Temporal Multivariate Networks

James Abello[1], Daniel Archambault[2], Jessie Kennedy[3], Stephen Kobourov[4], Kwan Liu Ma[5], Silvia Miksch[6], Chris Muelder[5], and Alexandru Telea[7]

[1] Rutgers University abello@dimacs.rutgers.edu
[2] Swansea University d.w.archambault@swansea.ac.uk
[3] Edinburgh Napier University j.kennedy@napier.ac.uk
[4] University of Arizona kobourov@cs.arizona.edu
[5] University of California at Davis {ma, cwmuelder}@ucdavis.edu
[6] Vienna University of Technology miksch@ifs.tuwien.ac.at
[7] University of Groningen a.c.telea@rug.nl

Abstract. Networks that evolve over time, or dynamic graphs, have been of interest to the areas of information visualization and graph drawing for many years. Typically, its the structure of the dynamic graph that evolves as vertices and edges are added or removed from the graph. In a multivariate scenario, however, attributes play an important role and can also evolve over time. In this chapter, we characterize and survey methods for visualizing temporal multivariate networks. We also explore future applications and directions for this emerging area in the fields of information visualization and graph drawing.

1 Introduction

In previous chapters, this book has primarily concerned itself with visualization methods for static, multivariate graphs. In a static scenario, the network has a number of attributes associated with its elements. These attribute values remain fixed and the challenge is to visualize the interactions between the network(s) and these attributes. Static multivariate graphs could be viewed as graphs with an associated high dimensional data set linked to its elements.

Time is simply another dimension in this multivariate data set that can interact with the vertices, edges, and attribute values of the network. However, humans perceive time differently as we know from our everyday interactions with the physical world. Thus, intuitively, this dimension is often handled differently when supporting the presentation of data that changes over time. Visualization applications and techniques have, and probably should, continue to exploit this fact, allowing for effective visualization methods of temporal multivariate graphs.

In this chapter, we define, characterize, and summarize the data and visualization techniques relating to temporal multivariate networks. Section 2 provides definitions and examples that characterize the networks we address in this chapter. We further refine our definitions of time in section 3. In section 4, we survey representations for dynamic multivariate networks and provide a survey of visualization techniques. We describe the visualization of temporal multivariate networks in the domain of software engineering in section 5. Finally, section 6 describes open problems in this area.

2 Definitions

In a variety of applications, time varying multivariate data can be viewed as evolving information networks whose structure is derived from data attributes (*i.e.* via similarity measures), or it is specified a priory (*i.e.* the flow of information over an underlying network), or it is the result of tracking behavioral statistics (*i.e.* network traces). The network and attributes can be:

– inherent to the fundamental data elements that are taken to be the network vertices *(name, age, gender, income, profession, interests . . .)*
– indicators of the type of relation between the network vertices *(professor of, father of, boss of, colleague of . . .)*
– attribute derived data *(time varying computational mappings from vertex attributes to edge attributes such as "pairs of stocks in markets whose performance has been above a given threshold during a time period")*
– structural derived statistics *(vertex ranks, network centrality, clustering measures . . .)*
– specified contexts in which the data occurs *(Tweets related to a given set of key words for a specified time period)*

In the next subsection, we adapt a model used in software engineering for the purposes of characterizing the types of dynamic, multivariate networks that can be visualized. Then, we propose mathematical formalizations of time varying multivariate networks.

2.1 Structure, Behavior, and Evolution

In a static multivariate network analysis scenario, we have a network structure, consisting of vertices and edges, as well as attributes associated with these vertices and edges. In a time varying scenario, both the graph structure and attribute values can evolve over time. In most cases, we can assume that the network structure at a given moment in time can *influence* how the attribute values evolve and vice versa. These interactions are in some respects very similar to those considered in some software engineering contexts [28]. Thus, we examine time varying multivariate networks appearing in biology and social networks under the lenses of *structure*, *behavior* and *evolution*.

– **Structure**: Pairings between parts or elements of a complex system. Structure mostly relates to the topology of the underlying network at a given time t.
– **Behavior**: Observable activity. Action or reaction of system elements under a given set of stimuli. Behavior mostly refers to the *attributes* associated with the underlying network elements and how they change over time.

	Biology	**Software Engineering**	**Social Networks**
Structure	Biological entities genes and interactions	Modules and couplings	A Twitter community network
Behavior	Gene expression levels	Program trace on the graph	retweet, mention and follower activity
Evolution	Organism development Experimental conditions	Changes to the code	Changes to community structure

Table 1. Examples of structure, behavior and evolution in the domains of biology, software engineering, and social networks.

– **Evolution**: Gradual development of a configuration or pattern over time. Evolution mostly relates to the *structural* changes of the overall underlying network over time.

To illustrate these concepts, we provide examples in Table 1 drawn from the application areas considered in this book: biology, software engineering, and social networks. As an analogy to understand the overarching idea, consider a physical space, such as a building. The *structure* of the building is the construction at a given time. Its *behavior* is how people use the building and its rooms or interact with the physical structure. Its *evolution* may involve bringing in a construction crew to knock down walls and build new ones, modifying the structure of the building as a result of observable decay in the physical infrastructure or as a response to ergonomical complaints of its occupants.

Note that in a time varying multivariate network scenario, both *behavior* and *evolution* can operate on each other. This generalization of the dynamics differs from the original software engineering approach where evolution could only influence behavior. An example of *evolution* influencing *behavior* in a biology scenario is when an experimental condition causes network structure to change or evolve, affecting in turn gene expression levels (*i.e.* behavior). An example of *behavior* influencing *evolution* in a social network scenario is when the interaction between actors in a social network (*i.e.* their behavior) causes ties to break or form, thus, evolving the network.

2.2 Formal Definitions of Temporal Multivariate Networks

To incorporate some of the main characteristics of time varying multivariate data we propose the following mathematical formalization of a time varying information data set [1].

The implicit assumption is that "time" is a universal reference "axis" with respect to which the data is being tracked. For now we assume that "time" is a totally ordered set, but as we will discuss later, it can also be taken to be a partially ordered set. A *time varying information data set* $G_{V,t}$ on a set of vertices V consists of a sequence $\{F(G_t)\}_{t>=0}$ where F is a multivariate function $F : R^h \times R^h \times R \to R^k$ and at each time t, G_t denotes the following collection of 4-tuples:

$$G_t = \{< V_{(x,y)}, V_x, V_y, t >: V_{(x,y)} = F(V_x, V_y, t)\} \tag{1}$$

V_x , V_y , and $V_{(x,y)}$ are vectors in R^h and R^k respectively and (x, y) is a pair of vertices in V. The underlying *information network structure* is determined by those pairs of vertices (x, y) in $V \times V$ for which there exists a four tuple

$$< V_{(x,y)}, V_x, V_y, t >$$

in some G_t.

The F cumulative behavior of $G_{V,t}$ up to and including t is the entry wise sum:

$$F_{<=t}(G_V) =< \sum_{j=0}^{t}(V_{(x,y)}), \sum_{j=0}^{t}(V_x), \sum_{j=0}^{t}(V_y) >$$

where the sum is taken over all the quadruples $< V_{(x,y)}, V_x, V_y, j >$ in G_j for $j <= t$.

A time varying information data set $G_{V,t}$ *evolves towards a network* G, if there exists a time $t > 0$ such that the *underlying network* of the union of G_j for $j >= t$ is isomorphic to G.

3 Refining Our Models and Definitions for Time

Time itself is an inherent data dimension that is central to the tasks of revealing trends and identifying patterns and relationships in the data. Time and time-oriented data have distinct characteristics that make it worthwhile to treat such data as a separate data type [2, 3]. Due to the importance of time-oriented data, its structure has been studied in numerous scientific publications (e.g., [2, 11, 41]). As proposed by Aigner *et al.* [3], we divide the aspects of time-oriented data into general aspects required to adequately model the time domain as well as hierarchical organization of time and definition of concrete time elements, also called human-made abstractions.

