. When looking at the visualization, the viewer should, at a glance, be able to determine the state of the measured network and spot potential issues like attacks or a possible network congestion.

Chapter 2 lists current related work in the field of visualizing network flow data. Chapter 3 provides an introduction into frequent itemset mining. Chapter 4 outlines our progress, problems and design decisions while developing our visualization application. Chapter 6 states unsolved issues and describes potential improvements and Chapter 7 gives a brief summary of the thesis .

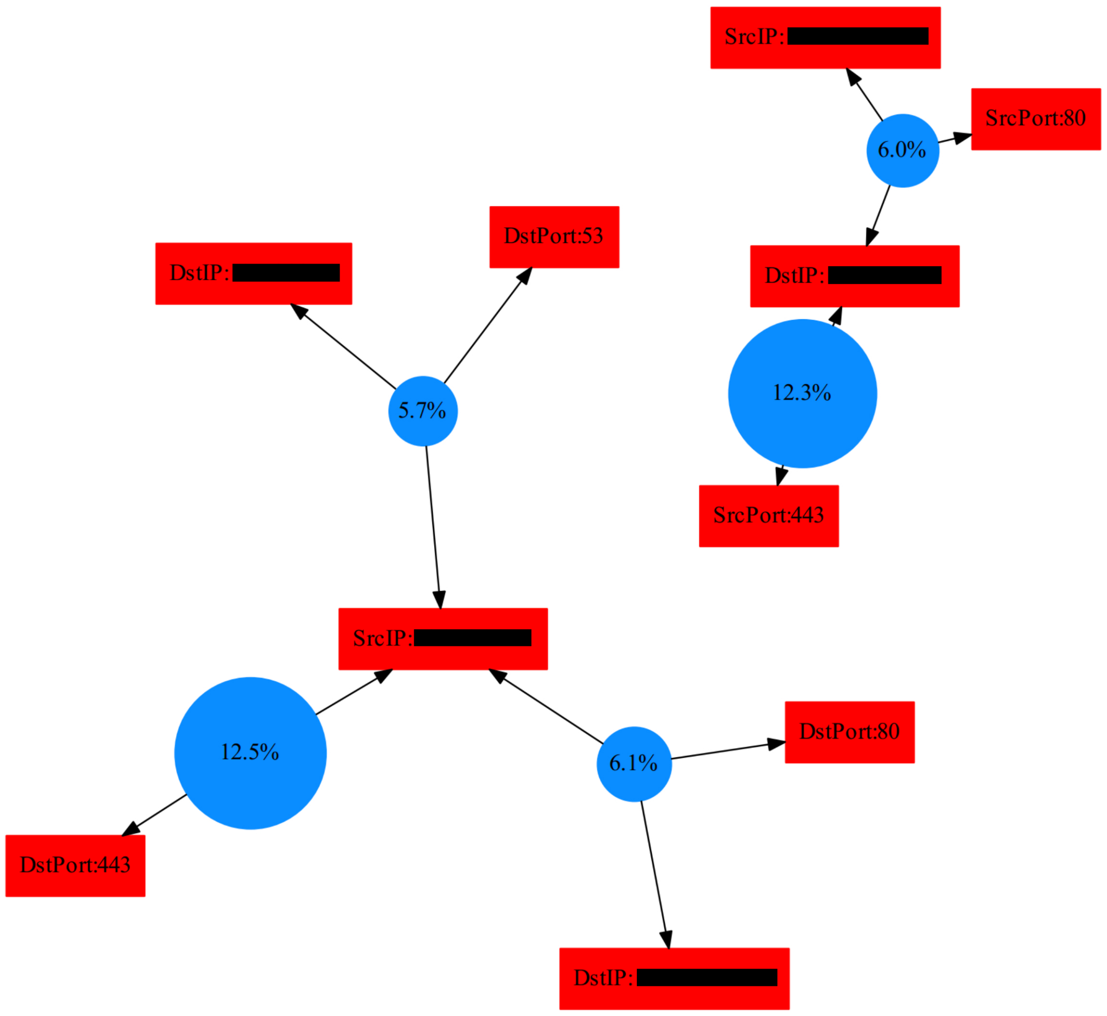
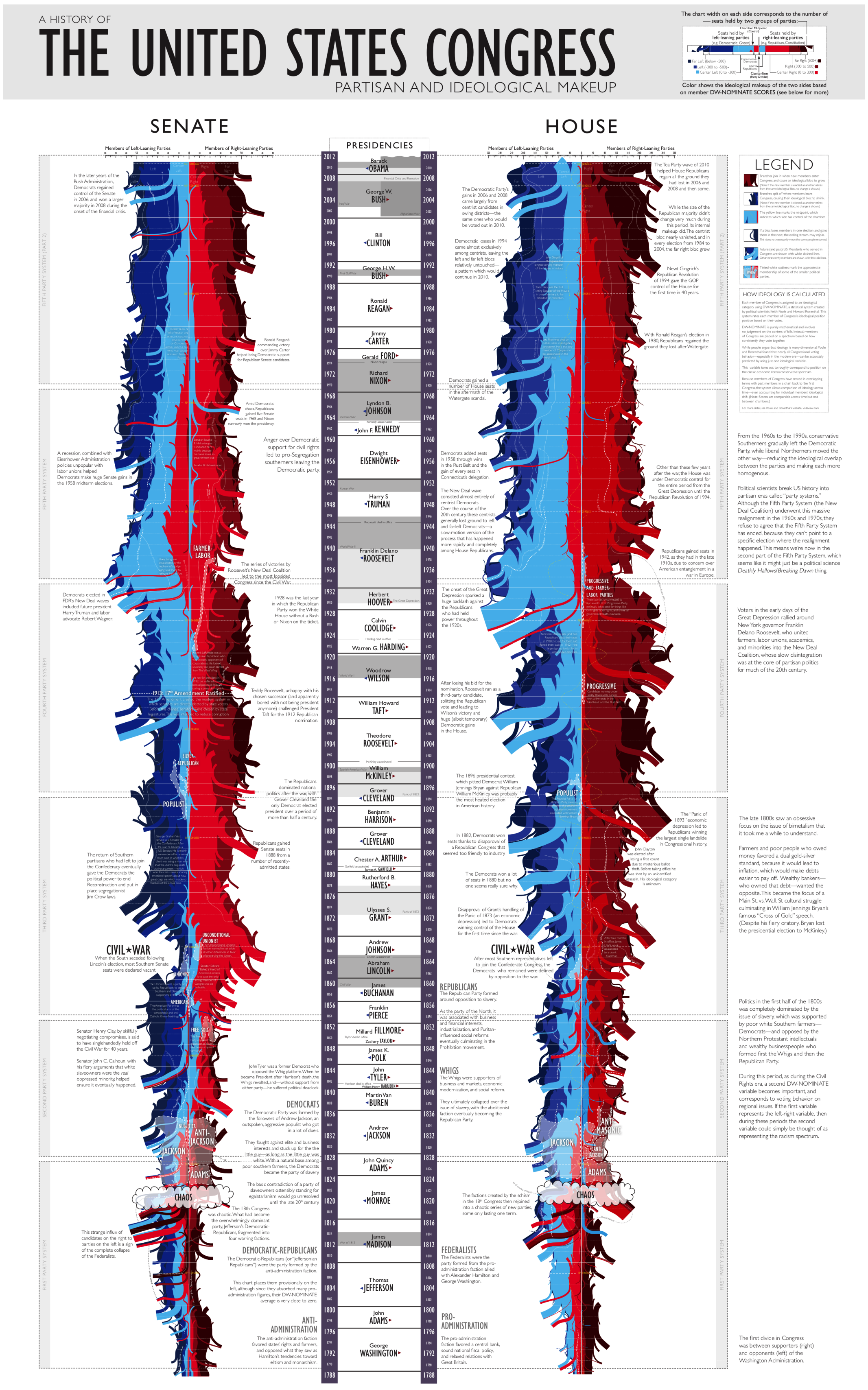


Figure 1.1: FIM Hypergraph



This Figure illustrates the evolving seat distribution among the policitcal parties within the U.S. Senate and House of Representatives. Time progresses from bottom to top.

Figure 1.2: XKCD: Congress

Chapter 2

Related Work

There is an extensive collection of available tools that visualize network traffic data. While some approaches use directed graphs to visualize connections between traffic attributes, others use scatterplots, histograms, parallel-coordinate plots, or treemaps.

AfterGlow [12] is a graph visualization tool that generates linked graphs based on user-defined specifications about edge thickness, node size, or coloring. The user provides the datasets in CSV format, for example monitored network traces generated using intrusion detection tools like tcpdump [17] or Snort [14]. The resulting visualizations are highly customizable graphs which use directed edges to emphasize relationships between relevant entities, for example source or destination IP addresses and port numbers.

In contrast, VisFlowConnect [19] uses parallel coordinate axes to display network traffic flows between internal and external hosts. The axes are drawn as parallel lines and each data point is represented as a chain of line segments which connects each of these axes at its data point value. Different line patterns or colors are used to distinguish data points. The visualization only shows data from a specified time window. In addition, VisFlowConnect features the ability to replay network events in an animation.

NFlowVis [11] uses a TreeMap [7] visualization to emphasize areas of greatly increased traffic. TreeMaps partition the space in hiearchically nested rectangles. The size of a rectangle is derived from the rectangle's measure of importance.

The Time-Based Network Traffic Visualizer(TNV) [6] provides a visualization over time of pre-recorded traffic dumps or, alternatively, live traffic. It uses a matrix display of host IP addresses and packet timestamps to depict flowing packets and links between local and remote hosts. The matrix displays time on the x-axis and source or destination IP addresses on the y-axis. Each column represents a time interval and each row a host. Each host's time interval is colored depending on the number of packets sent or received by the host. The analyst can zoom in and view interesting events up-close by employing search filters. Depending on the zoom level, the tool additionally shows an amount of preceding and succeeding traffic in raw data form.

Spinning Cube of Potential Doom [10], NVisionIP [9] and InetVis [18] visualize network traffic as 3-dimensional scatterplots. Traffic in an entire network, a subnet, or to a single machine can be analyzed. The scatterplot's x-axis represents all recorded source IP addresses, the y-axis shows the range of ports while the z-axis lists all addresses of the local network. The resulting clusters of points facilitate pattern detection while saving space when compared to a complex graph with many edges between nodes. NVisionIP extends this visualization scheme by additionally allowing the analyst to zoom in and explore interesting events in greater detail. InetVis further accepts the UDP and ICMP protocols and broadens the analyst's capablities by adding time-windows and replay features.

