Spectral Measures of Distortion for Change Detection in Dynamic Graphs

Luca Castelli Aleardi[1], Semih Salihoglu[2], Gurprit Singh[3] Maks Ovsjanikov[1]

Abstract We propose a novel framework for detecting, quantifying and visualizing changes between two snapshots of a dynamic network. Unlike existing approaches, which can be sensitive to noise, and are often based on heuristics, we show how a theoretically-justified, inherently multi-scale notion of change, or distortion, can be defined and computed using spectral graph-theoretic tools. Our primary observation is that informative, robust and multi-scale measures of change can be obtained by computing a real-valued function (which we call *the distortion function*) on the nodes of the input graph, via the optimization of a pre-defined *distortion energy* in a provably optimal way. Based on extensive tests on a wide variety of networks, we demonstrate the ability of our approach to highlight the evolution of the network in an informative and multi-scale manner.

1 Introduction

Many real-world evolving systems can be conveniently encoded as dynamic graphs (e.g., biological networks, where the connectivity can represent the evolution of protein interactions or collaboration networks, where new links and nodes are added over time [1, 2, 27]). A key challenge in the visualization and analysis of dynamic networks is capturing the structural changes in the graph in a robust and efficient way. In particular, a fundamental problem is to define principled measures of change or difference between graphs that can be used to highlight the modified regions, while not being sensitive to noise. Unfortunately, in many cases these two objectives are contradictory, especially in the presence of large networks with many structural changes, where simple measures result in highly noisy highlighted regions that are difficult to interpret. In this context, several methods [15, 17, 18] have been proposed to produce informative summaries of dynamic graphs. However, as discussed

[1]LIX, Ecole Polytechnique, France · [2]University of Waterloo, Canada · [3]Max Planck Institute for Informatics, Germany.

in Section 3, they typically suffer from the lack of precise control over the type of changes that are considered and the *scale* at which they are computed, e.g., local changes of individual vertices vs. global distortions to the structure of entire regions in the graph. Furthermore, these existing techniques typically do not allow to capture and highlight only *the primary* areas of change, and can result in cluttered visualizations. Finally, most existing methods are based on heuristics and often lack formal guarantees of global optimality.

In this paper, we propose an efficient, flexible and multi-scale framework for detecting and analyzing changes in dynamic graphs, in the online setting, where only the current and the previous timestamp information are known. Given a pair of graphs, our framework is based on defining and computing an optimal *node distortion function*, which associates to each vertex a real value that quantifies the change associated with this vertex across the two graphs (with higher values corresponding to larger changes). We use the term *distortion* to emphasize the fact that our function should measure the most dramatic changes in graph structure, at a given scale. To this end, we leverage concepts from spectral graph theory to both ensure robustness against noise (i.e. disparate and unrelated, local changes), by controlling the scale at which the changes are computed, and provide theoretical guarantees on the global optimality of the highlighted changes.

1.1 Related Work

The problem of capturing changes in dynamic graphs has recently attracted a lot of attention and a variety of different techniques have been proposed, including the use of animation (time-to-time mapping) or timeline (time-to-space mapping) methods (see [4], for a taxonomy of methods for dynamic graph visualization).

Network visualization. Many existing works [6, 7, 10, 11, 15, 17, 23, 30, 29] make use of a large variety of different graph drawing techniques in order to update the layout while preserving the mental map [23], which is considered as the main requirement to obtain good readability and facilitate graph exploration [3]. For instance, two interesting approaches combine the notion of vertex ages [17] and node pinning weights [15] with force-directed layouts to produce visualizations of dynamic graphs: the main idea consists to reduce node displacement via a mechanism based on the associated distortion at a vertex (which is close to simulated annealing [12]). The approach described in [15], where node pinning weights are associated with a distance-to-modification measure, is more sophisticated: it integrates force-directed layouts with a coarsening phase, and can exploit GPU parallelism for efficiency. Spectral methods, which have been extensively used for graph visualization for several decades [20, 22, 25], have recently been adapted to deal with dynamic networks [6, 10, 30]. While one approach is to extend the classic spectral layout to the case of dynamic graphs [6], another possible use of spectral methods is in the layout post-processing [10, 30, 31] that can be combined with arbitrary static layouts. In this case, node positions or even additional data such as grouping or temporal penalties can reduce node displacements.

Distorted Region Detection. There are relatively few works, in the context of network visualization that consider the problem of detecting and highlighting the regions with the most relevant changes and evolution. A notable exception is [18], where the authors address this problem by introducing a measure of relevance for vertices and weighted edges (called *strength*), and makes use of a threshold filtering based on a sliding time-window in order to efficiently visualize the most relevant evolving regions in the graph. Detecting relevant regions is a crucial ingredient that can be combined with other tools for the visualization of large networks [19].

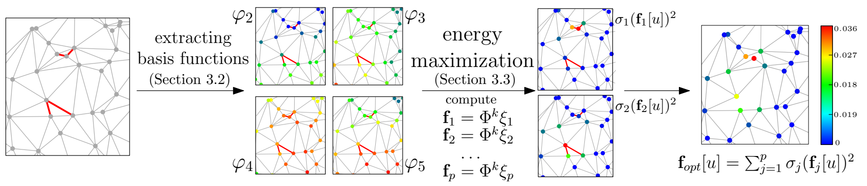
Our Motivation and Contribution. Our work is inspired by previous techniques [15, 17, 18] that define various distortion measures aimed at capturing changes between graphs. These distortion measures are then combined either with force-directed methods [16, 21, 28] for graph visualization, or integrated in detecting most *distorted regions*. In context of the former, a distortion function allows controlling the displacement of vertices by manipulating the forces acting on them, while in the latter it can be used to filter vertices by, e.g., thresholding their values. The main distinguishing characteristics of our framework is that it provides precise control over both the type of distortions and the scale at which changes are computed, with theoretical optimality guarantees for the computed distortion function. None of the existing methods enable these features in a single coherent framework.

2 Proposed Framework

Preliminaries: A dynamic network is defined by a sequence of graphs $\{G_1,\dots,G_T\}$ where $G_i = (V_i, E_i)$ represents a snapshot at time i. As in previous works [10, 11, 29, 15, 17, 18, 30], we assume that we are given the correspondence between nodes in two consecutive snapshots. The nodes and edges of each G_i are allowed to be associated with positive weights: we denote by $d_i(u), w_i(u,v)$ the weight of vertex u and of edge (u,v) in G_i, respectively. In the simplest case, $d_i(u)$ is the degree of vertex u in G_i, and, for unweighted graphs, $w_i(u,v) = 1$ if $(u,v) \in E_i$ and 0 otherwise. We then define a diagonal matrix D_i, s.t. $D_i(u,u) = d_i(u)$ and a symmetric matrix W_i, s.t. $W_i(u,v) = w_i(u,v) = w_i(v,u)$. Finally, we define the weighted Laplacian matrix L_{G_i} as: $L_{G_i} = U_i - W_i$, where U_i is a diagonal matrix, s.t. $U_i(u,u) = \sum_v w_i(u,v)$.