The **general aspects** are scale, scope, arrangement, and viewpoints.

1. Scale: ordinal vs. discrete vs. continuous. As a first perspective, we look at time from the scale along which elements of the model are given. In an *ordinal* time domain, only relative order relations are present (e.g., before, after). In *discrete* domains temporal distances can also be considered. Time values can be mapped to a set of integers which enables quantitative modelling of time values (e.g., quantifiable temporal distances). Discrete time domains are based on a smallest possible unit and they are the most commonly used time model in information systems. *Continuous* time models are characterized by a possible mapping to real numbers, *i.e.*, between any two points in time, another point in time exists (also known as dense time).

2. Scope: point-based vs. interval-based. Secondly, we consider the scope of the basic elements that constitute the structure of the time domain. *Point-based* time domains can be seen in analogy to discrete Euclidean points in space, *i.e.*, having a temporal extent equal to zero. Thus, no information is given about the region between two points in time. In contrast to that, *interval-based* time domains relate to subsections of time having a temporal extent greater than zero. This aspect is also closely related to the notion of granularity, which will be discussed later.

3. Arrangement: linear vs. cyclic. As the third design aspect, we look at the arrangement of the time domain. Corresponding to our natural perception of time, we mostly consider time as proceeding *linearly* from the past to the future, *i.e.*, each time value has a unique predecessor and successor. In a *cyclic* organization of time, the domain is composed of a set of recurring time values (e.g., the seasons of the year). Hence, any time value A is preceded and succeeded at the same time by any other time value B (e.g., winter comes before summer, but winter also succeeds summer).

4. Viewpoint: ordered vs. branching vs. multiple perspectives. The fourth subdivision is concerned with the views of time that are modelled. *Ordered* time domains consider things that happen one after the other. On a more detailed level, we might also distinguish between totally ordered and partially ordered domains. In a totally ordered domain only one thing can happen at a time. In contrast to this, simultaneous or overlapping events are allowed in partially ordered domains, *i.e.*, multiple time primitives at a single point or overlapping in time. A more complex form of time domain organization is the so-called *branching* time. Here, multiple strands of time branch out and allow the description and comparison of alternative scenarios (e.g., in project planning). In contrast to branching time where only one path through time will actually happen, *multiple perspectives* facilitate simultaneous (even contrary) views of time.

The **human-made abstractions** are granularities, time primitives, and determinacy.

1. Granularity and calendars: none vs. single vs. multiple. To tackle the complexity of time and to provide different levels of granularity, useful abstractions can be employed. Basically, *granularities* can be thought of as (human-made) abstractions of time in order to make it easier to deal with time in every-day life (like minutes, hours, days, weeks, months). More generally, granularities describe mappings from time values to larger or smaller conceptual units. If a granularity and calendar system is supported by the time model, we characterize it as *multiple* granularities. Besides this complex variant, there might be a *single* granularity only (e.g., every time value is given in terms of milliseconds) or *none* of these abstractions are supported (e.g., abstract ticks).

2. Time primitives: instant vs. interval vs. span. These time primitives can be seen as an intermediary layer between data elements and the time domain. Basically, time primitives can be divided into anchored (absolute) and unanchored (relative) primitives. *Instant* and *interval* are primitives that belong to the first group, *i.e.*, they are located on a fixed position along the time domain. In contrast to that, a *span* is a relative primitive, *i.e.*, it has no absolute position in time. Instants are a model for single points in time, intervals for ranges between two points in time, and spans a duration (of intervals) without a fixed position.

3. Determinacy: determinate vs. indeterminate. Uncertainty is another important aspect when considering time-oriented data. If there is no complete or exact information about time specifications or if time primitives are converted from one granularity to another, uncertainties are introduced and have to be dealt with. Therefore, the *determinacy* of the given time specification needs to be considered. A determinate specification is present when there is complete knowledge of all temporal aspects.

4 Survey of Representations and Algorithms

While static graphs arise in many applications, dynamic processes naturally give rise to graphs that evolve through time. Such dynamic processes can be found in software engineering, telecommunications traffic, computational biology, and social networks, among others. Dynamic graph drawing addresses the problem of effectively presenting such relationships as they change over time.

Static graph visualization has a long and venerable history, while dynamic graph visualization is a relatively newer field. But even though temporal graph representations are more recent, the variety of representations is still large, and there are a number of studies concerning the drawing of dynamic graphs [20, 5, 16]. As a dynamic graph can be thought of as a sequence of edge sets on the same set of vertices, it can be treated similarly to visualizing multiple relationships on the same data set. There are nearly as many ways to represent dynamic or multivariate networks as there are graph representations: simple node-link diagrams, directed graphs, clustered graphs, hierarchical and multi-level representations, matrix representations, spatialized (map-like) representations, etc. Dynamic graphs can be visualized with *global views*, where all the graphs are displayed at once, *merged views*, where all the graphs are agglomerated together, and with *sequenced views*, where timesteps are plotted individually, and either small multiples or animated morphing (fading in/out vertices and edges that appear/disappear) are used to compare timesteps.

It is worth noting here that it makes a difference whether the temporal visualization aims to show individual timesteps (e.g., collaboration between researchers in each individual year) or cumulative (e.g., new collaborations from current year are added to the already accumulated collaboration graph). Similarly, there is a difference between offline and online temporal visualization. In the offline setting, we are given all data in advance, whereas in the online setting the changes are happening on the fly. Most existing algorithms address the problem of offline dynamic graph drawing, where the entire sequence of graphs to be drawn is known in advance. This gives the layout algorithm information about future changes in the graph, which makes it possible to optimize the layouts generated across the entire sequence (e.g., the algorithm can leave enough space in anticipation of placing vertices that appear later in the sequence). Less work

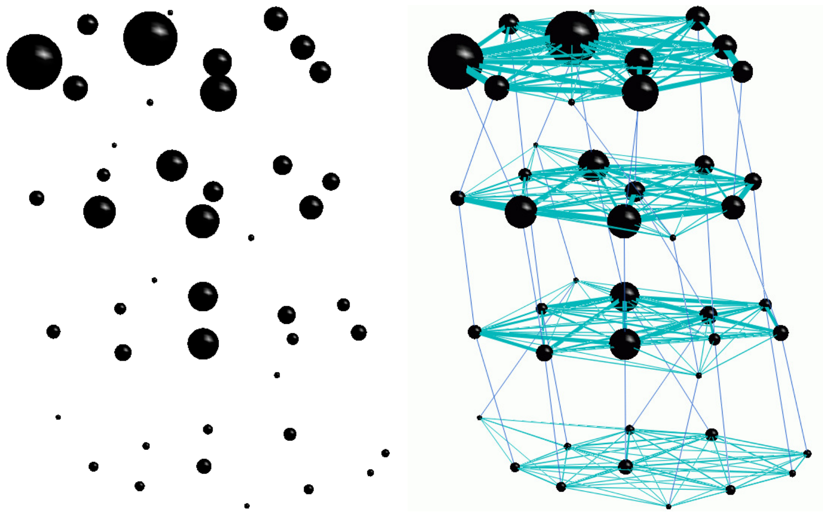


Fig. 1. A dynamic graph can be interpreted as a larger graph made of connecting graphs in adjacent timesteps [34].

has been done in the online setting, where the graph sequence to be laid out is not known in advance.

By far the most common method for visualizing dynamic graphs is to view the graph as a series of node-link diagrams whether as a sequence or all at once; see Fig. 1 and Fig. 2. Thus many dynamic graph layouts are based on static graph layout algorithms, which are used to lay out each timestep. Efforts to improve the quality and stability of the layouts lead to the development of full-fledged dynamic graph layout algorithms. Some visualization approaches eschew the node-link representation to better show temporal evolution, as in streamline representations and dynamic maps. There has also been work in summarizing the temporal evolution of dynamic graphs in more static representations. And finally, there are a number of analytic algorithms and approaches that have been extended to dynamic network visualization.