In BLINC [8], host behavior is analyzed at the transport layer. A range of behavioral patterns of network applications are observed and then visualized in graphs called graphlets, which are multipartite graphs that illustrate connections between source and destination IP addresses and port numbers. An unknown host application can be examined against all generated graphlets. If a close enough match is found, the application can be classified.

HAPviewer [4] transforms the graphlets used in BLINC to 5-partite graphlets by additionally classifying hosts according to the used protocol.

We base our approach on the work by Glatz et al. [5], which uses FIM mining to aggregate frequent flows from massive network traces. They further display the FIM output using hypergraphs, which visualize multi-attribute associations. Our paper extends the findings of Glatz et al. by implementing a visualization approach which plots the multi-attributive frequent flows in a timeseries and thus allows the viewer to inspect a time succession of FIM aggregate data.

Chapter 3

Frequent Itemset Mining

Frequent itemset mining (FIM) has been effectively used, for example, by Agrawal et al. [2], as an analysis tool for big data. FIM can be applied to a number of problems. Examples include network traffic analysis, trend analysis, stock market analysis, sensor network dataflows or e-commerce applications. FIM aims to find frequent patterns in data transactions.
We introduce the theory behind frequent itemset mining by referring to a simple explanatory scenario which originates from a scheme called market basket analysis.
Assume that a grocery store has n unique items that may be sold. Let $I = \{i_1, ..., i_n\}$ be the set of all items in the store. All customer purchases in the store are recorded in a set timespan. A customer purchase is called a transaction. Let $D = \{t_1, ..., t_m\}$ be the set of all recorded transactions. Every transaction in D contains a subset of items in I.
An itemset is defined as a subset of items of S. There are 2^n possible item combinations or itemsets. Every transaction contains a number of itemsets of set S. Subsets of transactions are itemsets of S as well. In terms of the market basket analysis, an example itemset is $\{milk, bread, butter\}$.
A frequent itemset is an itemset that appears in at least s transactions in D. The parameter s is called support and defines the proportion of transactions in D which contain the itemset in question. Choosing a support of 10%, the itemset $\{milk, bread, butter\}$ is only considered a frequent itemset if it appears in its entirety in at least 10% of all recorded transactions.
If we apply frequent itemset mining to network flow records, the items in I are not grocery store products but rather network attributes like IP addresses, port numbers, packet sizes. Transactions are not customer purchases, but rather recorded one-directional or bi-directional network packets or flows. However, the underlying theory applies accordingly.

A frequent itemset miner outputs a set $F = \{f_1, ..., f_l\}$ of l frequent itemsets all above a support of s. If the itemset $a = \{milk, bread, butter\}$ is frequent and thus in F, we can conclude that all (less specific) subset itemsets of a are also frequent and in F. Let $b = \{milk, bread\}$ be such a subset of itemset a. It is less specific because it imposes less constraints on the purchased items. It appears in at least as many transactions as itemset a, because it is completely contained in a.
If a frequent itemset in F has no superset in F, it is called a maximal frequent itemset. In maximal frequent itemset mining, all subsets of the maximal frequent itemsets are suppressed. This scheme can greatly reduce the number of generated frequent itemsets.
All subsets of a maximal frequent itemset hold less specificity compared to their maximal superset in F and are thus more frequent. For example, the frequent itemset $b = \{milk, bread\}$ holds less constraints when compared to the maximal frequent itemset $a = \{milk, bread, butter\}$.

Chapter 4

Design

The goal of this thesis is to visualize a timeseries of maximal frequent itemsets. After an analysis of the frequent itemset characteristics and observation of their evolution in the timeseries, we write an application that displays a graph which shows the time progression of FIM measurements in a clear and understandable way.
This chapter describes our process, our design decisions and the observed difficulties along the way. Sections 4.1 and 4.2 describe the basic shapes we decide to use for our plotting scheme. Section 4.3 introduces the concept of conflicting elements in the y-axis and describes mechanisms to minimize conflicting elements. In Section 4.4 we describe our visualization approach when the aforementioned conflict minimization schemes do not completely achieve the resolution of all occurring conflicts. Sections 4.5 and 4.6 describe our visualization's coloring and labeling schemes.

4.1 Plotting Itemset Transitions

Each FIM output file contains the list of frequent itemsets for its calculated time slice. The measured frequency of the itemsets represents an aggregate measurement over the measured time period. Based on ideas derived from the xkcd comic seen in Figure 1.2, we use a stacked graph

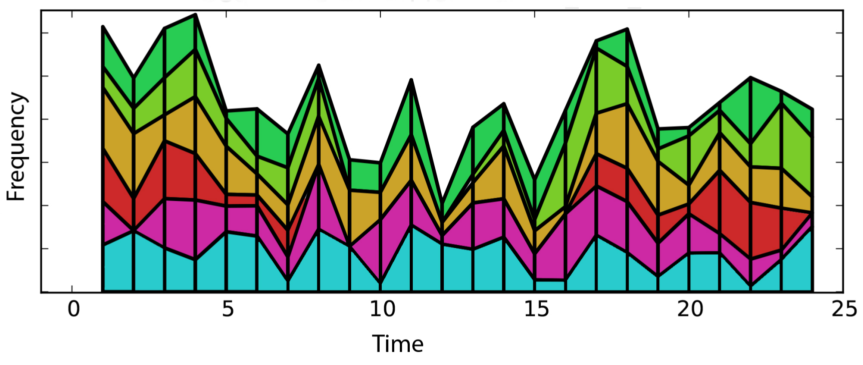


Figure 4.1: Stacked Graph

to illustrate the timeseries of itemset frequencies. A basic example of this scheme is shown in Figure 4.1. The x-axis denotes the sequence of subsequent measurements and can be thought of as the time axis. The width of an itemset in terms of the y-axis represents its measured frequency value. The stacked graph shows how itemsets grow and decrease in time.
Table 4.2 illustrates which itemset relationships can occur between two subsequent measurements in time. We look at two consecutive timeslots i and i+1. The nature of maximal frequent itemsets, as discussed in Section 3, causes appearances and disappearances of itemsets in consecutive timeslots. These changes occur when a frequent itemset falls below or rises above the support threshold s.

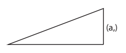
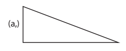
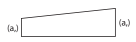
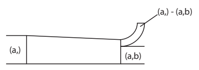
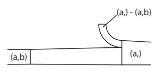
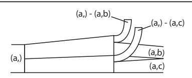
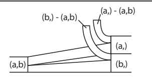
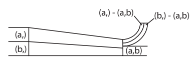
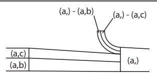
	Transition Scenarios	Timeslot i	Timeslot i+1	Itemset Connection Plots
1	itemset appears	-	(a,)	
2	itemset disappears	(a,)	-	
3	itemset stays	(a,)	(a,)	
4	gains attribute(s)	(a,)	(a,)	
5	loses attribute(s)	(a,b)	(a,)	
6	splits into supersets	(a,)	(a,b),(a,c),...	
7	splits into subsets	(a,b)	(a,),(b,),...	
8	merges into superset	(a,),(b,),...	(a,b)	
9	merges into subset	(a,b),(a,c),...	(a,)	

Table 4.2: Transition Scenarios

The fourth column in Table 4.2 shows the itemset transition shapes we designed for our visualization. Case 1 illustrates an itemset that appears in timeslot i+1. There is no subset or superset of this itemset in timeslot i. We draw a growing line from timeslot i to i+1. The itemset's frequency measurement in timeslot i+1 represents an aggregate measurement over the last time interval. The opposite of case 1 occurs in case 2, where an itemset disappears in timeslot i+1. Case 3 illustrates a connecting line between two consecutive measurements of a staying itemset.
Sometimes, instead of appearing or disappearing completely, itemsets lose or gain attributes in consecutive timeslots. We call itemsets with such observed attribute changes related. Two itemsets are related if the bigger itemset contains all set elements of the smaller itemsets. In other words, if it the bigger itemset is a superset of the smaller itemset and the smaller itemset is a subset of the bigger itemset. As stated in Section 3, a superset carries more specificity and is thus less frequent than its subset. An itemset in timeslot i may change into a single related itemset, as seen in cases 4 and 5. Cases 6 and 7 show an itemset changing into a number of related itemsets. Further, we see a number of itemsets in timeslot i merging into one itemset in timeslot i+1 in cases 8 and 9. The implications of related itemsets for our visualization scheme are further discussed in Section 4.2 where we introduce our concept of branches.

4.2 Inbranch and Outbranch Semantics

Maximal FIM data is highly dynamic. This is due to the fact that when a superset itemset becomes frequent enough to surpass the support threshold, it suppresses all its subsets including the previously frequent one. When a superset itemset suddenly loses frequency and drops below the support threshold, it no longer suppresses its next closest subset with frequency above the support threshold. These two scenarios result in frequent changes in appearances and disappearances of related itemsets and fluctuating frequencies across subsequent timeslots.
To illustrate the dynamic nature of maximal FIM data, we choose to draw inbranches or outbranches, inspired by the xkcd comic seen in Figure 1.2, whenever an itemset loses or gains attributes and thus frequency between subsequent timeslots.

Cases 4 and 5 in Table 4.2 describe a simple transition between related itemsets. In case 4 itemset (a) gains the attribute (b) and transitions to itemset (a,b). Since the itemset (a,b) is more specific and thus less frequent, we can note a frequency decrease in this transition from (a) to (a,b), as shown in the visualization in the fourth column. We highlight this decrease in frequency by drawing an outward branching curve out of the graph. The outbranch value in this case is (a)-(a,b). In case 5 the opposite happens. Itemset (a,b) loses attribute (b) and transitions to itemset (a) with an increase in frequency. We highlight this increase in frequency by drawing an inward branching curve into the graph. The inbranch value in this case is (a)-(a,b). The resulting width of the inbranch represents the comparative frequency gain incurred by losing specificity (when losing the attribute (b)), while the resulting outbranch width represents the comparative frequency loss incurred by gaining specificity (when gaining the attribute (b)). Cases 6 and 7 in Table 4.2 show two cases where an itemset splits into a number of itemsets. Essentially, we connect the initial itemset to each of the subsequent itemsets and draw and calculate individual branches as in the simple case. Cases 8 and 9 in Table 4.2 illustrate two cases where multiple itemsets merge into a related itemset. Here, each merging itemset is compared to the resulting itemset and the complete branch value is calculated as the sum of the individual branches.

In more complex graphs, itemsets regularly split and merge. It can become considerably complex to distinguish these merges and splits from simple transitions, which is why we add so called hatches to such itemset transition plots. Hatches are simply a pattern placed on top of the already colored shapes. We use vertical lines to illustrate that the itemset will split into multiple itemsets in the next timeslot and horizontal lines to illustrate a situation where itemsets merge into one resulting itemset.