Our framework aims at detecting the structural changes between two instances G_i and G_{i+1} of a dynamic network (graph) by computing a *distortion function* $f : V_i \longrightarrow \mathbb{R}$, such that $f(v)$ should: 1) quantify the changes at vertex v between G_i and G_{i+1}, with higher values corresponding to larger changes and $f(v) = 0$ corresponding to no changes, 2) be multi-scale to reflect changes in the neighborhood of v at any given scale, 3) be efficiently computable in practice. The second property is especially important to detect and highlight evolving *regions* in graphs and to gain resilience to noisy, dispersed changes throughout the graph.

General overview: To compute the *distortion function* that satisfies the above three criteria, we first define a *distortion energy* $E(G_i, G_{i+1}, f)$ that assigns a scalar score



extracting
basis functions
(Section 3.2)
φ_2
φ_3
φ_4
φ_5
energy
maximization
(Section 3.3)
compute
$\mathbf{f}_1 = \Phi^k \xi_1$
$\mathbf{f}_2 = \Phi^k \xi_2$
$\dots$
$\mathbf{f}_p = \Phi^k \xi_p$
$\sigma_1(\mathbf{f}_1[u])^2$
$\sigma_2(\mathbf{f}_2[u])^2$
$\mathbf{f}_{opt}[u] = \sum_{j=1}^{p} \sigma_j (\mathbf{f}_j[u])^2$
0.036
0.027
0.018
0.019
0

Fig. 1: Algorithm overview: given a graph and its modification, (removed edges are marked in red), we first construct a set of basis functions, then pick a distortion energy and compute its maximizers in the linear span of the basis. Finally, we compute the optimal distortion function that smoothly reflects the changes at each vertex from very high (in red) to least affected vertices (in blue).

to an arbitrary real-valued function $f : V_i \longrightarrow \mathbb{R}$. We then construct *a multi-scale family* $\mathscr{F}$ of functions. Finally, we compute the *optimal distortion function* f_{opt} which corresponds to the *maximizer* of $E(G_i, G_{i+1}, f)$ for given G_i, G_{i+1}, and s.t. f lies within the family $\mathscr{F}$ (see Fig. 1 for an illustration). We then use f_{opt} to visualize the changes across G_i, G_{i+1}, by color-coding the vertices, and to detect the most distorted regions as described in Section 3.

Our motivation for adopting this pipeline is that it allows us to 1) precisely control the types of changes the approach should be sensitive to via the choice of distortion energy, 2) control the *scale* of the computed changes via the choice of family $\mathscr{F}$ and 3) provide theoretical guarantees, making sure that the computed maximizer of $E(G_i, G_{i+1}, f)$ is globally optimal.

2.1 Distortion Energies

The main considerations when choosing a distortion energy are first to ensure that it is sensitive to the changes that are meaningful in the context of the evolution of G_i and second, it should be easy to optimize, so that computing the maximizer f of $E(G_i, G_{i+1}, f)$ w.r.t. f can be done efficiently in practice. To this end, we consider the following two distortion energies (notations from Sec. 2):

$$\begin{aligned} E_{\text{vertex diff}}(G_i, G_{i+1}, f) &= \frac{\sum_u f(u)^2 (d_i(u) - d_{i+1}(u))^2}{\sum_u f(u)^2 d_i(u)}, \\ E_{\text{edge diff}}(G_i, G_{i+1}, f) &= \frac{\sum_{u,v} (f(u) - f(v))^2 (w_i(u,v) - w_{i+1}(u,v))^2}{\sum_u f(u)^2 d_i(u)}. \end{aligned} \tag{1}$$

Intuitively, these two energies, $E_{\text{vertex diff}}$ and $E_{\text{edge diff}}$, are sensitive to the absolute *changes* (increase or decrease) of the weights of the vertices and the edges, respectively. These energies associate a scalar score to any real-valued function f, which can then be used to find the optimal distortion function by computing $f_{\text{opt}} = \arg\max_f E(G_i, G_{i+1}, f)$. In each case, $E(G_i, G_{i+1}, f)$ is large when f is supported on the regions where these weights change the most. Note also that both

$E_{\text{vertex diff}}(f)$, and $E_{\text{edge diff}}(f)$ are defined such that the change is scaled by the vertex weight $d_i(u)$ in the denominator. This is because a single edge addition or deletion is typically less important for a vertex with a large weight (or degree) than for a vertex with a small weight. However, this choice might also be application-specific. If this normalization is not necessary, it can easily be removed from all of the derivations below.

2.2 Choice of Scale via Reduced Functional Space

After selecting a distortion energy $E(G_i, G_{i+1}, f)$ our goal is to find the optimal distortion function f_{opt} by maximizing E for given G_i, G_{i+1}. Moreover, as mentioned in Section 2, we would like to be able to control the *scale* of the solution, in order to gain robustness to disparate, possibly noisy local changes, and to detect the areas or regions where the most important changes occur.

Thus, instead of maximizing $E(G_i, G_{i+1}, f)$ across all choices of f, we propose to consider the function f that lies in the appropriate functional subspace. More concretely, we enforce the function f to lie in the *linear subspace* spanned by some "desirable" functions φ, i.e., we force: $f = \sum_{j=1}^{k} a_j \varphi_j$ where φ_j are some k fixed pre-defined functions, i.e., $\varphi_j : V_i \longrightarrow \mathbb{R}$ and a_j are the unknown scalar coefficients. This way, computing the optimal f amounts to finding the coefficients $\{a_k\}$ such that $E(G_i, G_{i+1}, f) = E(G_i, G_{i+1}, \sum_j a_j \varphi_j)$ is maximized.

In practice, we control the *scale* of the solution via the choice of k, which corresponds to the dimensionality of the functional space. A small value of k corresponds to *global scale* as it enforces f to be chosen in the space spanned by a small set (of potentially globally supported) functions, whereas larger values of k provide more freedom for selecting the optimal distortion function f. In the limit, when k equals the number of vertices in the graph, and φ are linearly independent, then f can be chosen to be an arbitrary function, including an indicator function of a single vertex. In this paper, we consider the following two families of functions:

Option 1: Region-based functions. Perhaps the most intuitive choice of a functional family corresponds to simply taking φ_j to represent indicator (characteristic) functions of some regions on the graph G_i. In the simplest case, each such function can represent a neighborhood of some fixed vertex, of a given size. More precisely, given a partition of the vertex set V_i into k distinct regions $\{R_1, R_2, \ldots, R_k\}$ we define $\varphi_j : V_i \longrightarrow [0,1]$ as the indicator function of R_j: $\varphi_j(u) = 1$ if $u \in R_j$, and $\varphi_j(u) = 0$ otherwise. We can also incorporate a distance-to-modification behavior by simply defining $\varphi_j(u) = 1$ if $u \in R_j$ and u has a modified neighborhood, and $\varphi_j(u) = h(dist(u))$ where $dist(u)$ is the distance from u to the closest modification and $h : \{0, 1, \ldots n-1\} \rightarrow [0,1]$ is a decreasing function provided by the user (we define $h(dist(u)) = 1/(1 + dist(u))$ in the experiments reported in Section 3). In practice we use the partitions computed by the *Louvain algorithm* [5], which is a hierarchical method for community detection based on modularity optimization. Such a clustering method is especially appropriate in our setting since it provides a *resolution* parameter that controls the desired level in the clustering hierarchy, which naturally corresponds to the scale at which we analyze the graph.