4.1 Static Graph Layouts

Force-directed layouts (e.g., Fruchterman-Reingold [44], LinLog [77], Kamada-Kawai [62]) arrange graphs by iteratively refining the positions of vertices to incrementally reduce an energy function. This function varies between algorithms, but generally has the property that it is a function of the distances between vertices and the weights of the edges between them. These layouts are simple, and generally considered aesthetic, but they do not generally scale well to large or dense graphs.

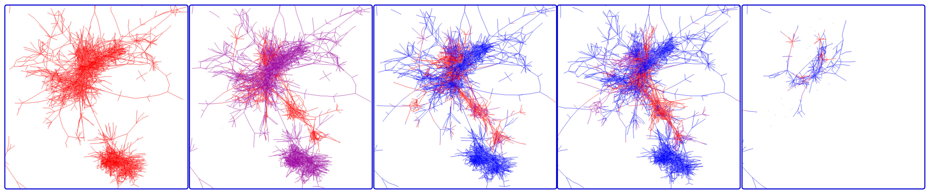


Fig. 2. Snapshots of the call-graph of a program as it evolves through time, extracted from CVS logs. Vertices start out red. As time passes and a vertex does not change it turns purple and finally blue. When another change is affected, the vertex again becomes red. Note the number of changes between the two large clusters and the break in the build on the last image [24].

More efficient layout algorithms use a multi-scale approach, such as the work of Cohen [23], the Fast Multipole Multilevel Method (FM^3) [52], and the Graph dRawing with Intelligent Placement (GRIP) algorithm [46]. These algorithms start by laying out a small approximation of a graph, then progressively laying out finer approximations of the graph, until the entire original graph is laid out. These algorithms generally use far fewer iterations, and thus perform far better than traditional force-directed approaches, while still producing similar results.

Even faster graph layout algorithms are available in the form of algebraic layouts, such as Algebraic Multigrid Computation of Eigenvectors (ACE) [64], High Dimensional Embedding (HDE) [53], the work of Brandes and Pich [18], or the Maxent method [48]. These calculate layouts directly using linear algebra techniques rather than using iterative force calculations. This generally makes them very fast. Clustering-based layouts have also been shown to be fast, as in the case of the treemap layout [74] or space-filling curve layout [73]. These methods work by clustering the graph in a preprocessing step and then mapping the clustering to the screen to define the layout itself.

4.2 Dynamic Graph Layouts

In dynamic graph drawing the goal is to maintain a nice layout of a graph that is modified via operations such as inserting/deleting edges and inserting/deleting vertices. A key property of in many real-world applications, where dynamic graphs naturally arise, is that the difference between any two timesteps is generally assumed to be incremental: that is, a small change relative to the size of the graph. If the change between timesteps is too large, then it is often more effective to treat them as separate, static networks. When visualizing evolving and dynamic graphs, two of the most important criteria to consider are:

1. *readability, or quality* of the individual layouts, which depends on aesthetic criteria such as display of symmetries, uniform edge lengths, and minimal number of crossings; and

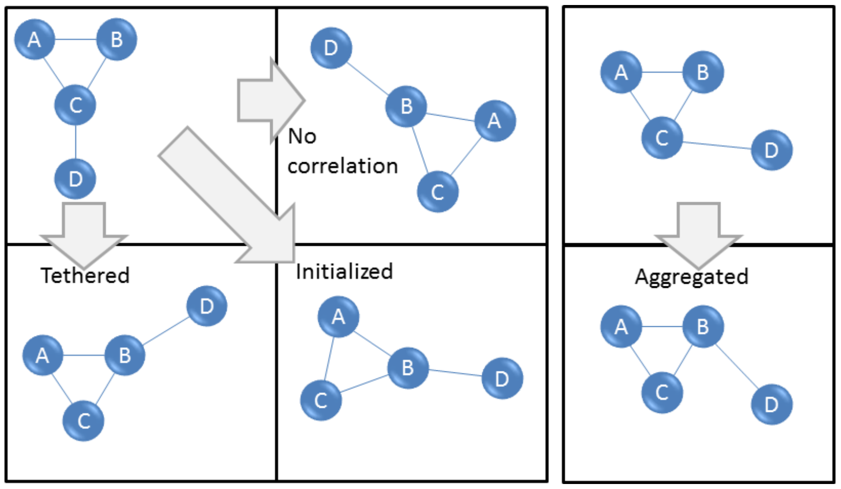


Fig. 3. Mental map preservation has been a forefront topic in dynamic graph layout. The level of layout stability can vary between approaches. Incremental approaches can range from having no correlation between timesteps to using the previous timestep as initialization to anchoring or tethering some vertices to previous positions. The most stable layouts agglomerate all timesteps together, but these could result in poor layouts at each timestep.

2. *mental map preservation, or stability* in the series of layouts, which can be achieved by ensuring that vertices and edges that appear in consecutive graphs in the series, remain in the same location.

There is an inherent trade-off between the stability and quality of any dynamic graph layout, as restricting the movement of vertices could make it impossible to achieve high quality layout of the individual timesteps. In fact, these two criteria are often contradictory and many dynamic graph layout approaches explore different ways of balancing stability and quality; see Fig. 3. At one end are quality optimizing layouts with little to no correlation between timesteps, and at the other are fixed layouts where the vertices never move, even if the layout is not ideal for any given timestep. Anchored layouts lie somewhere between the two extremes, where some vertices are fixed while others are allowed to move; see survey of Brandes *et al.* [16].

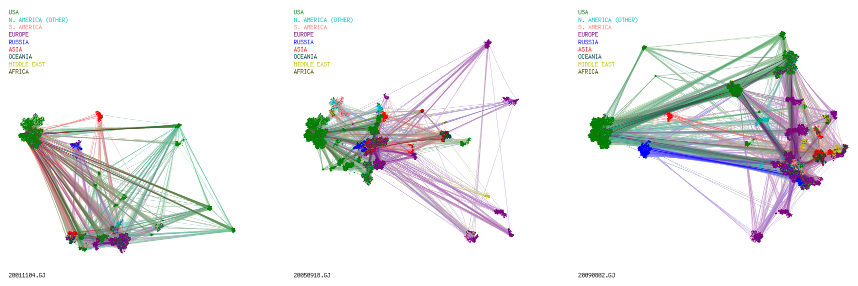
The input to this problem is a series of graphs defined on the same underlying set of vertices. As a consequence, nearly all existing approaches to visualization of evolving and dynamic graphs are based on extensions of static graph layouts, usually based on a force-directed method. The simplest methods just initialize a force directed layout with the previous layout of the timestep, as in [10, 37], but this offers little guarantees for stability as nothing actually constrains the motion of vertices. Early examples of this can be dated back to North's DynaDAG [78], where the graph is not given all at once, but incrementally. Most of these early approaches, however, are limited to special classes of graphs and usually do not scale to graphs over a few hundred vertices. `TGRIP` could handle the larger graphs that appear in the real-world. It was developed as part of a system that keeps track of the evolution of software by extracting information about the program stored within a CVS version control system [24]. Such tools allow programmers to understand the evolution of a legacy program: Why is the program structured the way it is? Which programmers were responsible for which parts of the program during which time periods? Which parts of the program appear unstable over long periods of time? `TGRIP` was used to visualize inheritance graphs, program call-graphs, and control-flow graphs, as they evolve over time; see Fig. 2.