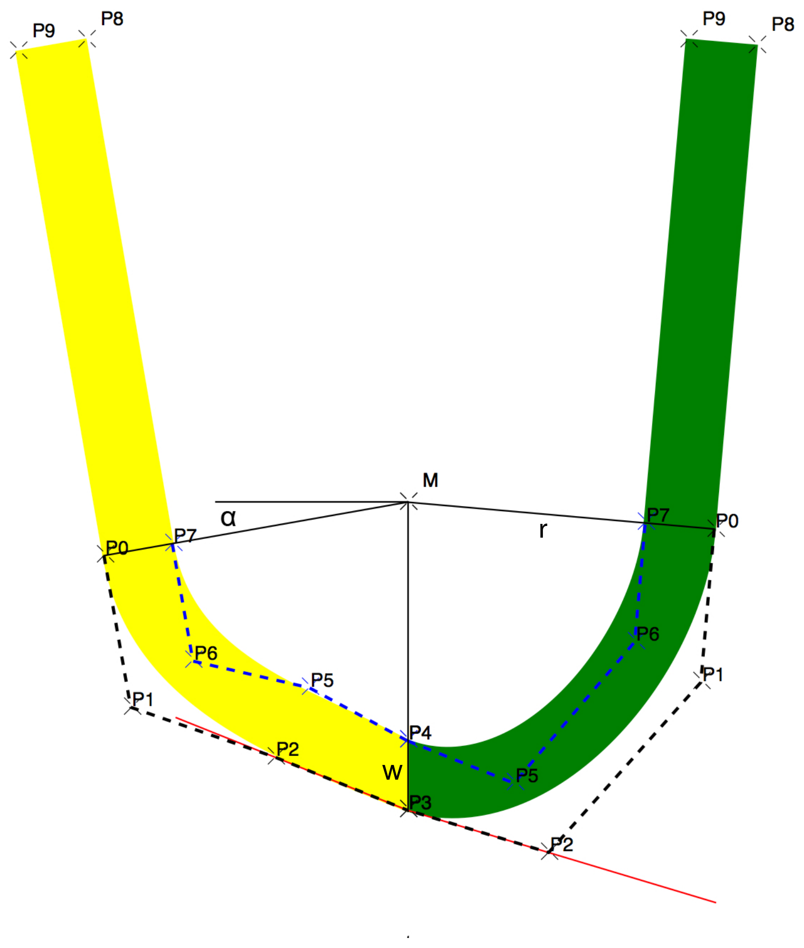


Figure 4.3: Inbranch and Outbranch Plots

Figure 4.3 shows a yellow inbranch and green outbranch. We use quadratic Bezier splines [3] to draw the lower and curved part of the branches, namely the two curved segments. We use 2 control points for each spline, for example P1,P2 or P5,P6. The spline stays contained within the convex hull formed by the two control points, the start point and the end point. The resulting spline connects from the start point to the endpoint. Its path is smoothly bent towards both control points along the way. The red lines below the branch curves represents an example slope of a lower itemset in the plot. A benefit of using Bezier splines for the curved segment of the branch is the fact that the spline shape can easily be aligned to this line by simply placing the control point on the line. At a set angle α, which generates the points P0 and P7, the branch lines transform into straight lines that prolong the branch upward in a rectangle. This draws the branch shapes out of the graphed area.

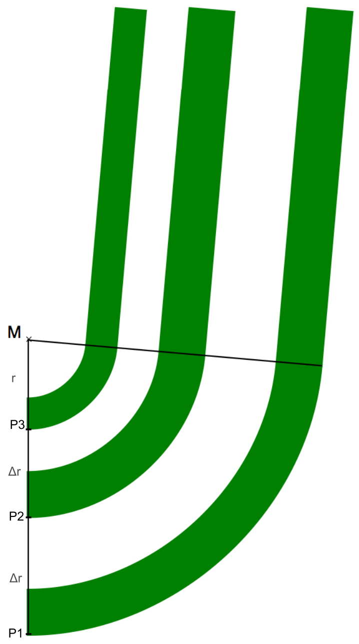


Figure 4.4: Multiple Branches

It is possible for timeslots to contain multiple branches. To perserve visual clarity, these branches should not intersect. We solve this potential problem by gradually increasing the radius for each lower positioned branch. Figure 4.4 shows non-intersecting branches of different radius lengths. We add the y-axis position offset of subsequent branches to the upper radius to calculate the next lower radius.

4.3 Layout Conflicts

An initial idea, which results in itemsets being connected in a stacked manner similar to Figure 4.1, is to plot the itemsets using a static ordering in terms of the y-axis. In this scheme, every itemset is given a unique id and we plot the itemsets in the y-axis according to ascending ids. Unfortunately, this approach results in an insufficient graphical representation, because it is susceptible to layout conflicts.
Layout conflicts occur in the visualization when non-neighboring itemsets merge. The connecting shapes between the merging itemsets, seen in cases 8 and 9 in Table 4.2, intersect with the connecting shapes of the itemsets stuck in between the merging group of itemsets. Figure 4.5 illustrates such a layout conflict.
Choosing a randomly selected static sequence of itemsets leads to unforseen situations, which renders the approach unusable. Permuting through all possible static sequences of itemsets is infeasible because the high number of distinct itemsets would prevent our application to reach any conclusion in due time. We can improve this time complexity problem, by partitioning the

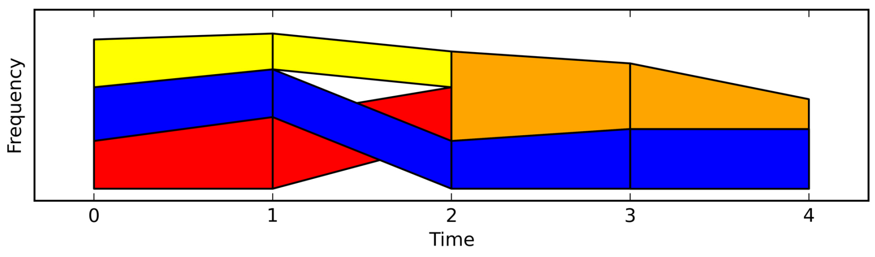


Figure 4.5: Layout Conflict

sequence into non-conflicting components. To achieve this, we build a structure we call Time Transition Graph. An example graph of medium complexity is shown in Figure 4.6.

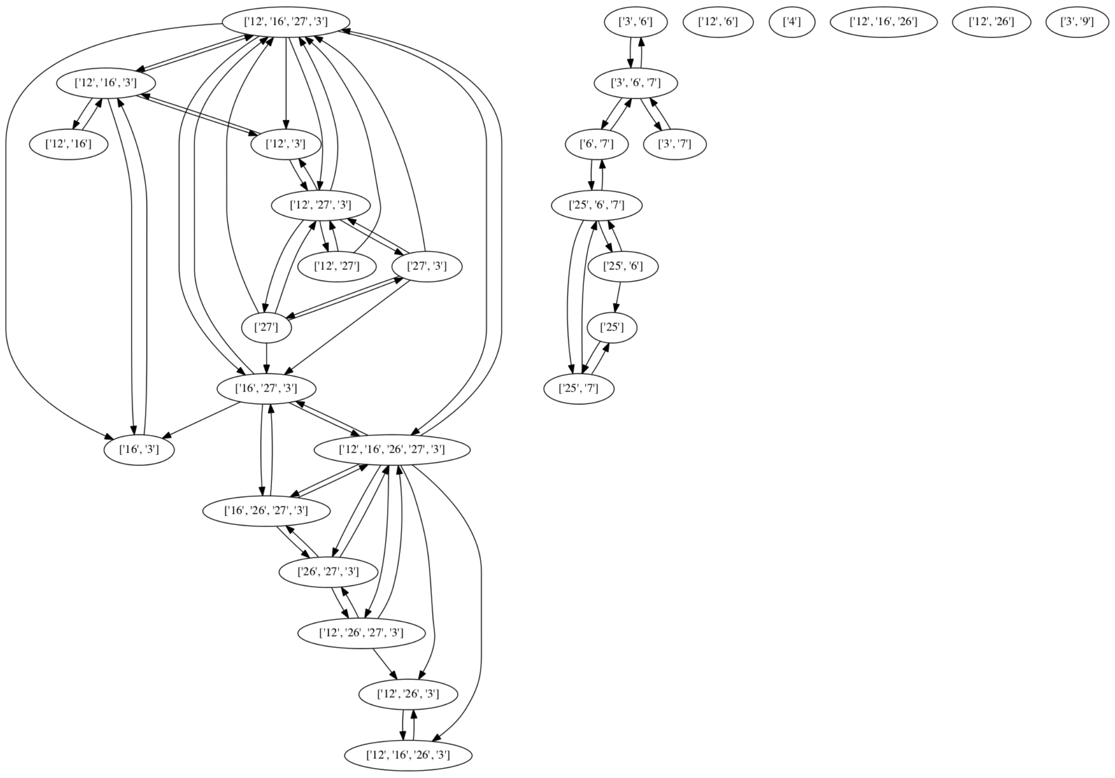


Figure 4.6: Time Transition Graph

The Time Transition Graph has one node for each distinct itemset generated by the FIM software. The edges are drawn by reading the FIM timeseries data. Every time a transition takes place between related itemsets in subsequent timeslots, an edge between the two itemsets is added in the Time Transition Graph.
Once generated, the graph is dissected into weakly connected components. A weakly connected component has no edges leading to any other such component, which means that the node group within a component may only conflict within itself. The generated components represent non-conflicting groups, because having no edge connection in the Time Transition Graph means having no transition and thus no conflict potential.
With the permutation size decreased, we first try to solve conflicts using a backtracking [15] algorithm. This approach fails in solving the problem of minimizing layout conflicts, but is nonetheless worth discussing, because it inspires the formulation of our final solution.