Option 2: Laplacian eigen-basis. We consider as basis the eigenfunctions associated with the *k smallest eigenvalues* of the generalized eigenvalue problem:

$$L_{G_i}\varphi = \lambda D_{G_i}\varphi \tag{2}$$

where L_{G_i} is the Laplacian matrix of G_i and D_{G_i} is the diagonal matrix of vertex weights. We choose this basis because the eigenfunctions of the Laplacian naturally have a multi-scale property, which intuitively corresponds to the equivalent of Fourier bases and which has been used extensively in the context of signal processing on graphs [26]. In our context, remark that each φ_j is associated with a non-negative eigenvalue $\lambda_j \geq 0$ (since L_{G_i} and D_{G_i} are symmetric positive semi-definite). Moreover, a simple calculation shows that for any $f = \sum_{j=1}^{k} a_j \varphi_j$

$$\frac{f^T L_{G_i} f}{f^T D_{G_i} f} = \frac{\sum_{u,v}(f(u)-f(v))^2 w_i(u,v)}{\sum_u f(u)^2 d_i(u)} \leq \lambda_k \tag{3}$$

Now, the quantity $\sum_{u,v}(f(u)-f(v))^2 w_i(u,v)$ can naturally be interpreted as the smoothness of the function, since it captures the sum of the squared differences of f along the edges of the graph. This means that if a function f lies in the span of the eigenfunctions corresponding to the k smallest eigenvalues of the problem in Eq. (2), then the smoothness of f, as defined in Eq. (3), is bounded by λ_k.

Note that for a connected graph, when $k = 1$, then f must be a constant function, since $\lambda_1 = 0$, and φ_1 must be constant. Conversely, if $k = n$ then f can be an arbitrary function, since φ are linearly independent. Thus, we interpret k as controlling the scale of the solution, where small k corresponds to global scale (very smooth functions), and large k corresponds to local scale (possibly arbitrarily irregular, or concentrated functions).

2.3 An Algorithm to Compute the Spectral Distortion

We can now design a simple procedure for computing the optimal distortion function, where all steps can be expressed in terms of linear algebraic computations. Assuming we want to maximize the energy $E_r(G_i, G_{i+1}, f)$, we compute the optimal distortion function f_{opt} with the following three steps:

1. Choose the value for the parameter k and compute the family $\mathscr{F}$ at that scale. For example, when using the Laplacian basis, compute the k smallest eigenvalues $\lambda_1, \ldots, \lambda_k$ and the corresponding eigenfunctions $\varphi_1, \ldots, \varphi_k$ of the problem $L_{G_i}\varphi = \lambda D_{G_i}\varphi$. Store these functions as columns of the matrix Φ^k.
2. Given the matrix Φ^k and an energy E_r, compute the optimum function $f_{opt} := \arg\max_{f \in \text{span}(\Phi^k)} E_r(G_i, G_{i+1}, f)$. For this, we solve the eigenproblem: $\max_\sigma S_{i+1}\xi =$

$\sigma\xi$ where the matrix S_{i+1} (of size $k \times k$) is given as:

$$\begin{aligned} E_{\text{vertex diff}} &: S_{i+1} := (\Phi^k)^T (D_{G_i} - D_{G_{i+1}})^2 \Phi^k \\ E_{\text{edge diff}} &: S_{i+1} := (\Phi^k)^T L^{-}_{G_i,G_{i+1}} \Phi^k \end{aligned} \tag{4}$$

Here the diagonal matrices D and the Laplacian matrices L follow the definitions given in the beginning of Section 2, and the matrix L^{-} is defined such that $L^{-}(u,v) = -(w_i(u,v) - w_{i+1}(u,v))^2$, and $L^{-}(u,u) = \sum_v (w_i(u,v) - w_{i+1}(u,v))^2$.

3. Finally, the optimal distortion function corresponds to the eigenvector $\xi_{\max}$ (having size k) associated with the largest eigenvalue σ_{max} of the eigenproblem from step 2. The function f on the vertices of G_i can be computed via the matrix product: $\mathbf{f}_{\text{opt}} = \Phi^k \xi_{\max}$.

The correctness of the algorithm above for computing the optimal distortion function is ensured by the following Lemma (see the Appendix for the proof):

Lemma 1. *The algorithm described in the three steps above is guaranteed to result in a distortion function that maximizes the given energy, while remaining within the subspace spanned by the eigenfunctions Φ^k.*

Note that the pipeline above is designed to compute a single optimal distortion function $\mathbf{f}_{\text{opt}}$ at the given scale k. In practice, this typically corresponds to detecting a single most distorted area or region of the graph. In order to compute the top p most distorted regions, it can be convenient to consider more than one eigenvector of S_{i+1} in step 3 above. In this case we take the linear combination of the squares of eigenvectors corresponding to the p largest eigenvalues of S_{i+1} described in step 2. In other words we compute: $\mathbf{f}_{\text{opt}} = \sum_{j \le p} \sigma_j (\mathbf{f}_j[u])^2$, where we define $\mathbf{f}_j = \Phi^k \xi_j$ and σ_j is the j^{th} *largest* eigenvalue of S_{i+1}. Finally we normalize $\mathbf{f}_{\text{opt}}$ to have values between 0 and 1.

Key parameters: The key parameters in our framework include: the choice of the basis, the distortion energy function, the choice of scale or smoothness parameter k, and the choice of p corresponding to the number of largest eigenvalues, related to the number of highest-distorted parts that are considered.

3 Experimental evaluation

We provide an experimental evaluation of our spectral distortion (denoted δ_{SP}) and compare it to other approaches proposed in the visualization of dynamic graphs.

Choice of the energy. Remark that the choice of energy is application specific. For example, if the network evolution is primarily controlled by change in *vertex degrees* (or weights), then $E_{\text{vertex diff}}$ is more appropriate. This happens, in particular, if the evolution contains *only* edge additions or removals but not both. We also report in [8] experimental results for the energy $E_{\text{edge diff}}$, that is more suitable for graphs evolving under both edge removals and additions.

