Knowl Inf Syst
DOI 10.1007/s10115-012-0575-9

REGULAR PAPER

Spectral evolution in dynamic networks

Jérôme Kunegis · Damien Fay · Christian Bauckhage

Received: 21 February 2011 / Revised: 30 March 2012 / Accepted: 6 October 2012

Abstract We introduce and study the spectral evolution model, which characterizes the growth of large networks in terms of the eigenvalue decomposition of their adjacency matrices: In large networks, changes over time result in a change of a graph's spectrum, leaving the eigenvectors unchanged. We validate this hypothesis for several large social, collaboration, rating, citation, and communication networks. Following these observations, we introduce two link prediction algorithms based on the learning of the changes to a network's spectrum. These new link prediction methods generalize several common graph kernels that can be expressed as spectral transformations. The first method is based on reducing the link prediction problem to a one-dimensional curve-fitting problem which can be solved efficiently. The second algorithm extrapolates a network's spectrum to predict links. Both algorithms are evaluated on fifteen network datasets for which edge creation times are known.

Keywords Graph kernels · Link prediction · Spectral graph theory · Network dynamics

1 Introduction

Many learning problems on networks can be described as link prediction: finding movies a user might like, recommending friends, predicting communication events, or finding related topics in a collaboration graph. In each of these cases, a given network is assumed to *grow* over time, and the task is to predict where new edges will appear and, in some cases, to also

J. Kunegis (✉)
Institute for Web Science and Technologies, Universität Koblenz–Landau, Universitätsstraße 1, 56070 Koblenz, Germany
e-mail: kunegis@gmail.com; kunegis@uni-koblenz.de

D. Fay
University College Cork, Cork, Ireland

C. Bauckhage
Fraunhofer IAIS, Sankt Augustin, Germany

predict their weights. While the view that a network grows over time is intuitive, only few link prediction algorithms use temporal data explicitly. In many recently published network datasets, however, edge creation times are known and could be exploited to study the dynamics of networks. If a pattern of network growth can be observed in networks, then this information can be used to derive a new link prediction algorithm, by just extrapolating the dynamical changes in the network into the future. Therefore, a representation of a network is desirable in which only a small part is dynamic, and the rest is static. Our contribution following this approach is based on the eigenvalue decomposition, splitting a network into eigenvalues that change over time, and eigenvectors that remain constant.

In this paper, we are concerned with those link prediction algorithms that have an algebraic description. We show that they imply a network growth model that we call the *spectral evolution model*. The spectral evolution model states that in terms of the eigenvalue decomposition of a network's adjacency matrix, growth can be described as a transformation of the eigenvalues, without significant change in the eigenvectors. Several common link prediction functions are of this form, such as rank reduction and the matrix exponential. These functions are graph kernels, and each of these algorithms represents a specific assumption about network growth, leading to a different spectral transformation function. The goal of this study is thus to give a method of selecting good graph kernels, and furthermore to generalize them to a generic spectral link prediction algorithm that only assumes the spectral evolution model.

By observing the spectral evolution of large networks, we arrive at two new ways of predicting links in evolving networks:

– First, we validate individual spectral link prediction functions by comparing them to actual spectral growth. This allows us to observe in a simple way whether a particular spectral link prediction function is suitable for a given network. For link prediction functions that match a given network's growth pattern, we are then able to learn their parameters.
– Second, instead of using a specific spectral link prediction function which may rely on invalid assumptions, we extrapolate the spectral growth of the network. This has two advantages: If a network grows in accordance with a given link prediction function, we can observe this. If a network does not, then we obtain a link prediction algorithm that still works better than any simple spectral link prediction function.

The experiments in this paper are performed on fifteen datasets, as given in Table 5 at the end of the paper. All datasets that we use are unipartite, unweighted network datasets. Some datasets allow parallel edges. In all datasets, the creation time of edges is known.