We implement the backtracking algorithm as a recursive algorithm that keeps appending itemsets to an initially empty sequence. It traverses the backtracker's search tree depth-first

and abandons any incorrect partial solution as soon as a colliding itemset is added to the growing sequence. A conflict is detected, when elements within the sequence share an edge in the Time Transition Graph but are not next to each other in the sequence. This means that the itemsets merge at some point in time, but are not next to each other in the y-axis sequence.
It returns the first complete sequence of itemsets that does not contain conflicting itemsets. As stated before, this approach does not find a solution if we run it on some of the more complex datasets. Because, as it turns out, there may not be a clean solution. Our investigation into why the backtracking algorithm produces lacking results leads us to two key findings. Our first finding

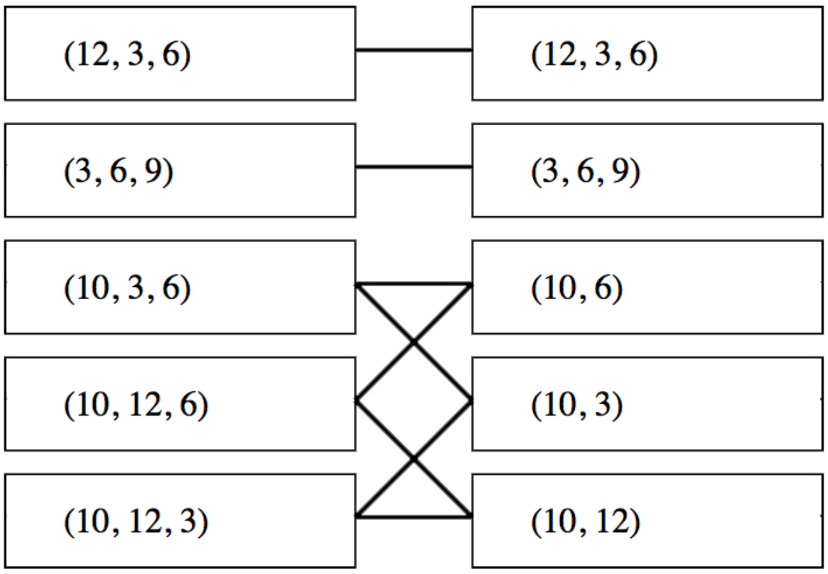


Figure 4.7: Unavoidable Conflict

is that there are certain itemset transitions where conflicts occur no matter how the itemsets are reordered. Figure 4.7 illustrates an example transition that has no conflict-free solution. Our

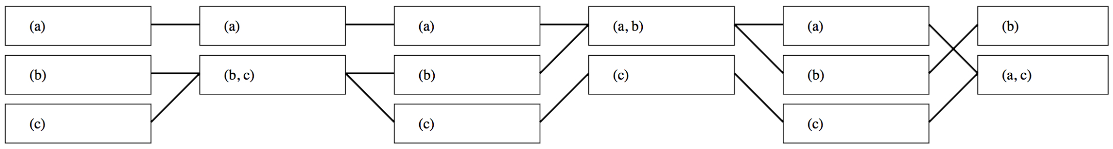


Figure 4.8: Static Split-Merge Scenario

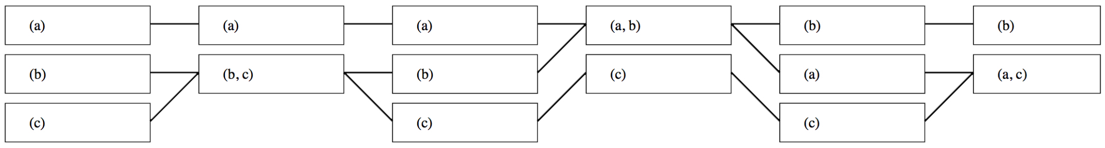


Figure 4.9: Dynamic Split-Merge Scenario

second finding can be explained by inspecting the timeseries shown in Figures 4.8 and 4.9.
Both Figures show a succession of splits and merges of itemsets (a),(b),(c),(a,b),(a,c) and (b,c). First, (b) and (c) merge and then split. Next, itemsets (a) and (b) merge and then split. Finally, (a) and (c) merge. In Figure 4.8 a static initial itemset ordering is imposed on the complete timeseries, which results in a conflict. In Figure 4.9 we allow the sequence to be reordered, which resolves the conflict.

We find, when using a static sequence, that this simple scenario always results in a conflict. Both (a) and (b) merge with (c) at some point, which means they both need to be neighbors of (c). The only way this can be assured is if (c) is placed between (a) and (b) in the sequence. Itemsets (a) and (b) also merge and thus need to be neighbors as well. Both requirements cannot be met when using a static sequence. This results in a layout conflict.
By abandoning the static sequencing approach we find a very simple and elegant solution, seen in Figure 4.9. The itemsets (a) and (b) are reordered after they split and allow (a) and (c) to be neighbors before they merge.

Our final approach of dealing with conflict minimization is to use an individual sequence for each timeslot to describe the ordering of itemsets along the y-axis. For this purpose we write an algorithm we call Conflict Solver, which collects all the itemsets from the FIM data and groups the read itemsets by their occurrence time, which results in one group of itemsets per timeslot. The algorithm first goes through the collected groups in forward motion, then it does the same backwards. At every step, it builds edges between the current and subsequent group of itemsets. Edges in this case denote a resulting connection between the itemsets in the visualization, which occurs when two consecutive itemsets stay the same or transition into a subset or superset.
It changes one itemset at a time on the right hand or left hand sequence (depending on the direction) and finds the best placement based on a cost function. The cost function is a weighted mixture of transition slope and edge intersection count. While it is the goal of achieving minimal intersections between the sequences, we also choose to return solutions with minimal edge slope, since the resulting plot looks better if connected itemset positions are not too shifted in terms of the y-axis. This is useful when setting the itemset labels, explained in Section 4.6

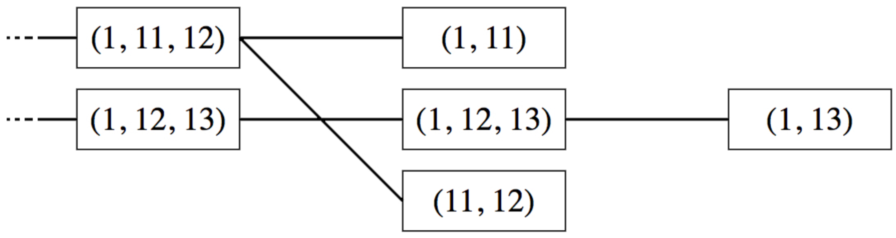


Figure 4.10: Forward Solvable Conflict

We let the algorithm run in both directions across the timeslot sequences, because some conflicts cannot be solved by only moving in one direction. Figure 4.10 shows the sequence output of the algorithm when only allowed to move in backward direction. We can clearly see a conflict between the first and second sequences in the Figure. This conflict, however is not complex. It can easily be solved by changing the second sequence and evaluating the edge intersections between the first and second sequences. When going backwards, the algorithm only changes the left hand sequence and this simple solution is not found. When moving in the forward direction, we change the right hand sequence and thereby solve the conflict.
After one round, which consists of one forward and one backward traversal of the sequences, it, again, checks the cost function output for improvement. If it finds improvement, it continues to do further rounds. If not, it exits.
We let the algorithm run a finite number of rounds. The round cutoff number is given to our program as an input parameter and is set to a default value of four rounds.

4.4 Splice Timeslot

In Section 4.3, we discuss conflicts and describe our approach of minimizing their occurrence by building non-conflicting sequences of itemsets for each timeslot.
There are certain itemset transitions where conflicts occur no matter how the itemsets are reordered. Figure 4.7 illustrates an example transition that has no conflict-free solution.

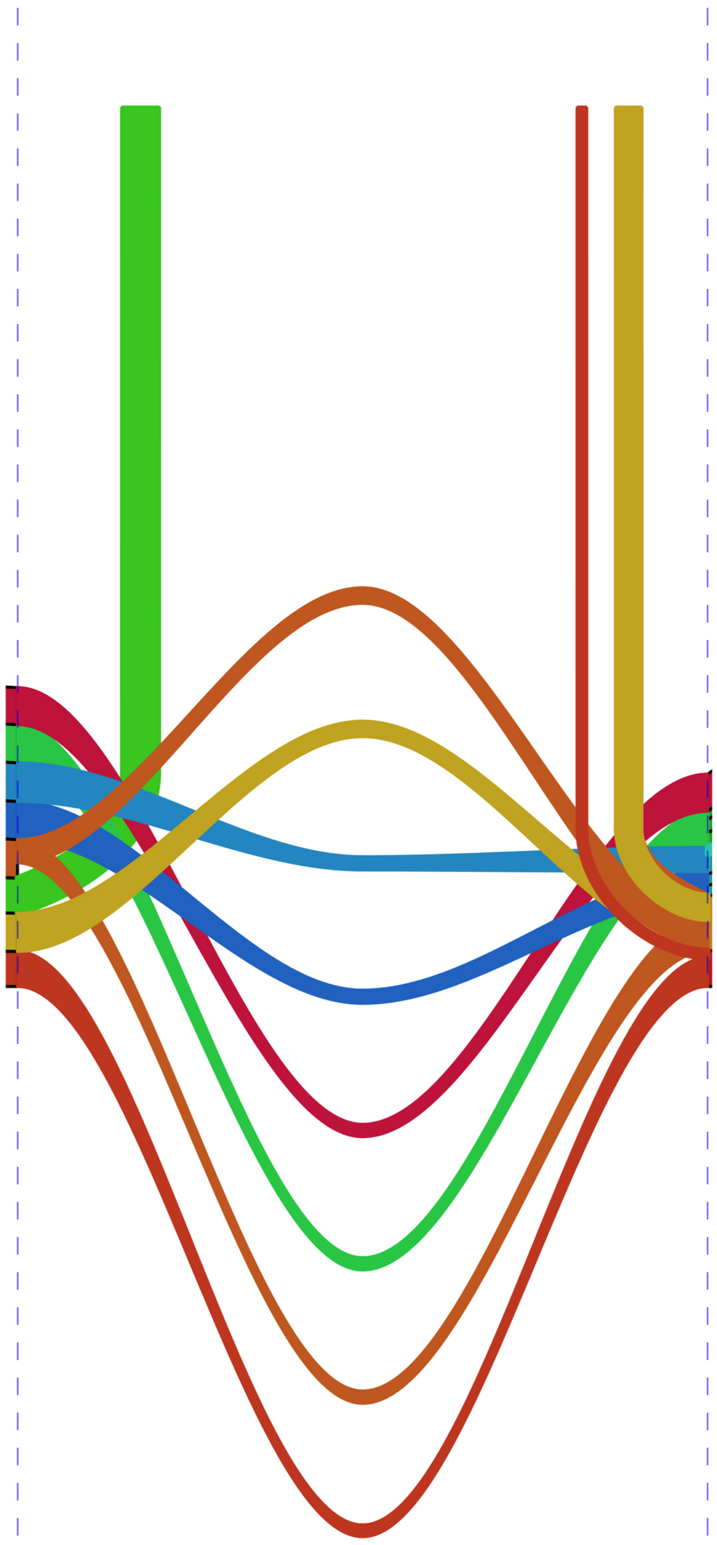


Figure 4.11: Splice Timeslot

Everytime an unavoidable conflict is encountered in the timeseries, we draw a special timeslot called Splice timeslot. See Figure 4.11 for an example.