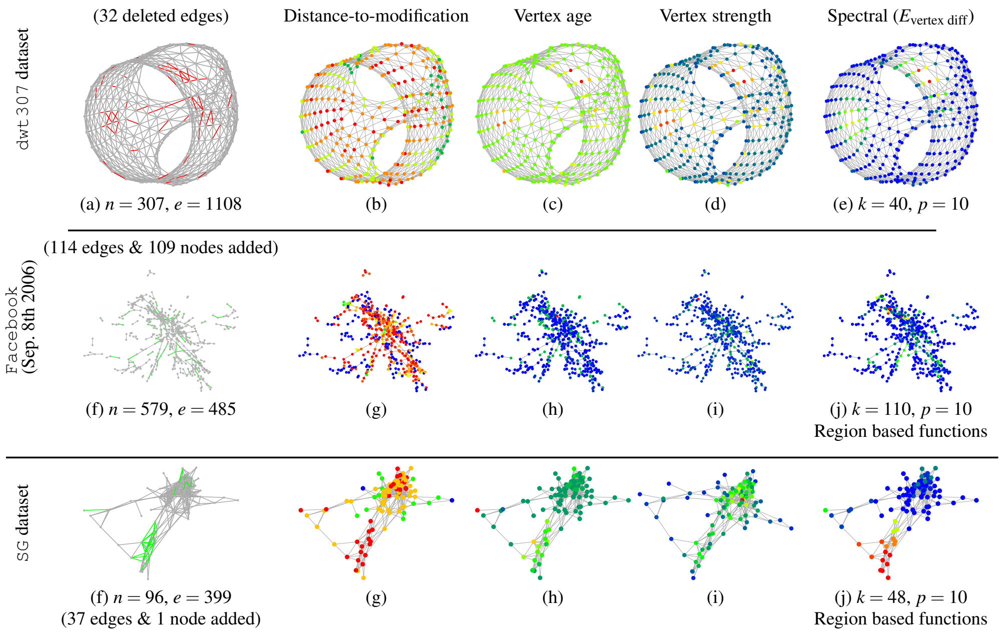


Fig. 2: (Left) Network evolution: the red (resp. green) segments represent the removed (resp. added) edges in a single time step. (Columns 2-5) Qualitative comparison between the distortion measures δ_{DM} (distance-to-modification), δ_{VA} (vertex age) and δ_{VS} (vertex strength) with our spectral distortion.

Since this is the case for the datasets we consider below, we only present results using $E_{\text{vertex diff}}$. We also report experimental results for the energy $E_{\text{edge diff}}$, that is more suitable for graphs evolving under both edge removals and additions.

3.1 Datasets

We perform tests on dynamic networks having different structural properties and evolution behavior: as in most existing works [10, 11, 15, 17], all of these graphs are undirected and unweighted.

Real-world networks. We consider real-world networks from the SuiteSparse Matrix Collection [13] (`3elt`, `dwt307`, etc.). The sizes of these graphs range from hundreds to thousands of nodes. As in [15] we construct a dynamic sequence of networks with a simple process based on a random edge decimation.

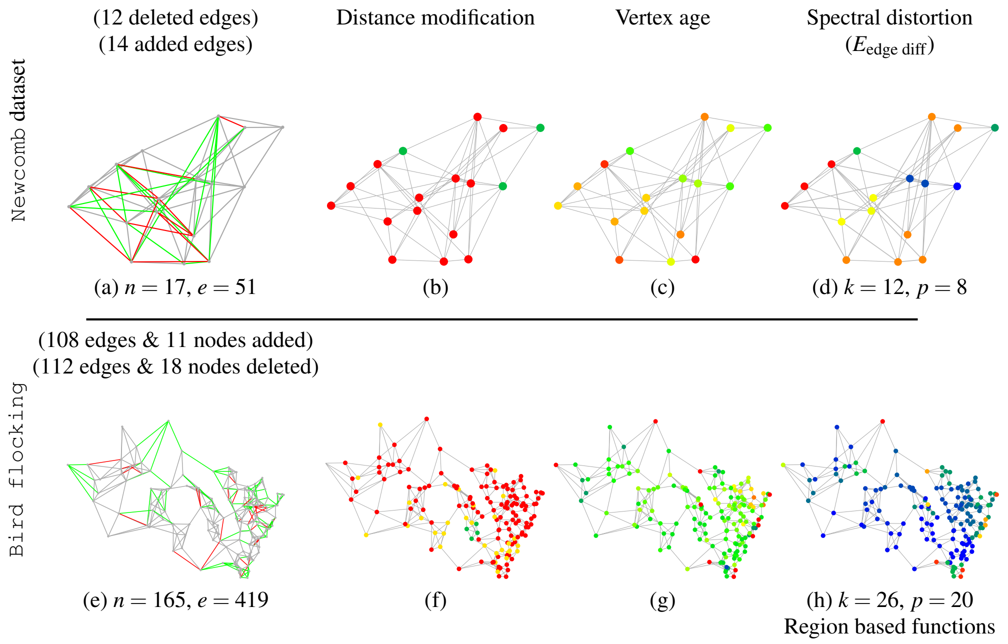


Fig. 3: Qualitative comparison of distortion functions for the `Newcomb` and `Bird flocking` datasets. We compare different distortion measures δ_{DM} (distance-to-modification) and δ_{VA} (vertex age) with our spectral distortion (obtained optimizing the energy $E_{\text{edge diff}}$).

Complex networks. We consider a sequence of networks (also evaluated in [11]) extracted from the `Facebook-Growth` dataset [1]: this dataset spans an interval from Sep. 2006 to Jan. 2009. we extract a sequence of dynamic networks following the evolution of the largest connected component over the first 15 days (each time step correspond to the growth during a single day).
Interaction networks. We consider the sequence of aggregated networks analyzed in [9] (referred to as `SG`), where nodes are individuals visiting the Science Gallery in Dublin and edges describe the face-to-face proximity between individuals over a daily time window: these networks have a few hundreds of nodes and evolve under an aggregation process (only vertex and edge additions occur).
Newcomb network. We consider the *Newcomb's fraternity* dataset (referred to as `newcomb` in previous works [10, 15, 17]), whose evolution is driven by a dramatic and global rewiring process.
Proximity networks. We also evaluate a second dataset whose growth is based on a dynamic process where new entities and links are added or removed over time. More precisely we consider a collection of networks (referred to as `bird flocking`),

[1] The raw data is available at `https://www.eecs.wsu.edu/~yyao/StreamingGraphs.html`

describing the collective motion of birds. Given the 3D locations of birds (from the dataset [14]) we construct a sequence of networks by computing the k-nearest neighbor graph at each time step (this choice is motivated by the behavior of interactions, which are independent of the distance between birds). This network sequence is particularly interesting since the evolution is driven by a dramatic edge rewiring process, including vertex additions and removals (which occur because of occlusion issues during the tracking phase).