First, we review the mathematical definitions needed in Sect. 2. Then, we give a motivational example and then define the spectral evolution model in Sect. 3. The model is tested on real networks in Sect. 4 and on common graph growth models in Sect. 5. Section 6 reviews two control tests to ensure that the spectral evolution model is a nontrivial property of real-world networks. As an application, we introduce the link prediction problem in Sect. 7. The Sects. 8 and 9 then present the two new link prediction algorithms based on the spectral evolution model. The two algorithms are evaluated in Sect. 10. Section 11 reviews related work and extensions to our approach, and Sect. 12 concludes this paper.

This paper is an extended and updated version of Kunegis et al. [28] and Kunegis and Lommatzsch [29]. In addition to presenting the results from these papers in a systematic way, this paper introduces the concept of latent preferential attachment, two control tests to validate the nontriviality of the spectral evolution model, and a systematic evaluation of all methods, including the analysis of four baseline link prediction algorithms.

2 Background

In order to analyze graphs, algebraic graph theory is a common approach. In algebraic graph theory, a graph with n vertices is represented by an $n \times n$ matrix. Various matrix decompositions can then be used to study the graph [12]. A certain number of interesting graph properties have been described spectrally, such as connectivity [48], centrality [6], balance [31], clustering [41,43], and even more remote applications such as face recognition [57].

Let $G = (V, E)$ be an undirected graph with vertices V and edges E. The edge set E can be represented by a matrix whose characteristics follow those of the graph. An unweighted graph $G = (V, E)$ on n vertices can be represented by an $n \times n$ 0/1 matrix $\mathbf{A}$ defined by:

$$\mathbf{A}_{ij} = \begin{cases} 1 & \text{if } \{i, j\} \in E \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Since G is undirected, $\mathbf{A}$ is symmetric. The matrix $\mathbf{A}$ is called the adjacency matrix of G. Another matrix we will use frequently is the degree matrix $\mathbf{D}$. This matrix $\mathbf{D}$ has the same size as $\mathbf{A}$, is diagonal, and its diagonal entries are the degrees of each node in the network, that is, the number of neighbors of each node, counting multiple edges if they are present. In other words, we can define $\mathbf{D}$ using

$$\mathbf{D}_{ii} = \sum_{(i,j) \in E} \mathbf{A}_{ij}. \tag{2}$$

Spectral graph theory is a branch of algebraic graph theory that applies various matrix decompositions to characteristic graph matrices such as $\mathbf{A}$ in order to study graph properties. The word *spectral* refers to the spectrum of a network, which is given by the eigenvalue decomposition of a graph's adjacency matrix, as described below.

Real symmetric matrices can be decomposed into eigenvalues and eigenvectors. This is the eigenvalue decomposition. Let $\mathbf{A}$ be the $n \times n$ symmetric adjacency matrix of an undirected graph with n vertices. Then $\mathbf{A}$ can be written in the following way:

$$\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}} \tag{3}$$

where $\mathbf{U}$ is an $n \times n$ orthogonal matrix, and $\mathbf{\Lambda}$ is an $n \times n$ diagonal matrix. The values $\mathbf{\Lambda}_{ii}$ are the eigenvalues of $\mathbf{A}$, and the columns of $\mathbf{U}$ are its eigenvectors. We will designate the eigenvalues by $\lambda_i = \mathbf{\Lambda}_{ii}$ and the eigenvectors by $\mathbf{u}_i = \mathbf{U}_{\cdot i}$ for $1 \leq i \leq n$.

The multiset of all eigenvalues $\{\lambda_i\}$ of a matrix is called its *spectrum*. The same eigenvalue λ may appear multiple times in the spectrum. In practice, only rank-k approximations can be computed for k much smaller than the size n of the matrix. Typical values for k in link prediction systems are not larger than 100.

3 Motivation

The hypothesis that we study states that in terms of the eigenvalue decomposition of a networks's adjacency matrix $\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}$, growth can be described as a transformation of the spectrum $\mathbf{\Lambda}$, without significant change in the eigenvectors $\mathbf{U}$.