This timeslot is stretched in terms of the x-axis by a factor, in order to enhance the timeslot visualization. The connecting shapes between the itemsets are no longer straight lines. Instead, we choose to draw all itemset transitions as curved lines that unravel at the start and then lead

back into the plot at the end of the timeslot.
In terms of the y-axis, these curves are no longer stacked neatly on top of each other compared to a regular timeslot with straight lines. We therefore allow intersections between the lines in terms of the y-axis.
This visualization approach is derived from the idea of a rope that is unlayed in order to separate its strands.

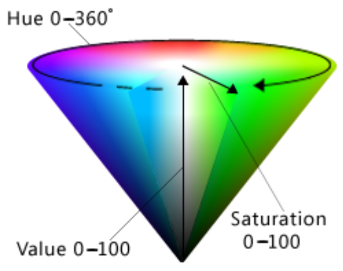


Figure 4.12: HSV color space

4.5 Itemset Coloring

Some datasets hold a large number of itemsets. These itemsets all need to be plotted in the timeseries. Even though we use labels, see Section 4.6, to distinguish between different itemsets, it improves the graph clarity to use a unique color for each itemset.
Our goal is to generate colors that are distinguishable from each other. We achieve this by choosing a different hue value for each itemset. In addition, we do not want any single itemset to stand out from the rest. This means that the percieved color intensity or saturation for all itemsets should be equal.
Of a number of studied color spaces, we find that the Hue-Saturation-Value (HSV) color space best meets our criteria.
In the HSV color space, any color can be described in terms of its hue, value and saturation. Figure 4.12, taken from [1], illustrates this cone shaped color space. As the value parameter increases, the brightness of the color increases. The color saturation increases as the radial offset from the centre grows. The color's hue is an angular measurement. If the angle is set at 0°, the resulting color becomes red. At angle 120° we produce green and at 240° blue. See [16] for more information.
For our purposes, since we want bright and saturated color tones across all itemset colorings, we use constant values for the parameters value and saturation. The individual itemset hues are set by simply dividing the 360° of possible hue angles by the number of itemsets we wish to color. The result is a sequence of equidistant points, one for each itemset, along a fixed circle segment within the cone. The circle segment size is defined by the radius, which is set by the constant saturation parameter. The circle's position within the cone is defined by the constant value parameter.
As previously discussed, some itemsets transform into other itemsets within the timeseries. We shall from now on call such transitioning itemsets related itemsets. These related itemsets share edges in the Time Transition Graph in Figure 4.6. To illustrate that these itemsets are related, we choose to use similar colors for related itemsets.

We build a sequence of all itemsets such that related itemsets are positioned closer together. This is achieved by first splitting the Time Transition Graph into separate components, then traversing each component starting with the nodes with minimal attributes. Whenever a node with children is encountered, the children itemsets are added on the right hand side of the parent in the sequence. This way, mutual children between two parent nodes are placed between the two parents in the sequence and will be given a color hue value that is between the two parent colors.

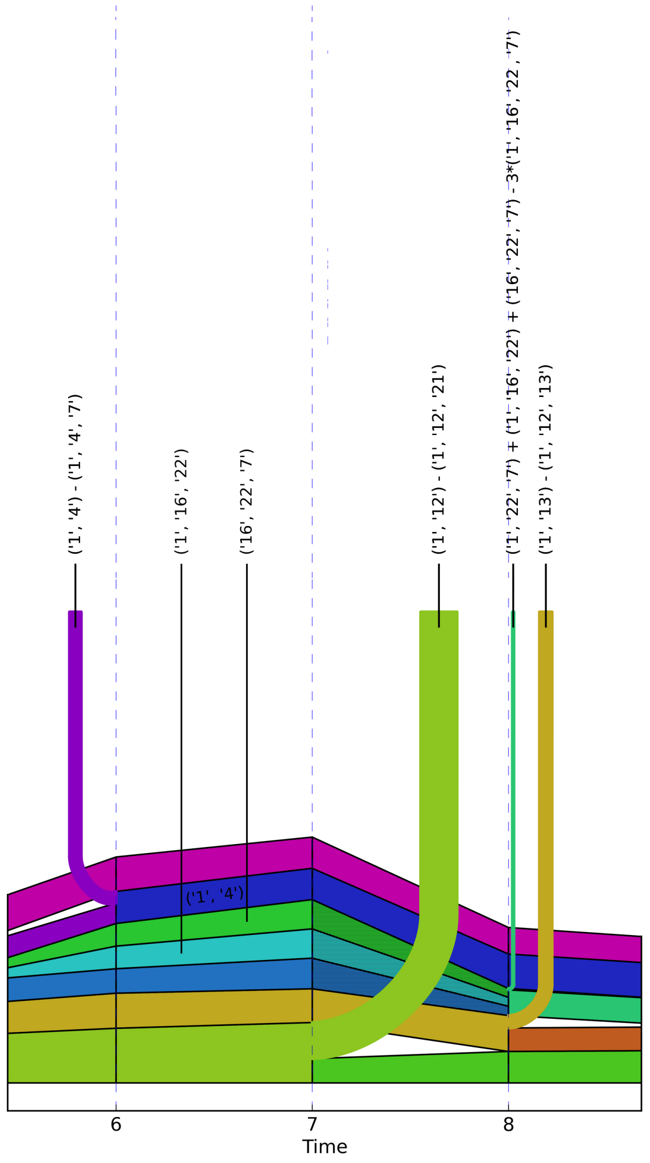


Figure 4.13: Labeling

4.6 Labeling

In addition to using colors to distinguish between different itemsets, as explained in Section 4.5, we decide to use visible text labels whenever a new itemset appears in the timeseries.
Especially in complex graphs with many itemsets and thus many very similar itemset colors, this solution helps the viewer to distinguish between itemsets. Because of the dynamic nature of FIM data, itemsets sometimes disappear for a number of timeslots. We want to make sure that we draw a separate label for every single period of occurrence of an itemset. The need

for this strict labeling scheme is again caused by the fact that itemset colors are not enough to distinguish between different itemsets.
We first analyze the FIM datasets and calculate exact placement ranges or lists of possible timeslots for each itemset label. The placement of the labels should be evenly distributed among the timeseries, otherwise some timeslots remain empty and others are cluttered with labels, which impairs visibility and may even cause labeltext intersections.
We assign a mobility number to each label. This mobility number corresponds to the number of possible timeslots a label can be assigned to. Our implementation aggregates labels with the same mobility value in moblity groups. Starting with the lowest mobility group, it selects a random label from within the group and assigns it to the currently most empty or label-free timeslot. Once all the labels of current mobility are placed, it moves on to the next higher mobility group.
Ideally, the label fits exactly into the itemset transition shape, but in cases where the space is too small for the chosen font size, we position the label outside of the plot and use a vertical black line to connect the label with the itemset transition shape. We choose to draw the label texts vertically upwards for labels that don't fit into the plot, because some itemset labels are long strings and would take up too much space when drawn horizontally. Indeed, such long strings are typical for the network FIM data we are visualizing, because some attributes in these itemsets are, for example, IP source or destination addresses that contain quite a lot of characters. For labels that fit into the plot, we rotate the label text such that it is aligned with the itemset transition shape slope.
In addition to labeling itemsets, we also need to assign and display frequency values to the outbranches and inbranches described in Section 4.2. We do this by setting branchlabels. These branchlabel texts can contain even more string characters than the itemset label texts, because the branch width is calculated by subtracting the less specific itemset from the more specific itemset, as described in Section 4.2. Here, as well, we decide to use a black line connecting the outbranch or inbranch to the branchlabel. The branchlabel text is also drawn vertically upwards.
We prevent the itemset labels from colliding with the branchlabels by splitting timeslots with branches into three zones. The first zone is set between the timeslot start and the first outbranch curve. The second zone is set between the last outbranch and the inbranch curves. The third zone is set between the timeslot end and the closest inbranch curve. Itemset labels are distributed among the zones based on the zone size, and within the zones, the labels are placed equidistantly. Figure 4.13 illustrates our labeling scheme.

Chapter 5

Results

In order to determine the resulting quality of our application, we first compare and judge a number of generated visualizations at different complexity levels in Section 5.1. Then we discuss the runtime of our application in Section 5.2. In Section 5.3, we conclude our judgement by discussing the limitations of our visualization approach.

5.1 Visualization Results

Figure 5.1 shows a FIM timeseries visualization produced using a low complexity FIM dataset. The data used to generate this visualization is listed in Table 5.5 under "20 b".
There are 8 distinct itemsets in this visualization. Due to this relatively small number of itemsets, the colors are easily distinguishable. Related itemsets that transition into each other are indeed similarly colored. For example, the orange colored itemset transitions into the red itemset and the green itemset transitions into the turquoise itemset. The other itemset colors in the graph are distinguishably different. There are only 5 inbranches and 3 outbranches drawn into and out of the plot during these few itemset transitions due to incurred frequency gains or losses. One itemset split occurs in the last timeslot. The labels are clearly visible both inside the shapes, where they fit, and outside of the plot connected with their connected line.
From a plot like this, a viewer can clearly gather all relevant information at a glance. This represents a good result.
Figure 5.2 shows a FIM timeseries visualization produced using a medium complexity FIM dataset. The data used to generate this visualization is listed in Table 5.5 under "15 b".
Compared to Figure 5.1, this plot is quite a bit more complex. There are 14 distinct itemsets in the visualization. The number of itemsets does not pose a problem when trying to distinguish separate itemset transition shapes between subsequent timeslots. The coloring scheme still works. We are able to see clear distinctions between the itemsets.
There is an increase in related itemsets in this visualization compared to Figure 5.2. We can see a total of 16 inbranches and 8 outbranches. The transitioning itemsets have similar colors. The branches are typically small in width but do not intersect, even if they are drawn in the same timeslot.
We find 5 splits and the same amount of merges in this visualitzation. As described shortly in Section 4.2, we use hatches (horizontal and vertical line patterns) to emphasize splitting or merging transitions of itemsets.
Our y-axis space is almost doubled because of the long branchlabel strings, which unfortunately compresses the itemset shapes. In this example, however, we can still distinguish single itemsets because there are not that many in the timeslot sequences and their colors are far enough apart.
The shapes connecting the itemsets in subsequent timeslots are too small to hold any label texts, which results in all the labels being drawn on the outside of the plot. We can see, however, that the labeling scheme is at its limit in this visualization. The labels in most of the timeslots are very close. In timeslot 16, for example, we detect two intersections between labels.
This visualization contains a Splice timeslot. Our algorithm is not able to resolve all conflict in the timeseries. In fact, Table 5.5 shows that there is one unresolved conflict in timeslot 17. The

Splice timeslot itemset colors are far enough apart that we can clearly discern where the curves originate and end.