Aggregate layouts such as in [70], are among the approaches that guarantee good stability by computing one layout for an aggregate graph made up of the union of all timesteps. Brandes and Corman [14] describe a system for visualizing network evolution in which both fixes vertices in constant locations, and uses a 3D super-graph, by showing each modification in a separate layer of a 3D representation with vertices common to two layers represented as columns connecting the layers. Thus, mental map preservation is achieved by pre-computing good locations for the vertices and fixing the position throughout the layers. An explicit tradeoff between quality and stability can also be provided as in the GraphAEL system [35]. There a super-graph of all timesteps is created and links between occurrences of the vertices in neighboring timesteps are added; see Fig. 1. By changing the weights of these inter-timestep edges one can emphasize stability (make inter-timestep edges very strong) or readability (make inter-timestep edges very weak). Such approaches [35, 36, 32, 39] generally use modified versions of traditional static layout algorithms directly, but often induce high memory usage and complexity because all timesteps are loaded at once. They are also only applicable to offline graph drawing, as the entire data range is needed at the beginning.

However, the most popular approach in recent years is to compute time varying network layouts by adding additional constraints that anchor vertices to their positions in the previous timestep [67, 42, 43]. These techniques work by adding some additional forces to the force direction calculation, but provide a good balance of cost, layout quality, and stability, and can be tuned by adjusting the anchor weights. These algorithms can also address the online dynamic graph drawing problem, as it is not necessary that the graph sequence is not known in advance. Brandes and Wagner adapt the force-directed model to dynamic graphs using a Bayesian framework [19]. An algorithm for visualizing dynamic social networks is discussed in [70]. Frishman and Tal consider dynamic drawing of clustered graphs [42] and of general graphs [43]. Brandes *et al.* have also performed a quantitative evaluation of the tradeoffs between layout quality and stability for these different classes of layouts [17].

There are also dynamic graph visualization approaches based on clustering. Kumar and Garland describe a method of animating clusters through time [65]. In this approach, a stratified, abstracted version of the graph is used, where the vertices are topologically sorted into a treelike structure (before layout) in order to expose interesting features.

Sallaberry *et al.* [94] cluster every timestep individually, associate the clusters across time, and use the space-filling curve approach to render each timestep; see Fig. 4. Pre-computing the clusters is computationally expensive. Hu *et al.* [58] propose a method based on a geographical metaphor to visualize a summary of clustered dynamic graphs. It also relies on clustering and aims to keep clusters stable over time.



(a) 2002-10-27 (b) 2005-09-18 (c) 2009-08-02

Fig. 4. Large networks add additional challenges in computational cost and perceptual limits (images from [94])

4.3 Animation Versus Small Multiples

Often, dynamic graph visualizations animate the transitions between node-link diagrams of timesteps [78, 29, 35, 49, 13, 43]. In these animations, vertices dynamically appear, disappear and move to produce readable layouts at each timestep. Diehl and Görg [29] and Görg *et al.* [49] consider graphs in a sequence to create smoother transitions. Animations as a means to convey an evolving underlying graph have also been used in the context of software evolution [24] and scientific literature visualization [35]. Creating smooth animation between changing sequences of graphs is addressed using spectral graph visualization in [15]. When using the animation/morphing approach, it is possible to change the balance between readability of individual graphs and the overall mental map preservation, as in the system for Graph Animations with Evolving Layouts, GraphAEL [35, 40]. Applications of this framework include visualizing software evolution [24], social networks analysis [9], and the behavior of dynamically modifiable code [30].

Robertson *et al.* [89] evaluate the effectiveness of three trend visualization techniques. The results indicate that animation, often enjoyable and exciting, is not always well suited to data analysis. The other common alternative for visualizing multiple timesteps is to statically place them next to each other as small multiples [101]. This eases the comparison of distant timesteps but only a small area can be devoted to each timestep, which reduces the readability of each graph. Cerebral [8] is a system that uses a biologically guided graph layout and incorporates experimental data directly into the graph display. Small multiple views of different experimental conditions and a data-driven parallel coordinates view enable correlations between experimental conditions to be analyzed at the same time that the data is viewed in the graph context. This combination of coordinated views allows the biologist to view the data from many different perspectives simultaneously.

Empirical studies to compare the advantages and drawbacks of these approaches ("Animation" vs. "Small Multiples") have been performed by Archambault *et al.* [7] as well as Farrugia and Quigley [38]. And even more recently, Rufiange *et al.* have developed a hybrid approach that lets the user interactively combine or switch between animations, small multiples, and plots that explicitly indicate what has changed [90].

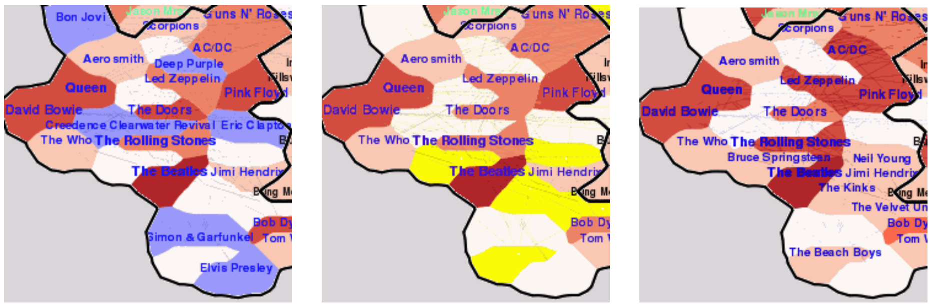
4.4 Mental Map Preservation

Preserving the mental map, or layout stability, is a major focus in many dynamic node-link representations approaches [17, 43, 58, 65, 92]. Even though several experiments have been performed to examine the effect of preserving the mental map in dynamic graphs visualization the results are mixed. The results of [87] were quite surprising. The experiment found that the most effective visualizations were the extreme ones, *i.e.*, the ones with very low or high mental map preservation, while visualizations with medium preservation were less effective [87]. With large networks, stability becomes even more important, but so does "motion coherency". Even small motions on each vertex are too much to perceive if they are chaotic, but if vertices move coherently, they can be perceived as a single group [94]. In a series of papers Archambault and Purchase evaluate various approaches for dynamic graph visualization and consider how they affect mental map preservation [7, 4, 6], also summarized in a recent survey [5].

4.5 Alternative Representations

Using maps to visualize non-cartographic data has been considered in the context of spatialization [97]. Map-like visualization using layers and terrains to represent text document corpora dates back to 1995 [103]. The problem of effectively conveying change over time using a map-based visualization was studied by Harrower [54]. More recently, Mashima *et al.* [68] use the GMap framework [57] to visualize dynamic graphs with the geographic map metaphor; see Fig. 5.

Also related is work on visualizing subsets of a set of items. Areas of interest in a UML diagram can be highlighted using a deformed convex hull [22]. Isocontours-based bubblesets can be used to depict multiple relations defined on a set of items [25]. Automatic Euler diagrams, which show the grouping of subsets of items by drawing contiguous regions around them have also been considered [96]. Apart from differences in the algorithms used to generate regions, all of these approaches create regions that overlap with each other (unlike the strict map metaphor where regions do not overlap).



(a) (b) (c)

Fig. 5. Evolution in the top 250 most popular bands on Last.fm: showing three consecutive snapshots from an animation, focusing on area that corresponds to Rock. An animated version is also available online at http://www2.research.att.com/ yifanhu/TrendMap/. (a) Highlighting in blue areas where artists are about to disappear: Bon Jovi, Deep Purple, Elvis, Simon & Garfunkel, CCR, and Eric Clapton. (b) Highlighting in yellow the areas where new artists are about to appear. (c) An image after new artists appear, showing the newcomers: Bruce Springsteen, Neil Young, The Kinks, and The Beach Boys.

Bezerianos *et al.* [12] describe a multivariate network visualization system, GraphDice, which uses a plot matrix to navigate multivariate graphs.