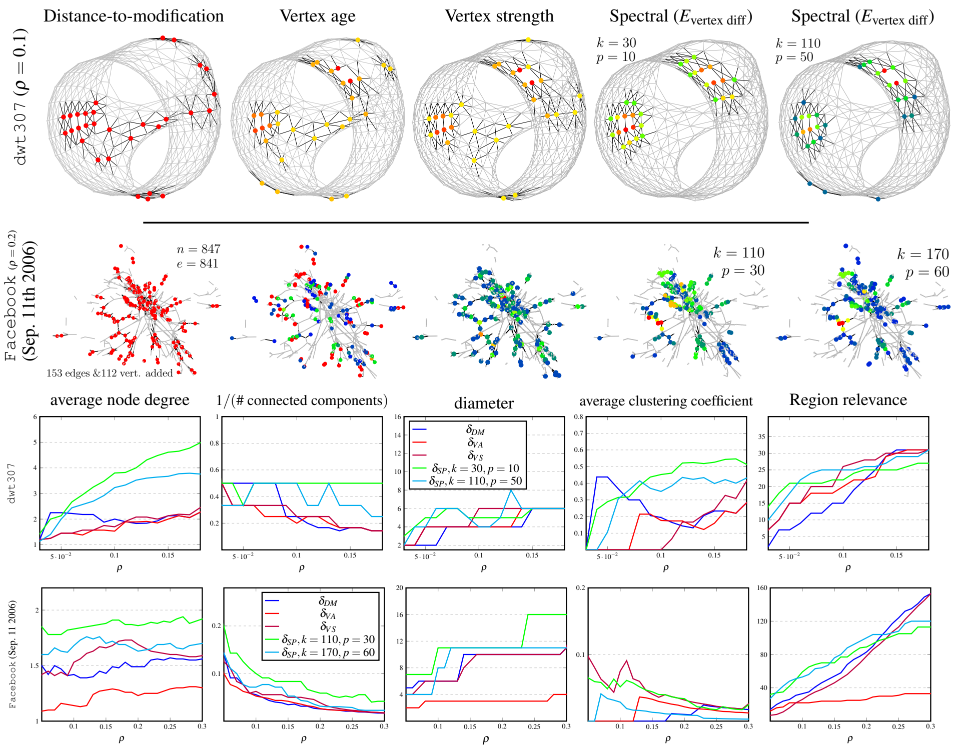


Fig. 4: Threshold filtering approach: in the pictures above we highlight a fraction ρ of vertices with highest distortion. In the charts below we plot the structural properties (*average node degree, average clustering coefficient, …*) and the *region relevance* of each filtered graph as a function of ρ.

3.2 Baselines

We compare our approach to three other notions of distortion: *distance-to-modification*, *vertex age* and *vertex strength* distortions based on the methods proposed in [15],

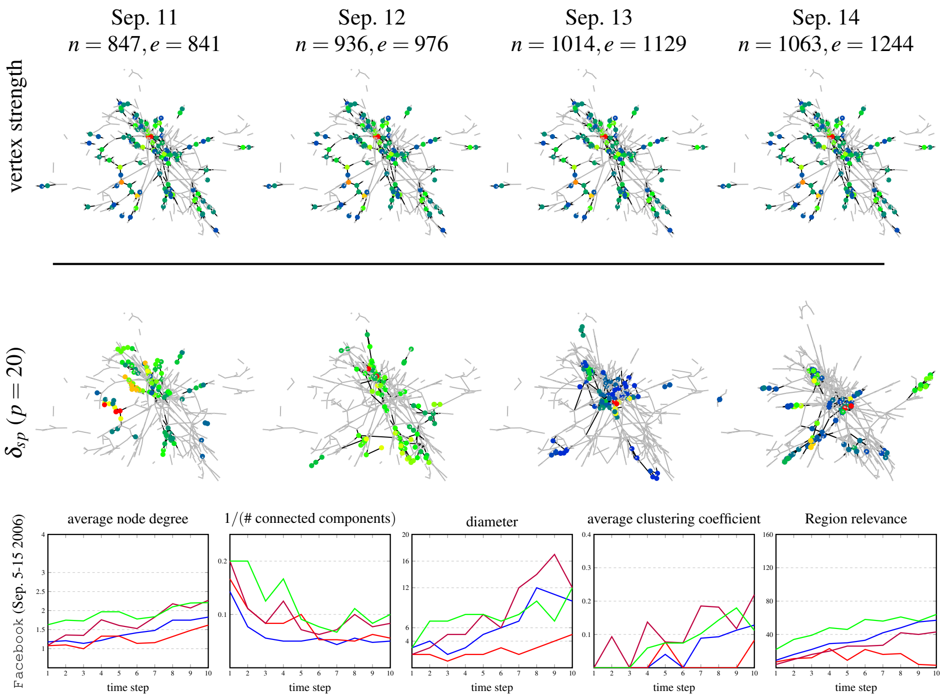


Fig. 5: Dynamic growth of the `Facebook` network: we compare distortion functions evaluating the evolution of structural properties over the first ten days from Sep. 5 to Sep. 15 2006. The pictures above show the graph filtered using spectral distortion over four time steps. All pictures and plots are obtained setting $\rho = 0.1$.

[17] and [18], respectively.
The *distance-to-modification* distortion is the function $\delta_{DM}(u) : V \rightarrow [0,1]$ defined by $\delta_{DM}(u) = h(dist(u))$ where h is a decreasing function, and $dist(u)$ is the graph distance between u and the closest modification in the graph. More precisely, following [15] we define $\delta_{DM}(u) = 1 - \alpha^{(1-\frac{dist(u)}{R_c})}$ where α (scale parameter) and R_c (cutoff distance) are user-supplied parameters.
To better distinguish between major and minor changes Gorochowski et al. [17] introduced an age function $age(u,i)$ that quantifies the amount of changes for vertex u at time i . We then take a *vertex age distortion* $\delta_{VA}(u) = e^{-\beta age(u,i)}$, where $\beta \in \mathbb{R}^+$ is a user supplied parameter.
Finally, we define the distortion δ_{VS} based on the notion of *vertex strength* following the approach proposed in [18], that integrates vertex degrees and makes use of an exponential sliding time-window (we normalize to get values in $[0,1]$).

For completeness we provide more details in the appendix.

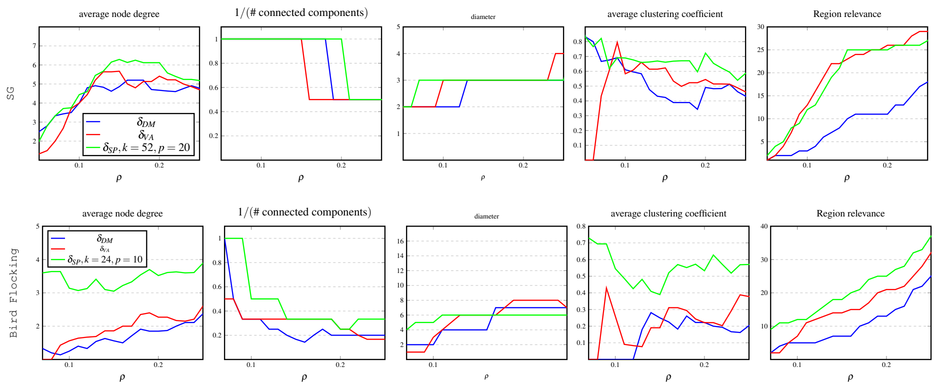


Fig. 6: Statistics concerning the threshold filtering approach for the `SG` and `Bird flocking` networks. We plot the structural properties and the *region relevance* as a function of ρ (fraction of vertices with highest distortion).