At a first glance, this plot is confusingly complex. But when we closely inspect single itemsets or small numbers of itemsets it is possible to gather some knowledge about the flow information contained in this visualization, because the labels and the colored itemsets enable the specifc lookup of information.

We conclude from this result, that our application makes a good effort to visualize medium complexity FIM datasets. The visualization is not initially or entirely clear, but a small effort on the analyst's side enormously aids in his understanding of the underlying FIM timeseries.

Figures 5.3 and 5.4 show a FIM timeseries visualization produced by using high complexity FIM datasets. The data used to generate these two visualizations is listed in Table 5.5 under "10 io" and "5 b", respectively.

Figure 5.3 illustrates 53 unique itemsets and Figure 5.4 illustrates 150.

In these visualizations, we can detect a weakness in the chosen itemset coloring scheme. Itemsets cannot be distinguished by their color value alone anymore. Itemset transitions are only visible due to the existence of branches, because the itemsets that transition virtually have indistinguishable colors. The itemset transition shapes are too narrow to discern as well because the long branchlabel texts enhance the y-axis space that needs to be visualized. We can thus only tell apart similarly colored groups of itemsets.

Splitting and merging itemsets are almost impossible to detect, also due to the narrow itemset transition shapes. The visualization in Figure 5.4 shows 61 splits and 61 merges.

The labeling is positioned on the outside of the plot as expected due to the narrow itemset transition shapes. The sequences are higher in terms of the y-axis, because there are generally more itemsets per timeslot, when compared to other, less complex, visualizations. The large number of small itemsets per timeslot and the many branches, which consume additional space, produce many labels per timeslot, which leads to label text intersections.

To make matters worse, any occurring Splice timeslots grow quite long in terms of the y-axis, because of the large number of itemset per timeslot. The remaining plot shrinks further in order to accomodate the entire height of the Splice timeslot. This decreases the overall detail of the visualization. With all the visualization schemes broken, the only thing that can be read from such a complex visualization is the overall frequency evolution pattern of all itemsets or similarly colored groups.

We conclude that our application is not able to adequately visualize such complex FIM datasets. Yet, we argue that the sheer number of details in such a FIM timeseries prohibits producing an understandable visualization. Thus, this is a problem imposed on the data level, which needs to be fixed on that level.

5.2 Conflict Solver Runtime

The Conflict Solver Algorithm, described in Section 4.3, is responsible for minimizing conflicts. It takes unordered time sequences and reorders them such that conflict occurrences are minimized and its successfulness can be measured by counting the still remaining conflicts after its completion and determining its runtime.

The algorithm has proven to be successful in minimizing conflicts up until medium complexity datasets. When using it against the provided FIM datasets, it is able to quickly solve all but unsolvable conflicts in the timeseries. That is, except for in the more complex datasets. It is effective, because it repeatedly reorders itemsets and only ever accepts improved solutions. These repeated improvements eventually lead to an acceptable solution. However, if used with more complex datasets, computation can get slow, because the lengths of the timeslot sequences are bigger, which means an increase of the time needed to find the best itemset reordering at each timeslot.

The most complex dataset is listed in the Table as "5 io". It contains 540 unique itemsets and its timeslot sequences range from 35 to 244 itemsets. Generating a visualizatoin of this dataset, using our default settings, takes more than 26 minutes.

As described in Section 4.3, we can reduce the runtime of the application by setting a constant value of rounds we allow the algorithm to perform. We measure round durations of between 3 and 4 minutes in the most complex dataset. Obviously, more rounds can lead to better results for reducing conflicts in more complex datasets. However, as visualizations of such highly complex data are of limited utility and runtimes for simple datasets are very low, we refrain from further optimization.

5.3 Limitations

Section 5.1 illustrates that our application can successfully visualize FIM datasets up until it reaches a complexity of medium value.
Complex input data increases the total itemset count, which causes the color scheme to deteriorate. This issue can be mitigated using labels.
An increase in itemsets also produces an increase in height of the plot and results in the need to place more labels. In addition, probabilistically, more itemsets will take part in more itemset transitions, which brings with it a greater number of Infows or outbranches. Table 5.5 shows a correlation between the itemset number and the number of outbranches or inbranches.
In order to accomodate more labels and branches in the fixed-size timeslots, while still preserving nonintersecting elements, we can stretch the x-axis, while preserving the y-axis at fixed size, otherwise the branches grow as well, because their radius depends on the branch width, which is set in terms of the y-axis.
This stretching of the x-axis, however, can only practically be done to a degree, because it in turn decreases the y-axis details. The y-axis details are important for distinguishing connected itemset shapes between timeslots, which equally important for visual clarity.
Additionally, there is a limit of x-axis slots we are able to draw while still being able to recognize the itemset shapes in the y-axis. Plotting timeseries with more than 50 timeslots yields insufficient visualzations, where the observer may only see the overall frequency evolution pattern of all itemsets or similarly colored groups.