4.6 Static Temporal Plots

One visualization approach for summarizing dynamic large graphs is to directly represent time as an axis. The most direct way to do this is to take 2D node-link diagrams and extend them to 3D with time as the third dimension; see Fig. 1). However, 3D can be cluttered, and has occlusion and other perceptual limitations. An interesting 2D approach based on parallel coordinates was proposed by Burch *et al.* [21], where vertices are ordered and positioned on several vertical parallel lines, and directed edges connect these vertices from left to right. The graph of each timestep is thus displayed between two consecutive vertical axes.

Such representations can get quite cluttered for larger graphs. Rather than depicting the entire network over time, another approach is to abstract the network into clusters and to show how they evolve. WilmaScope [31] does this in 3D by representing the clusters as tubes. An increasingly popular way to visualize the evolution dynamic clusters is the use of storylines [27, 63, 76, 79, 88, 98]. Most of these works reference hand-drawn diagrams such as XKCD's movie narrative charts [76] as inspiration, in which entities are represented as lines which move together when in the same group and separate when they are not. Plotweaver [80] is a tool to aid in semi-automatic generation of storyline plots, but it still requires significant user interaction. The works of Ogawa *et al.* [79] and Tanahashi et el. [98] aim to automate the process; see Fig. 6. However, producing

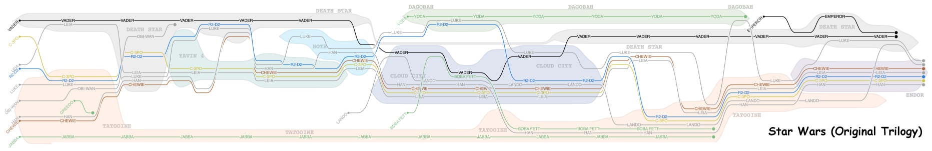


Fig. 6. Storylines can succinctly summarize the evolution of a dynamic graph (from [98]).

good results with these algorithms is computationally expensive, as they do not scale well to large data sets. To apply storyline techniques to dynamic graphs, an intermediary step of dynamic clustering must be derived [88, 94].

4.7 Dynamic Graph Analytics

Another relevant avenue of research has been the extension of analytic algorithms to dynamic graphs. Finding a partition of the vertices of a static graph according to its structure is a well studied problem; see survey by Schaeffer [95]. But clustering a dynamic graph is a less studied problem. One possibility is to use a global clustering, which is computed by applying a static clustering to an aggregate combination of all the timesteps in the dynamic graph. This creates a clustering which is on average good, but which can not capture the evolution of the network. Others have developed dynamic graph clustering algorithms in the context of visualization applications that track clusters across timesteps, allowing their memberships to evolve over time. Several approaches try to modify the clustering incrementally as the network changes [51, 50, 93]. Hu *et al.* [58] use a similar approach, but apply a heuristic to accelerate this process. Sallaberry *et al.* [94], on the other hand, cluster each timestep separately and then use Jaccard index to track the clusters across time.

Different from top-down methods above, there are also several bottom-up approaches that start with a single vertex and its immediate context. Additional relevant vertices and connections are revealed only on demand, based on graph structure or specialized degree-of-interest functions that can incorporate semantic importance or users' interaction histories [72, 55, 33, 45, 102, 26]. Recently, such approaches have been extended to dynamic graphs by incorporating temporal histories, and applying relevancy filtering to a storyline-based representation [75].

5 Applications to Software Engineering

Temporal multivariate networks play a key role in many aspects of software engineering (SE). To understand the related challenges, we need to understand

1. the *tasks* that they support in software engineering;
2. the *characteristics* of SE data leading to such graphs.

This section covers the above two points. For a full overview of applications of multivariate dynamic graphs in SE, we refer to Chapter **??**. Our focus here is more technical. Specifically, we aim to characterize SE graphs from the perspective of time modeling (Section 3), and the variability axes (of types) (Section 4). This in turn better explains the rationale behind the visual designs presented in Chapter **??**, and also why it is challenging to use visualization techniques developed for other types of temporal multivariate graphs to handle SE graphs.

Tasks Software engineering activities cover the entire software product lifetime, starting with requirement gathering, followed by architecting, design, implementation (coding), testing, release, and ending with maintenance. Graphs are created and used in all these stages, as shown in Table 2. As software systems change during their lifecycle, all above graphs are by nature time-dependent. Moreover, SE graphs involve elements and relations spanning several of the above activities. For example, in reverse engineering, we encounter graphs that link software test results with source code (and developers), class diagrams, and requirements.

Actions	**Examples of graphs**
Requirements	Requirements *vs* tasks *vs* stakeholders[60] UML use-case diagrams[91]
Architecting	System structure (layering, dataflows, component interactions)[100]
Design	UML component and package diagrams[91] UML class, activity diagrams[91]
Coding	Call, inheritance, type-use, and include graphs[28]
Testing	Type-instance graphs, control flow graphs[85] Resource allocation graphs[71].
Release	Deployment graphs[81], UML deployment diagrams[91]
Maintenance	Developer networks, code duplication graphs[69]

Table 2. Examples of multivariate temporal graphs in SE.

Data characteristics Temporal multivariate SE graphs have several characteristics which make their computation, efficient manipulation, and above all *understanding* very challenging. Below we outline the main such aspects.

Size: Depending on their type, SE graphs range from a few tens of elements (UML diagrams and developer networks) to hundreds of thousands (call graphs) or even millions of elements (control-flow graphs of large programs). The static call graph of the Mozilla Firefox browser (a medium-sized system as compared to large telecom or banking software) has, for example, over 500K edges [56]. Certain topology constraints exist for some graphs, *e.g.* class hierarchies are, usually, trees, and architecture dependencies form a directed acyclic graph. However, in the general case, little can be said about the global properties of SE graphs.

For instance, a call graph can be cyclic (or not), and can have a widely varying distribution of number of edges per vertex depending on application type.

Attributes: Each vertex and edge in a SE graph typically has several attributes. These describe both static and dynamic properties of the entity encoded by that vertex or edge. For instance, annotated semantic graphs (ASGs) for C++ programs have tens of such attributes [99]. Computing software quality metrics easily adds tens of other metrics [66]. Attribute types span a wide spectrum: numerical, categorical, text, and binary. Attribute *types* are key to effective program understanding. For instance, the C++ ASG in [99] contains around two hundred different vertex-attribute types that encode the different properties of the annotated C++ grammar. Being able to visually distinguish between different types is essential, *e.g.* for detecting the presence of specific design or execution patterns. Missing values are possible *e.g.* due to limitations of program analysis tools or due to incomplete program coverage for execution monitoring tools.

Dynamics: Graphs describing human aspects, such as developer activity, change slowly, given the continuous nature of software evolution [69]. However, other SE graphs exhibit different dynamics. For instance, in program execution graphs, large changes can occur in short time periods and few changes in other longer time periods. Dynamics is present both at the *structure* level (*e.g.* changes of a call graph topology as the program is run for different inputs or as code changes during maintenance), and also at the *attribute* level (*e.g.* different runtime metrics measured at static component level for different program executions).

Time modeling: Time is, formally, modeled as a discrete quantity, since both execution and changes of software code occur at discrete, moments. Time has a linear nature, describing the order of execution of program instructions or the order of changes in a repository. However, time can be seen as fully ordered or branching (Section 2). The branching case occurs *e.g.* when considering execution of multi-threaded programs or analyzing development activity of a repository with multiple branches. Both point-based and interval-based models are used, often interchangeaby, for the same analysis. For instance, a *version* in a software repository can refer to the moment when it was committed, but also to the time interval between this commit time and the next change of the same artifact.