Each algebraic link prediction algorithm, including most graph kernels, represents a specific assumption about network growth, leading to a different spectral transformation function. Each of these link prediction functions can be applied in different situations, depending on the

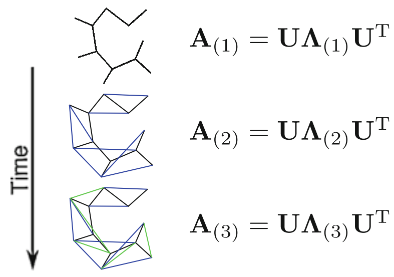

Time
$\mathbf{A}_{(1)} = \mathbf{U}\boldsymbol{\Lambda}_{(1)}\mathbf{U}^{\mathrm{T}}$
$\mathbf{A}_{(2)} = \mathbf{U}\boldsymbol{\Lambda}_{(2)}\mathbf{U}^{\mathrm{T}}$
$\mathbf{A}_{(3)} = \mathbf{U}\boldsymbol{\Lambda}_{(3)}\mathbf{U}^{\mathrm{T}}$

Fig. 1 Illustration of the spectral evolution hypothesis. As edges appear in a network, the eigenvalues of the network's adjacency matrix change, while the eigenvectors stay constant

assumptions made about the network. Despite this, all algebraic link prediction algorithms are of the same form: a change of the eigenvalues following a specific function, without change in the eigenvectors. Therefore, they are all generalized by the spectral evolution model. Starting with a network's adjacency matrix $\mathbf{A}$, we first compute its eigenvalue decomposition $\mathbf{A} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^{\mathrm{T}}$. It can then be shown that many common link prediction algorithms can be expressed as $\mathbf{U}F(\boldsymbol{\Lambda})\mathbf{U}^{\mathrm{T}}$, where F is a function that applies a real function $f(\lambda)$ to each diagonal element of $\boldsymbol{\Lambda}$.

Consider the following example: The matrix exponential of the adjacency matrix $\mathbf{A}$ is sometimes used as a link prediction function. The matrix exponential is defined as $\exp(\mathbf{A}) = \sum_{k=0}^{\infty} \mathbf{A}^k / k!$ and can be interpreted in the following way: $(\exp(\mathbf{A}))_{ij}$ equals a weighted sum over all paths from the node i to the node j in the network, where paths are weighted by the inverse of the factorial of their length. This description shows why the exponential of the adjacency matrix is a suitable link prediction function:

– The link prediction score is higher when there are many paths between i and j, because the powers of $\mathbf{A}$ count the number of paths.
– The link prediction score is higher when these paths are short, because the weights $1/k!$ are decreasing.

The matrix exponential can also be written in terms of the eigenvalue decomposition $\mathbf{A} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^{\mathrm{T}}$ as

$$\exp(\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^{\mathrm{T}}) = \mathbf{U}\exp(\boldsymbol{\Lambda})\mathbf{U}^{\mathrm{T}}, \tag{4}$$

where $\exp(\boldsymbol{\Lambda})$ can be computed by applying the real exponential function to $\boldsymbol{\Lambda}$'s diagonal elements. This shows that the matrix exponential is a spectral transformation of the adjacency matrix $\mathbf{A}$. We will see later that several other link prediction functions can be expressed as spectral transformations in an analogous way. First, however, we state the spectral evolution model.

Definition 1 (*Spectral evolution model*). A network that changes over time is said to follow the spectral evolution model when its spectrum evolves while its eigenvectors stay approximately constant.

This definition is not mathematically strict. Whether a network conforms to the spectral evolution model depends on the interpretation of when an eigenvector stays *approximately constant*. This fuzziness will be made precise in Sect. 6. For now, we will interpret the definition loosely, taking eigenvectors as approximately constant when their change is small. The spectral evolution model is visualized schematically in Fig. 1: In this simplified picture, a small network to which edges are added consecutively has three different spectra at three different times, but the same set of eigenvectors at all times.