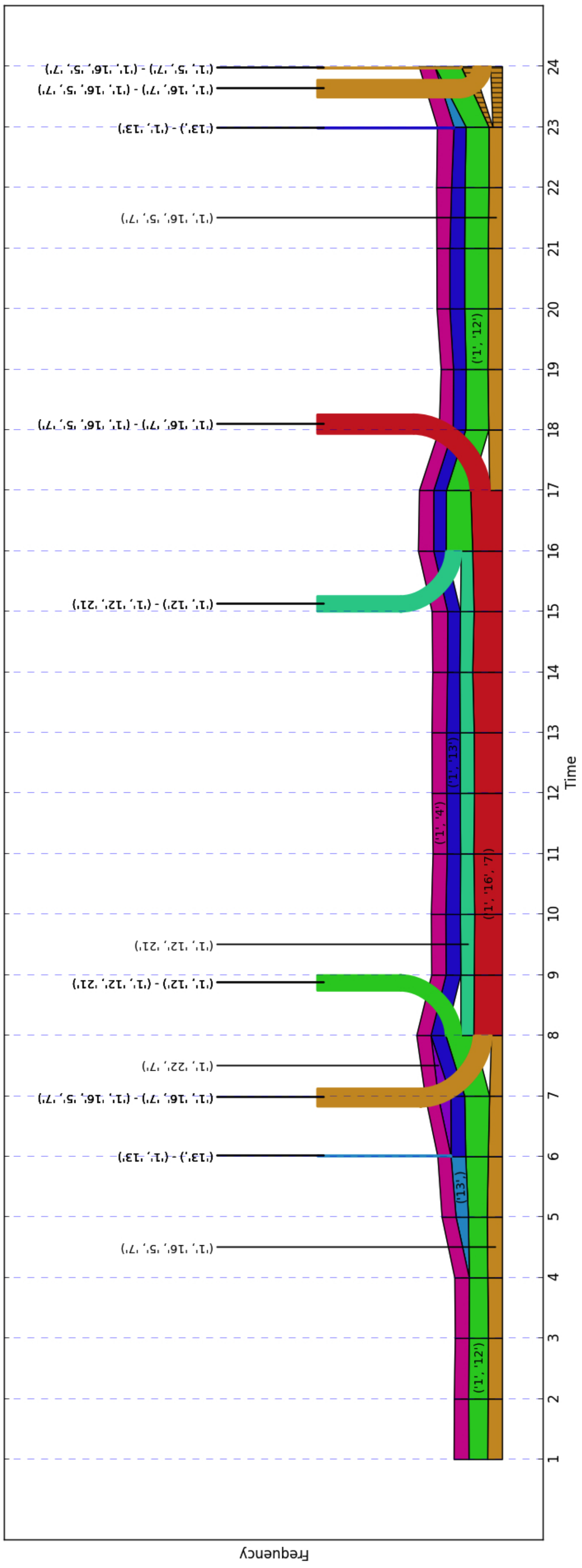


Figure 5.1: Resulting Visualization of a low complexity timeseries

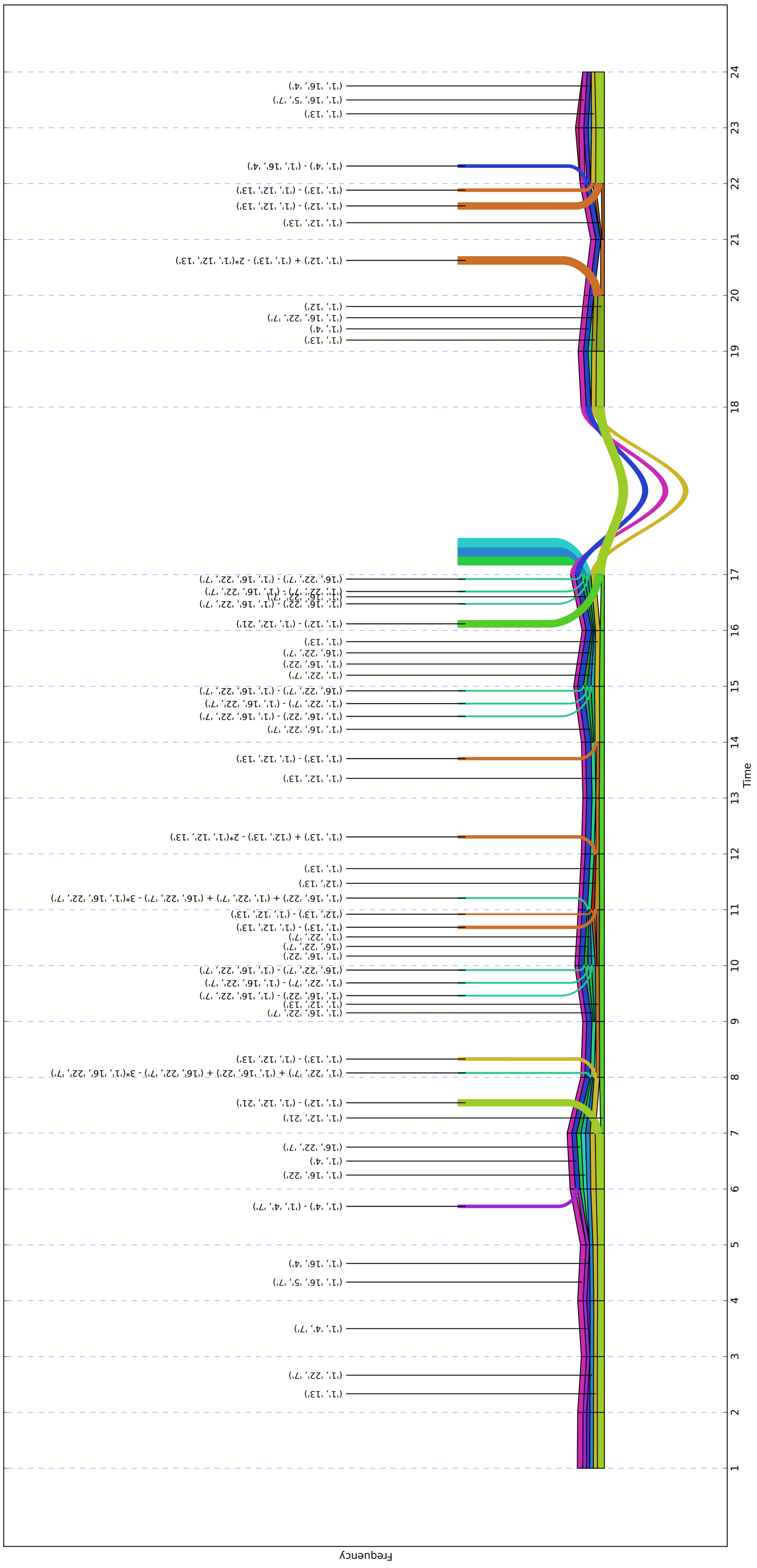


Figure 5.2: Resulting Visualization of a medium complexity timeseries

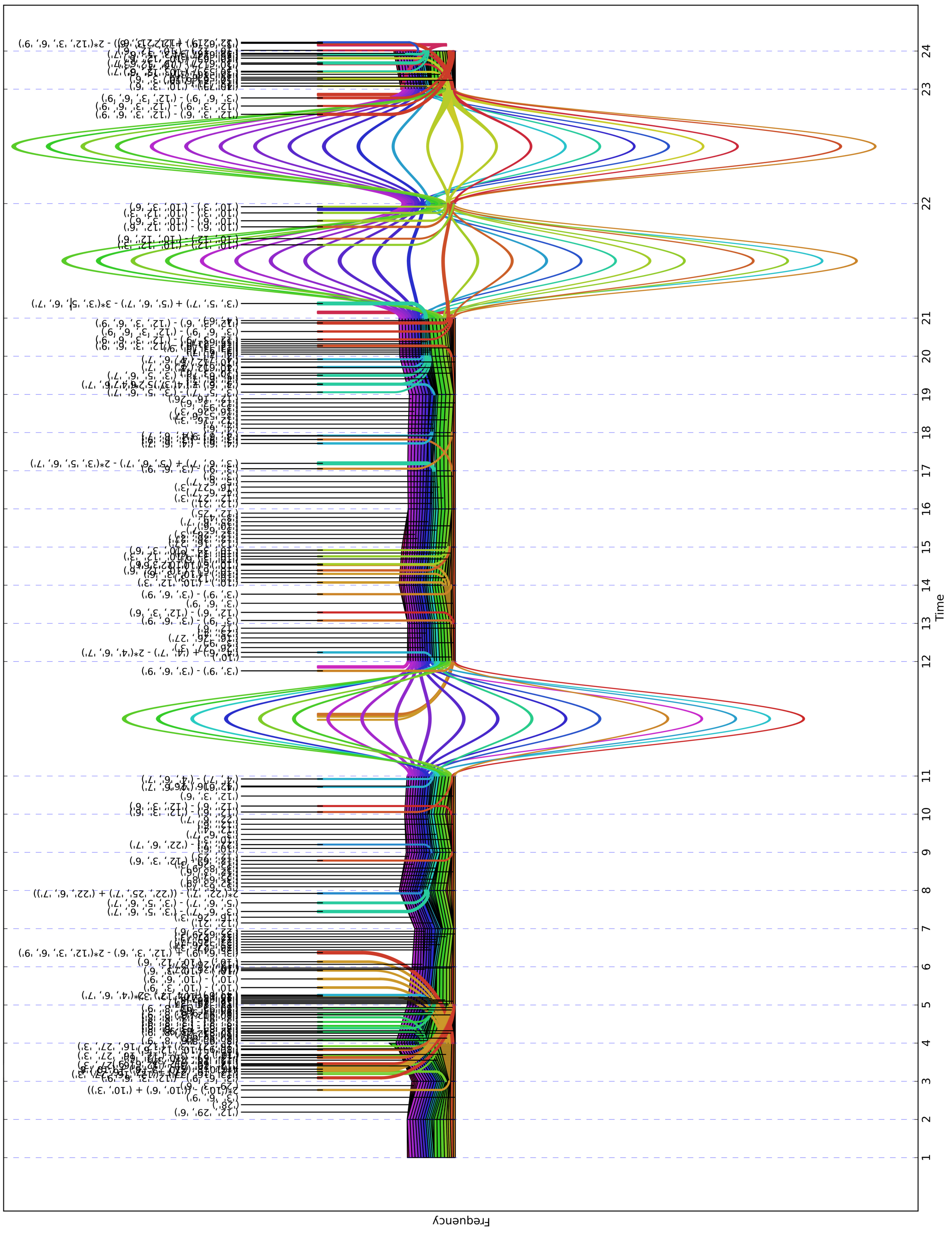


Figure 5.3: Resulting Visualization of a high complexity timeseries

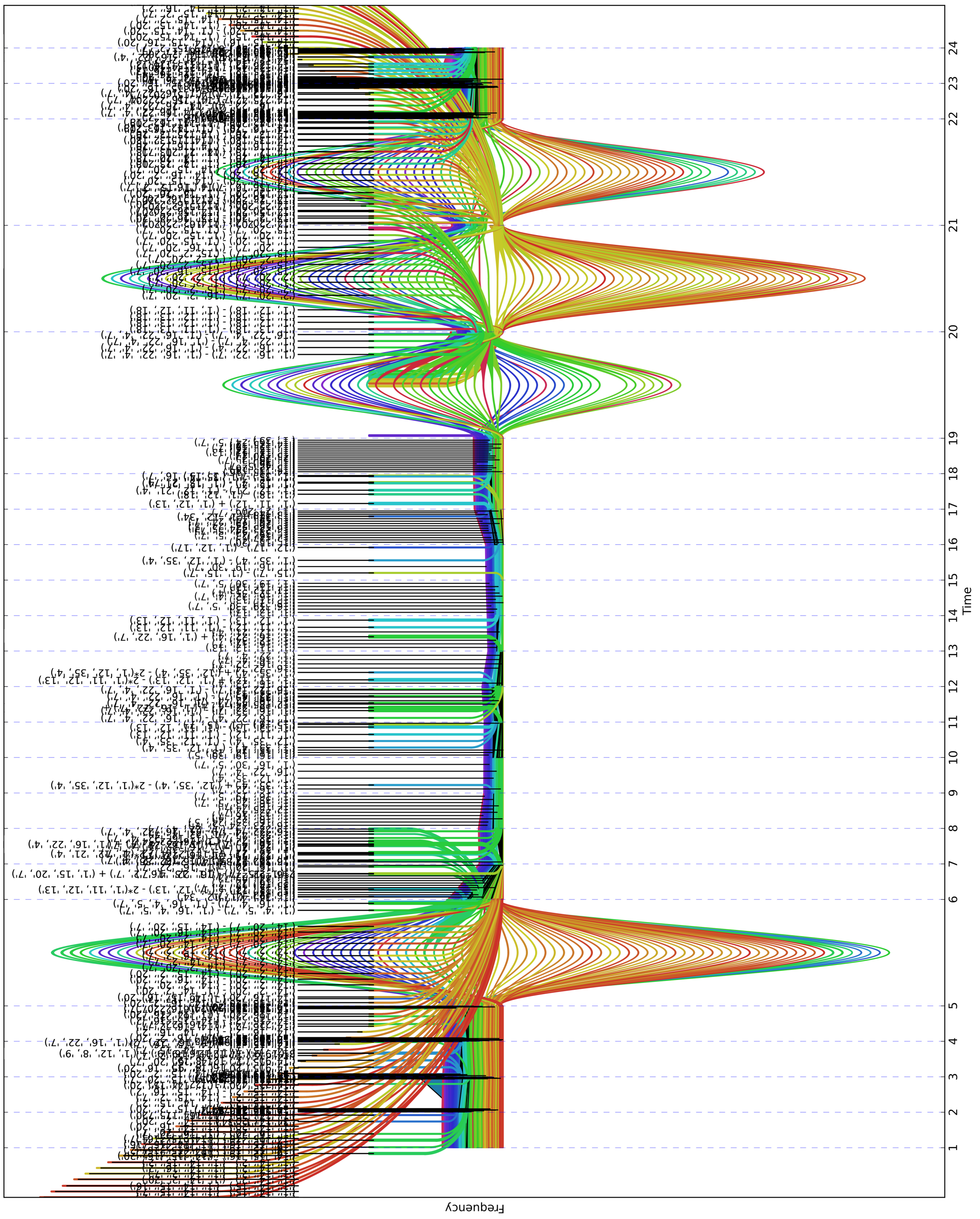


Figure 5.4: Resulting Visualization of a high complexity timeseries

graph	itemsets	max/min itemsets	runtimes	conflicts	conflict times	splits	merges	inbranches	outbranches	largest component
5 none	111	53/30	55.19 sec	8	2	32	32	80	57	9
10 none	36	18/11	18.04 sec	3	2	13	13	29	18	10
15 none	15	9/6	6.04 sec	-	-	2	3	10	8	3
20 none	8	6/4	2.85 sec	-	-	1	1	3	3	4
5 b	150	86/16	1 min 47.38 sec	436	4	61	61	107	104	5
10 b	29	15/6	14.71 sec	-	-	7	6	20	11	9
15 b	14	8/3	9.33 sec	1	1	5	5	16	8	5
20 b	9	5/3	2.91 sec	-	-	1	0	5	3	3
5 io	540	244/35	26 min 9.93 sec	3863	20	175	169	326	320	15
10 io	53	29/17	32.85 sec	17	3	23	23	50	33	5
15 io	27	19/13	6.18 sec	-	-	1	2	7	4	3
20 io	29	15/6	20.98 sec	3	1	15	15	33	20	10

Table 5.5: FIM dataset information & runtimes

Chapter 6

Outlook

During the course of this thesis, we have conducted an extensive analysis into sequence building and conflict avoidance. In addition, we have explored a number of coloring schemes and labeling methods.
The results in Section 5 state that we have adequately achieved our goal of visualizing FIM time-series. However, there are some parts within our proposed design that may be further explored or improved in order to achieve even better results. The following sections describe some ideas or guidelines for possible improvements and extensions.