Scale: Software understanding occurs on multiple levels of detail and following both a top-down and bottom-up process [85]. Hence, one needs to (visually) analyze software at several levels of detail or *scales*. SE graphs offer several natural scales, given their hierarchical, or compound, nature (Chapter **??**), *e.g.* function-class-file-folder or the structure given by a function call stack. Yet, several aspects make constructing efficient and effective multiscale SE-graph visualizations hard. Firstly, SE graphs are huge. The few above-mentioned levels of detail do not offer enough granularity to *automatically* simplify large graphs to levels where they can be displayed in an understandable manner. Automatically computing *additional* levels of detail is hard – for instance, what should be the meaning of an artifact larger than a file, but smaller than a folder? Secondly, many program understanding tasks require showing both fine-grained detail *and* coarse-scale structure in the same view. For instance, to debug a crash, we need to see the entire call stack, from the finest-grained instruction which caused the fault up to the coarse-level components which scope the fault. Finally, software is by nature *abstract*. As such, finding effective visual metaphors (for both the spatial graph embedding and attribute mapping) is challenging.

6 Open Problems

Although significant progress has been achieved in the design of visualization methods and tools for exploring multivariate temporal networks, several important open challenges remain. This section outlines a selection of challenges which are relevant to a broad subset of applications involving such graphs. Throughout the discussion, we use the notation introduced in Section 2.2.

6.1 Attribute dimensionality

As outlined in Section 5, SE graphs are high-variate, *i.e.*, have many attributes for each vertex or edge. Existing visualization techniques can simultaneously show a few (up to 3) attributes per graph element, by mapping these to shape, size, texture, color, and shading. However, this solution scales poorly for graphs of hundreds of thousands of elements. Separately, even for small graphs (hundreds of elements), showing tens of attributes per element is an open challenge. Parallel coordinate plots partially address this quest [59]. An interesting adaption hereof clusters graph vertices based on attribute values, and links the resulting icicle plots to a table-lens-like visualization of the edge attributes, to highlight attribute correlations [86]. Dimensionality reduction projects a set of high-dimensional attributes into $\mathbb{R}^2$ or $\mathbb{R}^3$ so that similarities between the original attributes are reflected in the low-dimensional distance [61, 82, 83]. Although such approaches scale well computationally for large sample counts [84], it is hard to visualize both attribute similarity *and* graph structure in the same embedding. Other approaches use interactive brushing, attribute selection, and linked views. However, none of the above methods fully enables users to correlate structure with attributes, and attributes among themselves, for highly-variate graphs.

6.2 Capturing patterns

In many use-cases, showing a picture of the (changing) graph is not sufficient, even when this picture is clutter and overlap-free. For instance, consider the task of locating *patterns* in the graph. Patterns are specific configurations of vertices and edges (topology) and attribute values which capture events of interest. Patterns are typically problem-dependent, and have a certain variability in both structure and attribute values. Consider finding a 'multithreading refactoring event' in a software code base: This would involve finding similar code fragments in a graph G_t, which describe serial code, and finding that they have been replaced by functionally-identical multithreaded code in the following revision G_{t+1} of the code base. Even the simpler 'design patterns' [47], well known and used in object-oriented software design, are hard to detect and visualize. The underlying reasons are twofold. First, patterns involve, by definition, *several* vertices, edges, and attribute values, so they correspond to portions of a graph visualizations. However, existing graph visualization techniques have difficulties in showing such data subsets in canonical ways, *i.e.*, in ways that make their visual detection easy. Secondly, patterns have a certain variability. Besides making automatic detection hard, this also implies that their graph visualizations will exhibit a necessary variability, which makes their visual detection hard. Finally, visually detecting dynamic patterns is very challenging – if animation is used, this poses high demands on the user's visual memory; if static visualizations are used, inherently dynamic patterns may be hard to grasp.

6.3 Data size

Large dynamic graphs involve large sets of vertices and edges and/or many sampling moments when the graph is captured. This implies many sample points taken over the domain of function F (Equation 1). Large graphs are hard to embed in a low-dimensional space ($\mathbb{R}^2$ or $\mathbb{R}^3$) so that the graph structure is easy to discern. This basic graph-drawing problem becomes one or two orders of magnitude larger for dynamic graphs. The data size problem becomes even larger for high-variate graphs.

It is insightful to consider how data size relates to the other challenges. Formally, we could argue that dynamic multivariate graphs (and their patterns) can be efficiently and effectively depicted using existing visualization methods, for small graphs. Hence, we could use subsampling, like in scientific data visualization, to reduce the graph size prior to visual exploration. To preserve features or patterns of interest, data-adaptive subsampling could be used. The main obstacle here is that we still lack a comprehensive theory for subsampling graphs *and* categorical attributes. As such, existing solutions addressing data size currently have to rely on aggregation and simplification algorithms and heuristics that are problem, scale, and even dataset-specific.

7 Summary and Conclusions

In this chapter, we characterized temporal multivariate graphs in terms of structure and time. We presented common terminology for discussing temporal multivariate graphs, a survey of existing techniques, focusing on software engineering applications, and a collection of open problems. We hope that this common terminology, data characterization, and organization of existing and future work will help foster further research in the emerging area of dynamic multivariate graph visualization.

References

1. J. Abello, S. Hadlak, H. Schumann, and H. Schulz. A modular degree-of-interest specification for the visual analysis of large dynamic networks. *IEEE Transactions on Visualization and Computer Graphics*, 2014. in press.
2. W. Aigner, S. Miksch, H. Schumann, and C. Tominski. *Visualization of Time-Oriented Data*. Springer, London, 2011.
3. N. Andrienko and G. Andrienko. *Exploratory Analysis of Spatial and Temporal Data: A Systematic Approach*. Springer, Berlin, 2006.
4. D. Archambault and H. C. Purchase. The mental map and memorability in dynamic graphs. In H. Hauser, S. G. Kobourov, and H. Qu, editors, *Proc. of the IEEE Pacific Visualization Symposium*, pages 89–96. IEEE, 2012.
5. D. Archambault and H. C. Purchase. The "map" in the mental map: Experimental results in dynamic graph drawing. *International Journal of Human-Computer Studies*, 71(11):1044 – 1055, 2013.
6. D. Archambault and H. C. Purchase. Mental map preservation helps user orientation in dynamic graphs. In *Graph Drawing (GD'12)*, volume 7704 of *LNCS*, pages 475–486, 2013.
7. D. Archambault, H. C. Purchase, and B. Pinaud. Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *IEEE Transactions on Visualization and Computer Graphics*, 17(4):539–552, 2011.
8. A. Barsky, T. Munzner, J. Gardy, and R. Kincaid. Cerebral: Visualizing multiple experimental conditions on a graph with biological context. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1253–1260, 2008.
9. M. Bastian, S. Heymann, and M. Jacomy. Gephi: an open source software for exploring and manipulating networks. *International AAAI Conference on Weblogs and Social Media*, pages 361–362, 2009.
10. S. Bender-deMoll and D. A. McFarland. The art and science of dynamic network visualization. *Journal of Social Structure*, 7(2), 2006.
11. C. Bettini, S. Jajodia, and S. X. Wang. *Time Granularities in Databases, Data Mining, and Temporal Reasoning*. Springer, Berlin, 2000.
12. A. Bezerianos, F. Chevalier, P. Dragicevic, N. Elmqvist, and J.-D. Fekete. Graphdice: A system for exploring multivariate social networks. *Computer Graphics Forum*, 29(3):863–872, 2010.
13. K. Boitmanis, U. Brandes, and C. Pich. Visualizing internet evolution on the autonomous systems level. In *Graph Drawing (GD'07)*, volume 4875 of *LNCS*, pages 365–376. Springer, 2008.
14. U. Brandes and S. R. Corman. Visual unrolling of network evolution and the analysis of dynamic discourse. In *Proc. of the IEEE Symposium on Information Visualization*, pages 145–151, 2002.
15. U. Brandes, D. Fleischer, and T. Puppe. Dynamic spectral layout with an application to small worlds. *Journal of Graph Algorithms and Applications*, 11(2):325–343, 2007.
16. U. Brandes, N. Indlekofer, and M. Mader. Visualization methods for longitudinal social networks and stochastic actor-oriented modeling. *Social Networks*, pages 291–308, June 2011.
17. U. Brandes and M. Mader. A quantitative comparison of stress-minimization approaches for offline dynamic graph drawing. In *Graph Drawing (GD'11)*, volume 7034 of *LNCS*, pages 99–110. Springer, 2012.