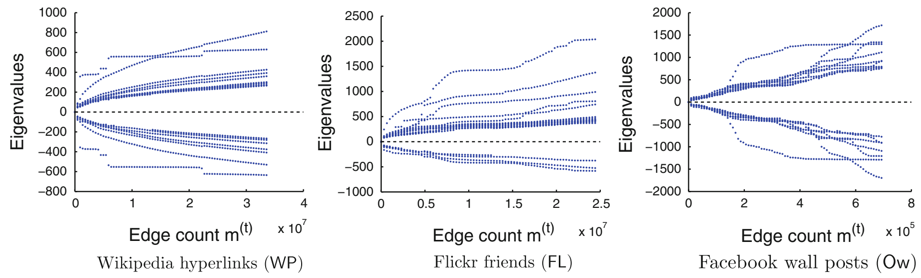


Fig. 2 The spectral evolution of large real-world networks in function of time. The X axis shows the number of edges m in the network at time t. The Y axis shows the top-k eigenvalues at time t

To examine the validity of the spectral evolution model, we take three steps. First, we verify it empirically in our collection of datasets. Then, we review existing link prediction functions that imply the spectral evolution model, and finally, we present control tests to verify that the spectral evolution model does not arise from other processes, such as random graph growth.

4 Empirical verification

Which network model accurately describes the growth of real networks? Given a network, which graph kernel best models its growth? To answer these questions, we examine the spectra of large, real-world networks and observe their temporal evolution. In the following sections, we look at the evolution of the eigenvalue decomposition of the networks given in Table 5 in the “Appendix”.

For each network, we split the set of edges into $T = 75$ bins by edge creation time. For each bin, we take the network as it was after the arrival of that bin's edges and compute the first k eigenvalues of the resulting adjacency matrix. In the networks we study in this paper, the creation times of edges are known. The set of edges is split into timeslices, and at each timeslice, the k dominant eigenvalues are computed. k is chosen in function of network size to give reasonable runtimes. We denote by $\mathbf{A}_{(t)}$ the adjacency matrix of all edges present after time t. Thus, $\mathbf{A}_{(T)} = \mathbf{A}$ is the full adjacency matrix.

Three representative datasets are used throughout this section:

– The English Wikipedia network, containing hyperlinks between articles (WP)
– The user–user friendship network of Flickr (FL)
– The Facebook user–user network of wall posts (Ow)

These three networks are unipartite, unweighted, and undirected, and link creation times are known for them. Only the Facebook wall post network has parallel edges. In other words, multiple edges may connect the same vertex pair in the Facebook graph.

4.1 Spectral evolution

Figure 2 shows the spectra of the three networks as functions of time. For each network, the plot shows the top-k eigenvalues $\mathbf{\Lambda}_{ii}$ by absolute value in function of time.

A first inspection of these plots suggests that the eigenvalues and singular values grow over time. The observed spectral growth of these networks is sometimes regular, as in the case of Wikipedia, or irregular, as in the case of Facebook. By regular growth, we mean that

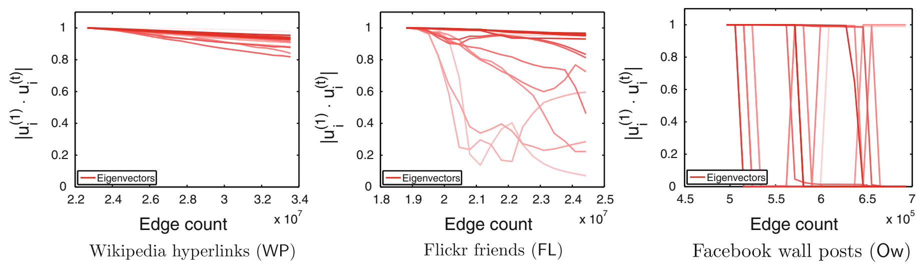


Fig. 3 The evolution of eigenvectors: The similarity between the eigenvectors of the full graph and the eigenvectors of partial graphs, by increasing number of missing edges. Each latent dimension is represented by one curve, with brighter colors for dominant latent dimensions

all eigenvalues grow roughly with the same speed. If growth is irregular as for the Facebook dataset, eigenvalues may overtake each other. In many datasets, the observed irregularity is only partial: Growth of eigenvalues as a whole does not seem to follow a specific pattern, although the growth of each individual eigenvalue is smooth. This is a first indication of the irregularity of network growth and will be a justification later on for extrapolating the eigenvalue growth into the future.