6.1 Itemset Coloring

The coloring scheme described in Section 4.5 uses the idea that related itemsets should have similar colors. The viewer inherently percieves these similarly colored itemsets as being related, which does achieve our initial goal.
However, there are also drawbacks to using this approach. If we attempt to plot a dataset with many itemsets and many transitions, the itemset color distances get smaller, which, in very complex cases, results in itemset colors becoming almost indistinguishable. Our conflict solver algorithm, described in Section 4.3, places related itemsets that may merge at some point closer together in the y-axis. So not only do we have the problem of indistinguishable itemsets due to very similar colors, but in addition, these indistinguishable itemsets are placed very close together in the plot.
Further work could mean a more extensive study of color spaces. The equidistant hue angle differences we use to achieve separate color tones for the itemsets do not signify equidistant resulting colors, because the hue transitions in the color space are not linear. Perhaps a new scheme could be developed with some effort to achieve truly equidistantly colored itemsets. This improved scheme would then enable a visualization result with more distinctly colored itemsets.

6.2 Conflict Solver Algorithm

Section 4.3 describes the algorithm we wrote to create minimally conflicting itemset sequences. As stated in Section 5.2, the algorithm can be quite slow when working with more complex datasets. This can be problematic, because especially in more complex data the probability of finding conflicts increases.
As further work, we suggest exploring a sequence building scheme that is entirely not based on such extensive and repeated itemset relocations, because such a growing number of elements can quickly escalate the runtime of the entire application.
Figures 4.8 and 4.9 show a situation where a conflict is avoided by using smart reordering of itemsets after they split. Perhaps the idea of reordering itemsets in such a way could be applied to derive a completely new approach of building minimally conflicting sequences. The resulting algorithm would move forward in time until it finds a merging itemset dependency. Subsequently, it would go back and check if any choice ever existed to move the dependent itemsets closer together in the y-axis sequences.

6.3 Absolute Frequency Plotting

Currently, we use relative flow frequency measurements to generate the FIM timeseries visualization. The FIM software, however, provides both relative and absolute FIM measurements. Our application could be extended to visualize relative and absolute FIM timeseries data in parallel. The resulting visualization would enable the viewer to see the overall growth of traffic flows and not just their percentual composition.

6.4 Offline and Online Plotting

So far, we use the FIM software output to plot a complete and static timeseries in an offline plotting scheme. We first collect all existing itemsets in the data and assign distinct colors to the itemsets, build the Time Transition Graph and derive sequences from the FIM dataset files in order to build minimally conflicting y-axis sequences. All these schemes work well, but they do not represent a dynamic and live approach in nature.
Further work could focus on extending the findings and results of this thesis such that the resulting construct is able to accept dynamic inflow of data in an online manner. This would enable real-time troubleshooting of network flows.
When using dynamic input, our application cannot further rely on any of the previously studied schemes. Our current itemset coloring scheme requires full knowledge of all the appearing itemsets, because it uses the number of distinct itemsets to calculate the individual Hue values. If used in an online manner, an entirely new coloring scheme would need to be developed. The color distances between itemsets cannot be calculated, because the number of total itemsets is not initially known.
Neither the static approach, nor our sequence building approach could help in minimizing conflicts, since we have no means of knowing how the final Time Transition Graph will look. But these difficulties are not as insurmountable as they appear. We propose using our Splice timeslot concept, described in Section 4.4, whenever a conflict occurs in subsequent input sequences. This way, all the conflict avoidance, which relies on full information, can be ignored and the remaining challenge would be to derive a new itemset coloring scheme.

Chapter 7

Summary

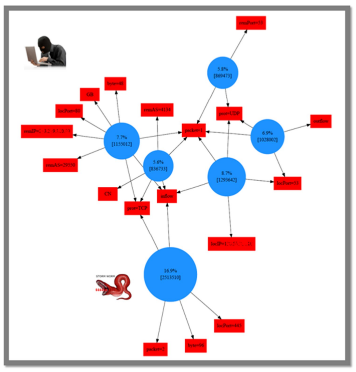
The main goal of this project is to visualize network flow timeseries. To accomplish this, we develop an application in Python, which takes FIM network timeseries data and generates a plot resembling xkcd's congress seat distribution comic [13].
This graphical representation is chosen because it seems well-suited to visualize the complex and dynamic nature of frequent itemset mining results in time.
After an initial itemset placement analysis, we develop an algorithm that produces a minimally conflicting timeseries. This timeseries is then plotted using distinct colors to illustrate changes in frequency and itemset attributes. We used labels to visualize the locations of itemsets in the plot and define flow frequencies between transitioning itemsets.
During the course of this thesis and as described in Section 4, an extensive number of itemset placement, branch design, coloring and labeling schemes have been attempted and analyzed. The visual output of our application proves that we have achieved an adequate way to visualize datasets of up to medium complexity. The sheer number of itemsets contained in more complex datasets makes them virtually impossible to visualize using a stacked plot, because even the most trained eyes have a hard time to keep track of frequency changes of hundreds of simultaneous itemsets in the y-axis, apart from the frequent branch shapes competing for the limited free space between subsequent timeslots.
It is not within the scope of this thesis to influence the frequent itemset mining results. However, our results suggest that if frequent itemset mining output were filtered to only include itemsets of some specific relevance, our plotting application could be used to produce much clearer and more illustrative results.

Appendix A

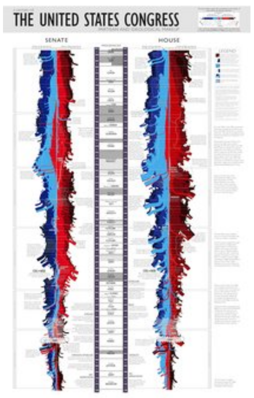
Original Task Description

Visualizing dynamics in FIM timeseries

Visualization of one hour of traffic
Involving 14,916,092 failed connections



Computer networks are kept working by "masters of complexity", who need to know in detail the infrastructure and configuration of a network, extract useful information from diverse and massive monitoring data, and iteratively test and verify their assumptions about the underlying causes of problems, while they are steered solely by their intuition. Effective visualization techniques for network monitoring data can greatly facilitate slow manual data exploration processes by enabling to absorb large amounts of data quickly as a good picture is worth a thousand words. A key challenge in visualizing network monitoring data, like traffic traces, is their massive volume, which can easily reach the order of petabytes when accumulating traffic over time.



In order to effectively explore large volumes of network traffic data, effective visualization techniques that reduce the cognitive burden of the analyst are essential. This thesis will contribute to a project towards building a visual network traffic exploration tool that can be used to create plots that show how network traffic changes over time. Recent work by the Communication Systems Group (CSG) of ETH introduced data analytics and visualization techniques for summarizing and visualizing big network traffic data [1]. The introduced techniques exploit frequent pattern mining, which is a well-known data mining approach. In [1], network traffic is visualized as a special type of graph, called hypergraph. Building on [1], the focus of this thesis is to adapt the visualization method used by Munroe [2] (see image to the left) to FIM timeseries data in order to provide an analysis tool producing intuitive and insightful visualizations.

Tasks:
1. Study related work on visualization and literature on frequent item-set mining.
2. Explore the scheme used by Munroe [2] and find out how it can be adapted to FIM timeseries.
3. Implement the adapted visualization in a program.

We would be happy to provide you more information about the methodology of the project and answer any questions. Please feel free to contact us.

Contacts:
Dr. Bernhard Ager: bager@tik.ee.ethz.ch, ETZ G90
Prof. Eduard Glatz: eglatz@tik.ee.ethz.ch, ETZ H86
Dr. Xenofontas Dimitropoulos: fontas@tik.ee.ethz.ch, ETZ G90

Requirements: Programming basics, knowledge in Python or Java or R or a comparable programming language. This thesis offers practical and theoretical tasks including the development of analysis software.

Related Work:
[1] E. Glatz, S. Mavromatidis, B. Ager and X. Dimitropoulos "Visualizing Big Network Traffic Data using Frequent Pattern Mining and Hypergraphs" Workshop on Internet Visualization (WIV), Nov. 2012.

[2] R. Munroe, "Congress", http://xkcd.com/1127/

Bibliography

[1] Hsv cone. http://i.msdn.microsoft.com/dynimg/IC45291.png, July 2013.

[2] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, volume 22, pages 207–216. ACM, 1993.

[3] Bill Casselman. Bezier curves. http://www.math.ubc.ca/~cass/gfx/bezier.html, July 2013.

[4] Eduard Glatz. Visualizing host traffic through graphs. In *Proceedings of the Seventh International Symposium on Visualization for Cyber Security*, pages 58–63. ACM, 2010.

[5] Eduard Glatz, Stelios Mavromatidis, Bernhard Ager, and Xenofontas Dimitropoulos. Visualizing big network traffic data using frequent pattern mining and hypergraphs. *Computing*, pages 1–12, 2013.

[6] John R Goodall, Wayne G Lutters, Penny Rheingans, and Anita Komlodi. Focusing on context in network traffic analysis. *Computer Graphics and Applications, IEEE*, 26(2):72–80, 2006.

[7] Brian Johnson and Ben Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Visualization, 1991. Visualization'91, Proceedings., IEEE Conference on*, pages 284–291. IEEE, 1991.

[8] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. Blinc: multilevel traffic classification in the dark. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 229–240. ACM, 2005.

[9] Kiran Lakkaraju, William Yurcik, and Adam J Lee. Nvisionip: netflow visualizations of system state for security situational awareness. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 65–72. ACM, 2004.

[10] Stephen Lau. The spinning cube of potential doom. *Communications of the ACM*, 47(6):25–26, 2004.

[11] Florian Mansmann, Fabian Fischer, Daniel A Keim, Stephan Pietzko, and Marcel Waldvogel. Interactive analysis of netflows for misuse detection in large ip networks. In *DFN-Forum Kommunikationstechnologien*, volume 149, pages 115–124, 2009.

[12] Raffael Marty. Afterglow. http://afterglow.sourceforge.net/, July 2013.

[13] Randall Munroe. Xkcd - congress. http://xkcd.com/1127/, October 2012.

[14] Marty Roesch. Snort. http://www.snort.org/, July 2013.

[15] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Access Online via Elsevier, 2006.

[16] Alvy Ray Smith. Color gamut transform pairs. In *ACM Siggraph Computer Graphics*, volume 12, pages 12–19. ACM, 1978.

[17] Loris Degioanni Steven McCanne, Gerald Combs. tcpdump. http://www.tcpdump.org/, July 2013.

[18] Jean-Pierre van Riel and Barry Irwin. Inetvis, a visual tool for network telescope traffic analysis. In *Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, pages 85–89. ACM, 2006.

[19] Xiaoxin Yin, William Yurcik, Michael Treaster, Yifan Li, and Kiran Lakkaraju. Visflowconnect: netflow visualizations of link relationships for security situational awareness. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 26–34. ACM, 2004.