18. U. Brandes and C. Pich. An experimental study on distance-based graph drawing. In *Graph Drawing*, pages 218–229, 2008.
19. U. Brandes and D. Wagner. A Bayesian paradigm for dynamic graph layout. In *Graph Drawing (GD'97)*, pages 236–247, 1998.
20. J. Branke. Dynamic graph drawing. In M. Kaufmann and D. Wagner, editors, *Drawing Graphs*, volume 2025 of *Lecture Notes in Computer Science*, pages 228–246. Springer, 2001.
21. M. Burch, C. Vehlow, F. Beck, S. Diehl, and D. Weiskopf. Parallel edge splatting for scalable dynamic graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2344–2353, 2011.
22. H. Byelas and A. Telea. Visualization of areas of interest in software architecture diagrams. In *ACM SoftVis'06*, pages 105–114, 2006.
23. J. D. Cohen. Drawing graphs to convey proximity: An incremental arrangement method. *ACM Transactions On Computer-Human Interaction*, 4(3):197–229, 1997.
24. C. Collberg, S. G. Kobourov, J. Nagra, J. Pitts, and K. Wampler. A system for graph-based visualization of the evolution of software. In *ACM SoftVis'03*, pages 77–86, 2003.
25. C. Collins, G. Penn, and S. Carpendale. Bubble sets: Revealing set relations with isocontours over existing visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1009–1016, 2009.
26. T. Crnovrsanin, I. Liao, Y. Wuy, and K.-L. Ma. Visual recommendations for network navigation. In *Proc. of the 13th Eurographics / IEEE - VGTC conference on Visualization*, EuroVis'11, pages 1081–1090, Aire-la-Ville, Switzerland, Switzerland, 2011. Eurographics Association.
27. W. Cui, S. Liu, L. Tan, C. Shi, Y. Song, Z. Gao, H. Qu, and X. Tong. Textflow: Towards better understanding of evolving topics in text. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2412–2421, 2011.
28. S. Diehl. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer, Berlin, 2010.
29. S. Diehl and C. Görg. Graphs, they are changing. In *Graph Drawing (GD'02)*, volume 2528 of *LNCS*, pages 23–30. Springer, 2003.
30. B. Dux, A. Iyer, S. K. Debray, D. Forrester, and S. G. Kobourov. Visualizing the behavior of dynamically modifiable code. In *IWPC*, pages 337–340, 2005.
31. T. Dwyer. Extending the wilmascope 3d graph visualisation system — software demonstration. In S.-H. Hong, editor, *APVIS*, volume 45 of *CRPIT*, pages 39–45. Australian Computer Society, 2005.
32. T. Dwyer and D. R. Gallagher. Visualising changes in fund manager holdings in two and a half-dimensions. *Information Visualization*, 3:227–244, 2004.
33. N. Elmqvist and J.-D. Fekete. Hierarchical Aggregation for Information Visualization: Overview, Techniques, and Design Guidelines. *IEEE Transactions on Visualization and Computer Graphics*, 16(3):439–454, 2009.
34. C. Erten, P. J. Harding, S. G. Kobourov, K. Wampler, and G. Yee. Exploring the computing literature using temporal graph visualization. In *Electronic Imaging 2004*, pages 45–56, 2004.
35. C. Erten, P. J. Harding, S. G. Kobourov, K. Wampler, and G. V. Yee. GraphAEL: Graph animations with evolving layouts. In *Graph Drawing (GD'03)*, volume 2912 of *LNCS*, pages 98–110. Springer, 2004.
36. C. Erten, S. Kobourov, V. Le, and A. Navabi. Simultaneous graph drawing: layout algorithms and visualization schemes. *Journal of Graph Algorithms and Applications*, 9(1):165–182, 2005.

37. M. Farrugia and A. Quigley. Cell phone mini challenge: Node-link animation award animating multivariate dynamic social networks. In *IEEE Visual Analytics Science and Technology*, pages 215 –216, oct. 2008.
38. M. Farrugia and A. Quigley. Effective temporal graph layout: A comparative study of animation versus static display methods. *Journal of Information Visualization*, 10(1):47–64, 2011.
39. K.-C. Feng, C. Wang, H.-W. Shen, and T.-Y. Lee. Coherent time-varying graph drawing with multi-focus+context interaction. *IEEE Transactions on Visualization and Computer Graphics*, 2011.
40. D. Forrester, S. G. Kobourov, A. Navabi, K. Wampler, and G. V. Yee. Graphael: A system for generalized force-directed layouts. In *Graph Drawing (GD'03)*, pages 454–464, 2004.
41. A. U. Frank. Different Types of "Times" in GIS. In M. J. Egenhofer and R. G. Golledge, editors, *Spatial and Temporal Reasoning in Geographic Information Systems*, pages 40–62. Oxford University Press, New York, NY, USA, 1998.
42. Y. Frishman and A. Tal. Dynamic drawing of clustered graphs. In *Proc. of the IEEE Symposium on Information Visualization*, pages 191–198, Washington, DC, USA, 2004. IEEE Computer Society.
43. Y. Frishman and A. Tal. Online dynamic graph drawing. *IEEE Transactions on Visualization and Computer Graphics*, 14:727–740, 2008.
44. T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, 21(11):1129–1164, 1991.
45. G. W. Furnas. Generalized fisheye views. In *Human Factors in Computing Systems CHI*, pages 16–23, 1986.
46. P. Gajer and S. G. Kobourov. GRIP: Graph drawing with intelligent placement. In *Graph Drawing (GD'00)*, pages 222–228, London, UK, 2001. Springer-Verlag.
47. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
48. E. R. Gansner, Y. Hu, and S. C. North. A maxent-stress model for graph layout. In *Proc. of the IEEE Pacific Visualization Symposium*, pages 73–80, 2012.
49. C. Görg, P. Birke, M. Pohl, and S. Diehl. Dynamic graph drawing of sequences of orthogonal and hierarchical graphs. In *Graph Drawing (GD'04)*, volume 3383 of *LNCS*, pages 228–238. Springer, 2005.
50. R. Görke, P. Maillard, C. Staudt, and D. Wagner. Modularity-driven clustering of dynamic graphs. In *Proc. of the 9th international conference on Experimental Algorithms*, SEA'10, pages 436–448, Berlin, Heidelberg, 2010. Springer-Verlag.
51. R. Grke, T. Hartmann, and D. Wagner. Dynamic graph clustering using minimum-cut trees. In F. Dehne, M. Gavrilova, J.-R. Sack, and C. Tth, editors, *Algorithms and Data Structures*, volume 5664 of *Lecture Notes in Computer Science*, pages 339–350. Springer Berlin Heidelberg, 2009.
52. S. Hachul. *A Potential-Field-Based Multilevel Algorithm for Drawing Large Graphs*. PhD thesis, Universitaet zu Koeln, 2002.
53. D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. In *Graph Drawing (GD'02)*, pages 207–219. Springer-Verlag, 2003.
54. M. Harrower. Tips for designing effective animated maps. *Cartographic Perspectives*, 44:63–65, 2003.
55. J. Heer and D. Boyd. Vizster: visualizing online social networks. In *Proc. of the IEEE Symposium on Information Visualization*, pages 32–39, 2005.
56. H. Hoogendorp, O. Ersoy, D. Reniers, and A. Telea. Extraction and visualization of call dependencies for large C/C++ code bases: A comparative study. In *Proc. ACM VISSOFT*, pages 137–145, 2009.