We cannot, however, know at this point that two consecutive eigenvalues with similar value are related to the same eigenvector. In fact, any continuous or small change to a matrix could lead to a small change in the spectrum. For the spectral evolution model to be valid, spectra must not only grow, but eigenvectors corresponding to individual eigenvalues must be stable. This is inspected in the next test.

4.2 Eigenvector evolution

To verify whether the spectral evolution model describes the growth of large, real networks, we compare for each t the eigenvectors of $\mathbf{A}_{(t)} = \mathbf{U}_{(t)}\mathbf{\Lambda}_{(t)}\mathbf{U}_{(t)}^{\mathrm{T}}$ with the eigenvectors of $\mathbf{A}_{(T)} = \mathbf{U}_{(T)}\mathbf{\Lambda}_{(T)}\mathbf{U}_{(T)}^{\mathrm{T}}$ in Fig. 3. In these plots, we rely on the eigenvectors to be ordered by their corresponding eigenvalue. The plots show one curve for each latent dimension i, showing the number of edges added between times T and t on the X axis and the absolute dot product of the ith eigenvector at times t and T on the Y axis. The similarity between eigenvectors is computed as the cosine distance in the following way:

$$\mathrm{sim}_{ij}(T,t) = \left|\mathbf{U}_{(T)i\cdot}^{\mathrm{T}}\mathbf{U}_{(t)j\cdot}\right| \tag{5}$$

The figure shows, for each latent dimension i, the curve of $\mathrm{sim}_{ii}(T,t)$ in function of t. Eigenvectors corresponding to large eigenvalues are shown in bright colors and those corresponding to smaller eigenvalues in light colors.

The eigenvector evolution plots suggest the following interpretation. The general trend leaves the eigenvector similarity as measured by the dot product near one for the lifetime of the network, with similarity being higher for those eigenvectors with larger eigenvalues. In some networks such as Facebook, the similarity of eigenvectors suddenly drops to zero at specific points in time. As we will see next, this is due to eigenvectors exchanging places in the network, or in other words, the eigenvalues changing order. The next test will inspect these permutations.

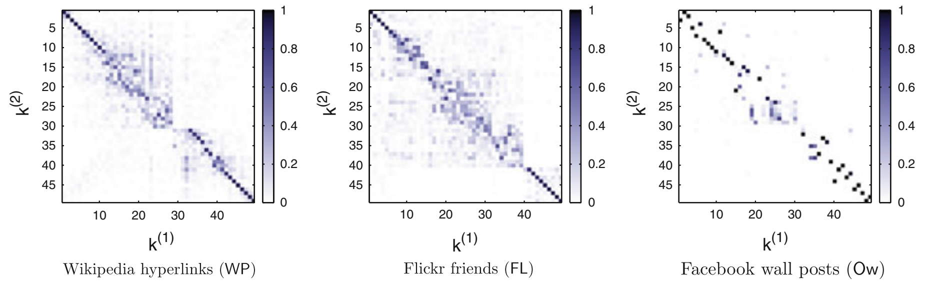


Fig. 4 The absolute dot product of all eigenvector pairs at two times in network evolution. These plots show permutation matrices (with 0 in *white* and 1 in *black*) when the network evolution is purely spectral and eigenvalues are simple. The derivation from a diagonal matrix gives an indication to the monotony of the underlying spectral transformation

4.3 Eigenvector stability

How stable are eigenvectors over time? To answer this question we compute the absolute eigenvalues of eigenvector pairs at two times t_a and t_b. In this section, t_a will be the time when 75 % of edges are present in the network, and t_b will be the time when all edges are present.