3.3 Evaluation: Region extraction

We evaluate different distortion measures by their ability to highlight relevant *regions* of the graph that undergo the biggest changes. Thus, we use each measure to first extract the regions on the graph as follows:

Approach 1: threshold filtering. In the simplest case, we simply keep a small fraction $\rho \in [0,1]$ of the vertices having the largest distortion: as suggested in [18], we compute the subgraph induced by non isolated vertices, containing at most $\rho|V_{i+1}|$ vertices (see Figure 4 for an illustration).

Approach 2: BFS traversal. One drawback of the previous approach is that it can result in many disconnected regions, especially when the network evolution is dramatic. To counter this, we also use each distortion function to perform a BFS traversal (vertices with large distortion are visited first) starting from some initial seed vertex, which guarantees a single connected region (whose size is denoted N_r). This process is repeated taking as seeds a fraction ρ of vertices with largest distortion.

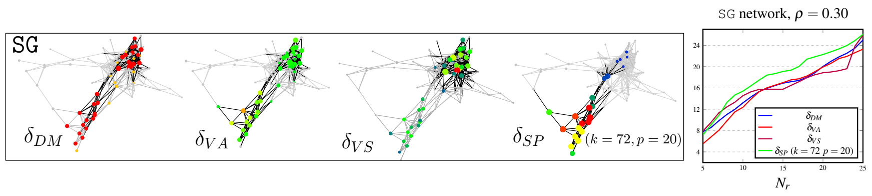


Fig. 7: Region extraction approach based on BFS traversal. (left) for each distortion function (and for a fixed value of N_r), we highlight the *average region* extracted for the SG network. (right) plots of the average region *relevance* (higher is better).

Region quality. We evaluate the extracted regions according to several measures of quality: first, as suggested in [18] we measure the *structural properties* [2] of the extracted regions: for instance, higher values of the average node degree (or 1/#connected components) are better, as they correspond to filtered regions which are locally more dense and less fragmented. Second, we evaluate the total number of edge modifications (additions or deletions) involving vertices that lie in a region, referred to as *region relevance* (higher values are better).

3.4 Experimental comparison and discussion

Qualitative comparison

In our first evaluation we plot the distortion functions obtained by different methods across several datasets in Fig. 2 (for the energy $E_{\text{vertex diff}}$) and Fig. 3 (for the energy $E_{\text{edge diff}}$). In all our examples we use the heat map color-scale representing distortion values in the range $[0 \dots \max_{u \in V} \delta(u)]$, from blue to green to red. As can be seen in Fig. 2, previous methods can be either *too* sensitive, highlighting a large portion of the graph (e.g., in the case of the δ_{DM} distortion) or fail to distinguish *regions* or parts of the graph that undergo the changes (in the case of the δ_{VA} distortion for the SG, for example).

Baseline approaches clearly fail to distinguish between major and negligible modifications, and are not very informative when dealing with dramatic changes as the for the Newcomb and bird flocking networks (see Fig. 3).

On the other hand, our method is precise and at the same time captures the regions that undergo the most changes the multi-scale manner.

Threshold filtering

As done in[18], we compare distortion functions by evaluating the evolution of the structural properties, performing the extraction of relevant regions with the threshold filtering approach. The layouts and plots of Fig. 4 and Fig. 6 show that the

[2] We make use of the GraphStream library [24] for the computation of structural network parameters.

regions detected with our spectral approach do not lead to dramatic fluctuations of the main structural properties: the filtered vertices are likely to define connected subgraphs, whose structural properties vary in a smooth way, even for small values of ρ. This does not hold for basic distortion measures (especially δ_{DM} and δ_{VA}), that lead to highly disconnected and sparse sub-graphs, and more drastic fluctuations of structural properties, not being able to distinguish the most relevant changes from local noise. Observe that our spectral approach offers to the user the capability of choosing the parameters k and p in order to reach the right scale (tuning k) and only keep the highest distorted regions (tuning p). This feature helps the user to select, depending on the application and desirable goals, the regions with the most relevant evolution, while discarding local irrelevant noisy changes (see layouts and plots of region relevance in Fig. 4 and Fig. 7): even for small value of ρ, the spectral distortion allows us to correctly highlight the most relevant changed regions, while δ_{DM}, δ_{VA} and δ_{VS} lead to much more dispersion, not being able to discarding noisy modifications. Observe that as ρ increases, the capability of δ_{DM}, δ_{VA} and δ_{VS} distortions to capture modifications also increases (see the plot of region relevance in Fig. 4). Above a given threshold the total amount of changes captured by spectral distortion does not increase significantly since irrelevant local changes are ignored: this is the unavoidable price to pay for keeping the filtered graph fragmented as little as possible.

The plots of Fig. 5 show the evolution of structural properties over the first ten days of the `Facebook` network sequence (all results and layouts are obtained setting $\rho = 0.1$): the spectral distortion always captures a larger amount of modifications, while leading in overall to a smoother transition of the filtered graphs, which are more dense and locally connected, between consecutive time steps.

BFS-driven region extraction

For a fixed value of ρ, we plot in Fig. 7 the average relevance Rel_{av}, varying the size N_r. As confirmed by our experiments, the spectral distortion always allows us to detect regions with higher values of the region relevance. This reflects the ability of spectral distortion to distinguish between regions, depending on the amount of local changes: a region strongly highlighted by spectral distortion will be visited first, leading to a significant contribution to the average energy. While small regions with a negligible amount of changes will be discarded during the BFS traversal, since they are unlikely to be detected by the spectral distortion.

3.5 Time Complexity and Runtime Performance

Below we provide a discussion on the theoretical and practical complexity of the algorithms described in this work. We have developed a pure Java implementation of all algorithms presented in this work; for the calculation of the Laplacian eigen-

			Distortion	Spectral distortion			Force-directed layout	
			vertex age	Step 1 ($k = 40$)		Steps 2 and 3	one iteration	
Network	vertices	edges		eigen-basis	region functions	$k = 40$	FR91 [16]	with octrees
`SG`	96	399	0.0003	0.037	0.002	0.0003	0.002	0.001
`dwt307`	307	1.1K	0.0015	0.048	0.004	0.0007	0.009	0.003
`3elt`	4720	13.7K	0.009	0.22	0.075	0.003	0.73	0.11
`barth5`	15606	61.4K	0.026	0.84	0.32	0.011	15.2	0.57

Table 1: This table reports the average runtime performance of all steps involved in the computation of graph distortions and force-directed layouts (for a single evolutionary step). The complexity of the vertex age distortion (implemented in Java) is compared to the computation of spectral distortion setting $k = 40$. The Laplacian eigen basis is computed using Matlab 14 (option 2), while the region-based functions (option 1) are computed with a Java implementation of the Louvain algorithm. The last two columns reports the time required to run a single iteration of a force-directed layout: we implement the standard `FR91` layout, as well as the fast version with approximate calculation of repulsive forces using octrees. All performances are expressed in seconds.

basis we use the eigen solver provided by Matlab (version 14). All our tests are run on a HP EliteBook, equipped with an Intel Core i7 (2.66GHz) and 8GB of RAM, using Java 1.8.