57. Y. Hu, E. R. Gansner, and S. G. Kobourov. Visualizing graphs and clusters as maps. *IEEE Computer Graphics and Applications*, 30(6):54–66, 2010.
58. Y. Hu, S. G. Kobourov, and S. Veeramoni. Embedding, clustering and coloring for dynamic maps. In *Proc. of the IEEE Pacific Visualization Symposium*, pages 33–40, 2012.
59. A. Inselberg. *Parallel Coordinates: Visual Multidimensional Geometry and Its Applications*. Springer, 2009.
60. C. Jaramillo, A. Gelbukh, and F. Isaza. Pre-conceptual schema: a conceptual-graph-like knowledge representation for requirements elicitation. In *Proc. MICAI*, pages 27–37, 2006.
61. P. Joia, F. V. Paulovich, D. Coimbra, J. A. Cuminato, and L. G. Nonato. Local affine multidimensional projection. *IEEE Transactions on Visualization and Computer Graphics*, 17:2563–2571, 2011.
62. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Inf. Process. Lett.*, 31(1):7–15, 1989.
63. N. W. Kim, S. K. Card, and J. Heer. Tracing genealogical data with timenets. In *Proc. of the International Conference on Advanced Visual Interfaces*, (AVI '10), pages 241–248, New York, NY, USA, 2010. ACM.
64. Y. Koren, L. Carmel, and D. Harel. ACE: A fast multiscale eigenvectors computation for drawing huge graphs. *Proc. of the IEEE Symposium on Information Visualization*, pages 137–145, 2002.
65. G. Kumar and M. Garland. Visual exploration of complex time-varying graphs. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):805–812, 2006.
66. M. Lanza and R. Marinescu. *Object-Oriented Metrics in Practice - Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Springer, 2006.
67. K. A. Lyons. Cluster busting in anchored graph drawing. In *CASCON*, pages 7–17, 1992.
68. D. Mashima, S. G. Kobourov, and Y. Hu. Visualizing dynamic data with maps. *IEEE Transactions on Visualization and Computer Graphics*, 18(9):1424–1437, 2012.
69. T. Mens and S. Demeyer. *Software Evolution*. Springer, 2008.
70. J. Moody, D. McFarland, and S. BenderdeMoll. Dynamic network visualization. *American Journal of Sociology*, 110(4):1206–1241, 2005.
71. S. Moreta and A. Telea. Multiscale visualization of dynamic software logs. In *Proc. Eurovis*, pages 11–18, 2007.
72. T. Moscovich, F. Chevalier, N. Henry, E. Pietriga, and J.-D. Fekete. Topology-Aware Navigation in Large Networks. In *SIGCHI conference on Human Factors in computing systems*, pages 2319–2328, 2009.
73. C. Muelder and K.-L. Ma. Rapid graph layout using space filling curves. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1301–1308, 2008.
74. C. Muelder and K.-L. Ma. A treemap based method for rapid layout of large graphs. In *Proc. of the IEEE Pacific Visualization Symposium*, pages 231–238, 2008.
75. C. W. Muelder, T. Crnovrsanin, and K.-L. Ma. Egocentric storylines for visual analysis of large dynamic graphs. In *Proc. of 1st IEEE Workshop on Big Data Visualization (BigDataVis)*, pages 56–62, Oct 2013.
76. Xkcd #657: Movie narrative charts. http://xkcd.com/657, dec. 2009.
77. A. Noack. An energy model for visual graph clustering. *Lecture Notes in Computer Science*, 2912:425–436, Mar. 2004.

78. S. C. North. Incremental layout in DynaDAG. In *Graph Drawing (GD'95)*, volume 1027 of *LNCS*, pages 409–418. Springer, 1996.
79. M. Ogawa and K.-L. Ma. Software evolution storylines. In *Proc. of the International Symposium on Software Visualization*, (SoftVis '10), pages 35–42, New York, NY, USA, 2010. ACM.
80. V. Ogievetsky. Plotweaver xkcd/657 creation tool, March 2009. https://graphics.stanford.edu/wikis/cs448b-09-fall/FPOgievetskyVadim.
81. A. Orso, J. Jones, and M. J. Harrold. Visualization of program-execution data for deployed software. In *Proc. ACM SOFTVIS*, pages 67–75, 2003.
82. F. Paulovich, D. Eler, J. Poco, C. Botha, R. Minghim, and L. G. Nonato. Piece wise Laplacian-based projection for interactive data exploration and organization. *Computer Graphics Forum*, 30(3):1091–1100, 2011.
83. F. V. Paulovich, L. G. Nonato, R. Minghim, and H. Levkowitz. Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. *IEEE Transactions on Visualization and Computer Graphics*, 14(3):564–575, 2008.
84. F. V. Paulovich, C. Silva, and L. G. Nonato. Two-phase mapping for projecting massive data sets. *IEEE Transactions on Visualization and Computer Graphics*, 16:1281–1290, 2010.
85. S. L. Pfleeger and J. M. Atlee. *Software Engineering: Theory and Practice (4th ed.)*. Prentice Hall, 2009.
86. A. Pretorius and J. van Wijk. Visual inspection of multivariate graphs. *Computer Graphics Forum*, 27(3):967–974, 2008.
87. H. Purchase and A. Samra. Extremes are better: Investigating mental map preservation in dynamic graphs. In *Proc. of the 5th International Conference on Diagrammatic Representation and Inference (Diagrams 2008)*, volume 5223 of *LNCS*, pages 60–73. Springer, 2008.
88. K. Reda, C. Tantipathananandh, A. Johnson, J. Leigh, and T. Berger-Wolf. Visualizing the evolution of community structures in dynamic social networks. *Computer Graphics Forum*, 30(3):1061–1070, 2011.
89. G. Robertson, R. Fernandez, D. Fisher, B. Lee, and J. Stasko. Effectiveness of animation in trend visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14:1325–1332, 2008.
90. S. Rufiange and M. J. McGuffin. DiffAni: Visualizing dynamic graphs with a hybrid of difference maps and animation. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2556–2565, 2013.
91. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 2nd edition, 2004.
92. P. Saffrey and H. Purchase. The "mental map" versus "static aesthetic" compromise in dynamic graphs: A user study. In *Proc. of the 9th Australasian User Interface Conference (AUIC2008)*, pages 85–93, 2008.
93. B. Saha and P. Mitra. Dynamic algorithm for graph clustering using minimum cut tree. In *SDM*, pages 581–586. SIAM, 2007.
94. A. Sallaberry, C. W. Muelder, and K.-L. Ma. Clustering, visualizing, and navigating for large dynamic graphs. In *Graph Drawing (GD'12)*, LNCS, pages 487–498. Springer, 2013.
95. S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.
96. P. Simonetto, D. Auber, and D. Archambault. Fully automatic visualisation of overlapping sets. *Computer Graphics Forum*, 28(3):967–974, 2009.

97. A. Skupin and S. I. Fabrikant. Spatialization methods: a cartographic research agenda for non-geographic information visualization. *Cartography and Geographic Information Science*, 30:95–119, 2003.
98. Y. Tanahashi and K.-L. Ma. Design considerations for optimizing storyline visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2679–2688, 2012.
99. A. Telea and L. Voinea. An interactive reverse engineering environment for large-scale C++ code. In *Proc. ACM SOFTVIS*, pages 67–76, 2008.
100. A. Telea, L. Voinea, and H. Sassenburg. Visual tools for software architecture understanding: A stakeholder perspective. *IEEE Software*, 27(6):46–53, 2010.
101. E. R. Tufte. *Envisionning Information.* Graphics Press, 1990.
102. F. van Ham and A. Perer. Search, Show Context, Expand on Demand: Supporting Large Graph Exploration with Degree-of-Interest. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):953–960, 2009.
103. J. A. Wise, J. J. Thomas, K. Pennock, D. Lantrip, M. Pottier, A. Schur, and V. Crow. Visualizing the non-visual: spatial analysis and interaction with information from text documents. In *Proc. of the IEEE Symposium on Information Visualization*, pages 51–58, 1995.