We will call $\mathbf{A}_a$ the adjacency matrix at time t_a and $\mathbf{A}_b$ the adjacency matrix at time t_b. We consider the following eigenvalue decompositions:

$$\mathbf{A}_a = \mathbf{U}_a \mathbf{\Lambda}_a \mathbf{U}_a^{\mathrm{T}} \tag{6}$$

$$\mathbf{A}_b = \mathbf{U}_b \mathbf{\Lambda}_b \mathbf{U}_b^{\mathrm{T}} \tag{7}$$

We then compute $\mathrm{sim}_{ij}(t_a, t_b)$ for all pairs (i, j). We show the resulting matrices using white for zero and black for one, and continuous shades in between in Fig. 4. These plots give an indication as to what extent eigenvectors are preserved over time. If all eigenvalues are distinct and network evolution is purely spectral, the matrices $\mathrm{sim}_{ij}(t_a, t_b)$ are permutation matrices. In addition, they are diagonal when the latent dimensions do not overtake each other. The derivation from a diagonal matrix gives an indication to the monotony of the underlying spectral transformation.

Testing the eigenvector stability in this way has one drawback. If two eigenvalues are equal, an exchange between their eigenvectors does not change the matrix. This case can be recognized on the eigenvector permutation plots of Fig. 4 as sub-squares containing intermediate values between zero and one. These sub-squares are in fact arbitrary orthogonal matrices, since any orthogonal basis of the multidimensional eigenspace corresponding to a multiple eigenvalue can be returned by the eigenvalue decomposition. To avoid this, the next test is designed to be robust against such multiple eigenvalues.

4.4 Spectral diagonality test

We now present a test of the amount of change in the eigenvectors which we call the spectral diagonality test. If

$$\mathbf{A}_a = \mathbf{U}_a \mathbf{\Lambda}_a \mathbf{U}_a^{\mathrm{T}} \tag{8}$$

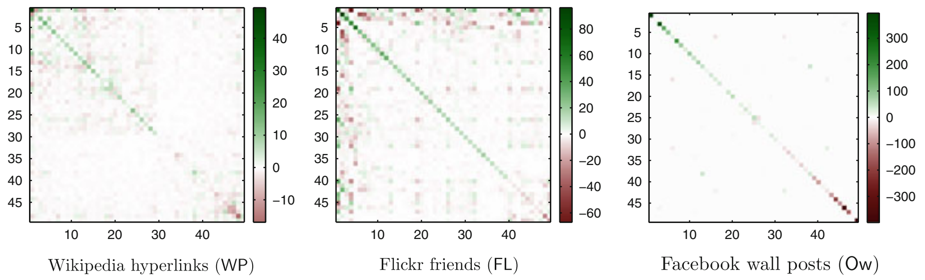


Fig. 5 The diagonality test matrix $\mathbf{\Delta}$ of spectral network evolution. If network evolution is perfectly spectral, the plots show a clean diagonal

is the eigenvalue decomposition of a network's adjacency matrix at time t_a, then at a later time t_b it is assumed to become

$$\mathbf{A}_b = \mathbf{U}_a(\mathbf{\Lambda}_a + \mathbf{\Delta})\mathbf{U}_a^{\mathrm{T}}, \tag{9}$$

where $\mathbf{\Delta}$ is diagonal. Because $\mathbf{U}_a$ has orthogonal columns, we can compute the best fit of $\mathbf{\Delta}$ in a least-squares sense by

$$\mathbf{\Delta} = \mathbf{U}_a^{\mathrm{T}}(\mathbf{A}_b - \mathbf{A}_a)\mathbf{U}_a. \tag{10}$$

The matrix $\mathbf{\Delta}$ is intended to give an indication to what extent growth is spectral. It is shown for the three example networks in Fig. 5. We can make several observations: First, all three matrices are nearly diagonal. In fact, we make this observation for all datasets in our collection. However, the deviation from a diagonal matrix is not equal for all datasets. For the Facebook wall posts network (Ow), the matrix is very near to a diagonal matrix, and thus, the growth is very nearly spectral. For the Flickr friendship network (FL), the deviation is larger. This is a pattern that we observed in almost our entire collection of datasets: Datasets where multiple edges are allowed mostly have better spectral growth than those where only simple edges exist. A simple explanation can be given for communication networks such as email networks: In these networks, an edge will appear between already connected nodes with a certain probability, making growth more spectral. In fact, if edges appeared exactly by this probability, the adjacency matrix would simply be multiplied by a constant, and growth would thus be spectral.