First observe that the computation of the spectral distortion in a single evolutionary step (recall the procedure in Section 2.3) is expressed in terms of linear algebra computations, which makes our algorithm very easy to implement and quite efficient in practice. To simplify the notation and discussion about practical performance, we assume that the number n of nodes is roughly the same in G_i and G_{i+1}.

Computation of basis functions.

The most expensive step in our distortion estimation algorithm is the computation of the basis functions.

When using option 2 (Laplacian eigen-basis) the extraction of the functions φ_j involves computing the first k eigenvalues of a generalized eigenproblem of size $n \times n$ for a graph with n nodes. In practice we use an iterative Arnoldi-Lanczos algorithm implemented in Matlab's `eigs` function [3]. The running time strongly depends on the sparsity structure of the input matrices and in practice scales approximately linearly with respect to the number of edges and quadratically with respect to k for small values of k.

When using option 1 (region-based functions) the runtime performances are much better: the timing cost is dominated by the computation of a clustering parti-

[3] We also perform tests using several Java libraries for linear algebra (`Jama`, `Colt`, `Parallel Colt`, `MTJ`, ...): as far as we can see there are no Java libraries providing efficient eigen solvers (for the generalized eigen problem) for processing large sparse linear systems.

tion of the desirable size. For this we make use of the Louvain algorithm based on modularity optimization, whose complexity is assumed to be $O(n \log n)$ in practice, which allows processing large networks efficiently.

The runtime performances reported in Table 1 (columns 5 and 6) correspond to the computation of basis functions for $k = 40$.

Energy optimization.

The steps (2) and (3) of our algorithm are much less expensive, as the matrices involved have sizes $k \times k$ and $k \times n$ respectively (k and p being usually much smaller than n), which leads to approximate complexity of $O(k^3 + k^2 n + pn)$: in practice the runtime performance of these steps are negligible and comparable to the ones of the computation of the vertex age distortion (see Table 1).

Finally, observe that according to the results reported in Table 1 (see last two columns), the overall computational cost of our spectral distortion is, for sufficiently large values of n, smaller then the timing cost required to run a single iteration of a standard force-directed method [16], which requires $O(|E_i| + |V_i|^2)$ time in the basic setting. When using a fast approximation of repulsive forces (using octrees), the timing cost of spectral distortion is comparable to the one required to run a single iteration.

4 Conclusion, Limitations and Future Work

We propose a novel, multi-scale framework for robustly detecting and visualizing changes in networks across two different time stamps (G_i and G_{i+1}). However, it is possible to integrate a time term in the computation of spectral distortion, that would take into account a full sequence $G_1, G_2, \ldots, G_{i+1}$ (as done in [17, 18]). Our framework described in Section 2.1, although illustrated on unweighted graphs, is rather general and allows to consider graphs with both edge and vertex weights.

References

1. Ahmed, A., Xing, E.P.: Recovering time-varying networks of dependencies in social and biological studies. PNAS **106**(29), 11,878–11,883 (2009)
2. Arbeitman, M.N., Furlong, E.E., Imam, F., Johnson, E., Null, B.H., Baker, B.S., Krasnow, M.A., Scott, M.P., Davis, R.W., White, K.P.: Gene expression during the life cycle of drosophila melanogaster. Science **297**(5590), 2270–2275 (2002)
3. Archambault, D.W., Purchase, H.C.: The ”map” in the mental map: Experimental results in dynamic graph drawing. Int. J. Hum.-Comput. Stud. **71**(11), 1044–1055 (2013)
4. Beck, F., Burch, M., Diehl, S., Weiskopf, D.: The State of the Art in Visualizing Dynamic Graphs. In: R. Borgo, R. Maciejewski, I. Viola (eds.) EuroVis - STARs (2014)
5. Blondel, V., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal Of Statistical Mechanics: Theory And Experiment (2008)

6. Brandes, U., Fleischer, D., Puppe, T.: Dynamic spectral layout with an application to small worlds. J. Graph Algorithms Appl. **11**(2), 325–343 (2007)
7. Brandes, U., Wagner, D.: A bayesian paradigm for dynamic graph layout. In: Graph Drawing, 5th Intern. Symp., GD ’97, pp. 236–247 (1997)
8. Castelli Aleardi, L., Salihoglu, S., Singh, G., Ovsjanikov, M.: Spectral Measures of Distortion for Change Detection in Dynamic Graphs (extended version) (2018). URL `https://hal.archives-ouvertes.fr/hal-01864079`. (long version of the extended abstract submitted to Complex Networks 2018)
9. Cattuto, C., Van den Broeck, W., Barrat, A., Colizza, V., Pinton, J., Vespignani, A.: Dynamics of person-to-person interactions from distributed rfid sensor networks. PLOS ONE **5**(7), e11,596 (2010)
10. Che, L., Liang, J., Yuan, X., Shen, J., Xu, J., Li, Y.: Laplacian-based dynamic graph visualization. In: IEEE PacificVis, pp. 69–73 (2015)
11. Crnovrsanin, T., Chu, J., Ma, K.: An incremental layout method for visualizing online dynamic graphs. In: Graph Drawing and Network Visualization, pp. 16–29 (2015)
12. Davidson, R., Harel, D.: Drawing graphs nicely using simulated annealing. ACM Trans. Graph. **15**(4), 301–331 (1996)
13. Davis, T.A., Hu, Y.: The university of florida sparse matrix collection. ACM Trans. Math. Softw. **38**(1), 1 (2011). Http://www.cise.ufl.edu/research/sparse/matrices
14. Evangelista, D., Ray, D., Raja, S., Hedrick, T.: Three-dimensional trajectories and network analyses of group behaviour within chimney swift flocks during approaches to the roost. Proceedings of the Royal Society B **284**(1849) (2017)
15. Frishman, Y., Tal, A.: Online dynamic graph drawing. IEEE Trans. Vis. Comput. Graph. **14**(4), 727–740 (2008)
16. Fruchterman, T.M.J., Reingold, E.M.: Graph drawing by force-directed placement. Softw., Pract. Exper. **21**(11), 1129–1164 (1991)
17. Gorochowski, T.E., di Bernardo, M., Grierson, C.S.: Using aging to visually uncover evolutionary processes on networks. IEEE Trans. Vis. Comput. Graph. **18**(8), 1343–1352 (2012)
18. Grabowicz, P.A., Aiello, L.M., Menczer, F.: Fast filtering and animation of large dynamic networks. EPJ Data Science **3**(1), 27 (2014)
19. Hadlak, S., Schulz, H., Schumann, H.: In situ exploration of large dynamic networks. IEEE Trans. Vis. Comput. Graph. **17**(12), 2334–2343 (2011)
20. Hall, K.M.: An r-Dimensional Quadratic Placement Algorithm. Management Science **17**(3) (1970)
21. Kamada, T., Kawai, S.: An algorithm for drawing general undirected graphs. Inf. Process. Lett. **31**(1), 7–15 (1989)
22. Koren, Y.: Drawing graphs by eigenvectors: theory and practice. Computers and Mathematics with Applications **49**(11), 1867 – 1888 (2005)
23. Misue, K., Eades, P., Lai, W., Sugiyama, K.: Layout adjustment and the mental map. J. Vis. Lang. Comput. **6**(2), 183–210 (1995)
24. Pigné, Y., Dutot, A., Guinand, F., Olivier, D.: Graphstream: A tool for bridging the gap between complex systems and dynamic graphs. Emergent Properties in Natural and Artificial Complex Systems (2007)
25. Pisanski, T., Shawe-Taylor, J.: Characterizing graph drawing with eigenvectors. J. Chem. Inf. Comput. Sci. **40**(3), 567–571 (2000)
26. Shuman, D.I., Narang, S.K., Frossard, P., Ortega, A., Vandergheynst, P.: The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. IEEE Signal Processing Magazine **30**(3), 83–98 (2013)
27. Tang, J., Scellato, S., Musolesi, M., Mascolo, C., Latora, V.: Small-world behavior in time-varying graphs. Physical Review E **81**(5), 055,101 (2010)
28. Walshaw, C.: A multilevel algorithm for force-directed graph-drawing. J. Graph Algorithms Appl. **7**(3), 253–285 (2003)
29. X. Du, Y.W., Wu, L.: A multi-constraint layout algorithm for dynamic network visualization. In: IEEE 2nd International Conference on Big Data Analysis (ICBDA), pp. 832–836 (2017)

30. Xu, K.S., Kliger, M., III, A.O.H.: A regularized graph layout framework for dynamic network visualization. Data Min. Knowl. Discov. **27**(1), 84–116 (2013)
31. Yuan, X., Che, L., Hu, Y., Zhang, X.: Intelligent graph layout using many users' input. IEEE Trans. Vis. Comput. Graph. **18**(12), 2699–2708 (2012)

5 Appendix

5.1 Proof of Lemma 1

The validity of Lemma 1 follows from the following statement:

Lemma 2. *The maximizer of $E_{vertex\ diff}$ and $E_{edge\ diff}$ among all functions in the span of Φ^k is given by $\Phi^k\xi_{\max}$ where $\xi_{\max}$ is the eigenvector corresponding to the largest eigenvalue of S_{i+1}, given in Eq. (4) in Section 2.3.*

Proof. First note that any function f in the span of Φ^k can be written as $f = \Phi^k\mathbf{f}$, where $\mathbf{f}$ is some vector of coefficients. Moreover, for any function f, we can write the energies in matrix-vector form as follows:

$$\begin{aligned}
E_{\text{vertex diff}}(f) &= \frac{\sum_u f(u)^2(d_i(u)-d_{i+1}(u))^2}{\sum_u f(u)^2 d_i(u)} \\
&= \frac{f^T(D_{G_i}-D_{G_{i+1}})^2 f}{f^T D_{G_i} f}, \text{ while} \\
E_{\text{edge diff}}(f) &= \frac{\sum_{u,v}(f(u)-f(v))^2(w_i(u,v)-w_{i+1}(u,v))^2}{\sum_u f(u)^2 d_i(u)} \\
&= \frac{f^T L^-_{G_i,G_{i+1}} f}{f^T D_{G_i} f}.
\end{aligned}$$

The last equality holds since for any matrix W, s.t., $W(u,u) = -\sum_{v\neq u} W(u,v)$, we have: $f^T W f = -\sum_{u,v} W(u,v)(f(u)-f(v))^2$. Now note that $L^-_{G_i,G_{i+1}}$ satisfies this property and $L^-(u,v) = -(w_i(u,v)-w_{i+1}(u,v))^2$. This implies that if $f = \Phi^k\mathbf{f}$, then for both energies we have:

$$E(f) = \frac{\mathbf{f}^T S_{i+1}\mathbf{f}}{\mathbf{f}^T \Phi^k D_{G_i}\Phi^k\mathbf{f}},$$

where S_{i+1} is defined appropriately (as in Eq. (4)). Furthermore since Φ^k are given as the eigenvectors corresponding to the generalized eigenproblem $L_{G_i}\phi = \lambda D_{G_i}$, they are orthonormal with respect to the matrix D_{G_i}, so that $\Phi^k D_{G_i}\Phi^k = Id$. Therefore we have: $E(f) = \frac{\mathbf{f}^T S_{i+1}\mathbf{f}}{\mathbf{f}^T\mathbf{f}}$. By the standard min-max theorem, the vector that maximizes this ratio (the Rayleigh-Ritz quotient) must be the eigenvector corresponding to the largest eigenvalue of S_i, which implies the result of the lemma.

5.2 Definition of vertex age and vertex strength

Vertex age.

For the sake of completeness, we provide here the definition of *vertex ages* as described in [17]. In order to quantify the amount of changes involving a vertex u at time i one can make use of a function $age(u,i) : V \times \mathbb{N} \to \mathbb{R}^+$ whose definition integrates both the degree of the node, as well as the total amount of unchanged incident edges.

At the beginning all vertices in G_1 have the same initial age, so $age(u,1) = 1$ for all $u \in G_1$. A new node added at time $i+1$ gets age 1; otherwise for a given vertex $u \in G_{i+1}$ at time $i+1$ we have:

$$age(u,i+1) = \begin{cases} age(u,i)+1, & \text{if } u \text{ has degree 0 in } G_{i+1} \\ age(u,i)\frac{\mathscr{A}^{rem}_{i+1}(u)}{\mathscr{A}^{tot}_{i+1}(u)}+1, & \text{otherwise} \end{cases}$$

where $\mathscr{A}^{rem}_{i+1}(u)$ (resp. $\mathscr{A}^{add}_{i+1}(u)$ and $\mathscr{A}^{del}_{i+1}(u)$) is the sum of vertex ages of all unchanged (resp. new and removed) edges of vertex u between G_i and G_{i+1}.

Finally we set $\mathscr{A}^{tot}_{i+1}(u) = \mathscr{A}^{rem}_{i+1}(u) + \mathscr{A}^{add}_{i+1}(u) + \mathscr{A}^{del}_{i+1}(u)$.

Vertex strength.

We provide an illustration of the computation of the *vertex strength*, following the approach described in [18]. The main idea is to assign to each vertex v a score, called vertex strength, which quantifies the relevance of the dynamic changes involving v. Initially each vertex v_i has a score corresponding to its degree (we remind that in our setting we deal with unweighted graphs). The vertices with highest score are saved in a buffer of fixed size (the size, denoted N_b, is a user-supplied parameter).

Then vertices and edges are processed in chronological order. When a new vertex that does not belong to the buffer is processed then we insert it the buffer, replacing the vertex with lowest score (lying in the buffer). Otherwise, when we process vertices already in the buffer, the modification (removal or addition) of an edge (u,v) leads to increment the score of both u and v by 1. In order to discard inactive nodes (not involved in recent modifications), the score of all vertices is periodically decreased by a multiplicative forgetting factor $0 \leq c_f \leq 1$.

In our implementation we set $c_f = 0.25$, and we run the forgetting mechanism when moving from G_i to G_{i+1}: at that time we also perform an update of the buffer as done in [18] (nodes are sorted according to their score).