5 Generalizing other models

In the previous section, we have observed that the spectral evolution model holds in many large, real network datasets. In this section, we will derive the spectral evolution model from other, more specific graph growth models. As we will see, many common graph growth models implicitly rely on spectral growth. In other words, the spectral evolution model can be understood as a generalization of the models presented in this section. We will look at these models and study how they can be expressed as a change of a network's eigenvalue decomposition.

5.1 Triangle closing

Arguably, the simplest graph growth model is the *triangle closing* model. This model predicts that in a graph, new edges will appear between nodes that have common neighbors, thus forming ("closing") triangles. In social networks, the triangle closing model is known as the *friend of a friend* model. The triangle closing model is one of the easiest to compute nontrivial link prediction methods and is used in practice on very large datasets [35].

Algebraically, this model can be expressed as the square $\mathbf{A}^2$ of the adjacency matrix. The matrix $\mathbf{A}^2$ contains, for each vertex pair (i, j), the number of common neighbors of i and j:

$$\left(\mathbf{A}^2\right)_{ij} = \sum_k \mathbf{A}_{ik}\mathbf{A}_{jk} \tag{11}$$

The triangle closing model will thus predict those links that complete the highest number of triangles. The square $\mathbf{A}^2$ can be expressed using the eigenvalue decomposition $\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}$ as

$$\mathbf{A}^2 = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}\mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}} = \mathbf{U}\mathbf{\Lambda}^2\mathbf{U}^{\mathrm{T}}. \tag{12}$$

Here, we use the fact that $\mathbf{U}$ is orthogonal and thus $\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}$. As we see, the eigenvalues $\mathbf{\Lambda}$ are replaced by their squares $\mathbf{\Lambda}^2$. The triangle closing model is thus a spectral transformation. Note that this is also true when using the reduced eigenvalue decomposition of $\mathbf{A}$. Since $\mathbf{\Lambda}$ is diagonal, its square can be computed by squaring each of its diagonal element. This equation shows that the triangle closing model $\mathbf{A}^2$ is a spectral transformation that corresponds to the real function $f(\lambda) = \lambda^2$.

5.2 Path counting

The triangle closing model states that edges will appear between nodes separated by a path of length two. In real networks, however, we also may expect edges to connect two vertices that are more than two edges apart. We will thus suppose that new edges complete cycles of any length, but give lower weight to longer cycles [32]. The number of paths of a given length k between any two vertices of a graph can be expressed by the kth power of its adjacency matrix $\mathbf{A}$. By the same argument as for $\mathbf{A}^2$, we can derive that $\mathbf{A}^k$ is a spectral transformation of $\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}$:

$$\mathbf{A}^k = \mathbf{U}\mathbf{\Lambda}^k\mathbf{U}^{\mathrm{T}} \tag{13}$$

Thus, the kth power of the adjacency matrix corresponds to the spectral transformation λ^k. Because matrix multiplication is distributive, a linear combination of powers of $\mathbf{A}$ is also a spectral transformation. In general, every polynomial $p(\mathbf{A})$ is a spectral transformation:

$$p(\mathbf{A}) = \mathbf{U}p(\mathbf{\Lambda})\mathbf{U}^{\mathrm{T}} \tag{14}$$

$$\alpha\mathbf{A}^2 + \beta\mathbf{A}^3 + \gamma\mathbf{A}^4 + \cdots = \mathbf{U}(\alpha\mathbf{\Lambda}^2 + \beta\mathbf{\Lambda}^3 + \gamma\mathbf{\Lambda}^4 + \cdots)\mathbf{U}^{\mathrm{T}} \tag{15}$$

In particular, a polynomial $p(\mathbf{A})$ can be used as a graph growth model whenever its coefficients are nonnegative and decreasing, that is, if $\alpha > \beta > \gamma > \cdots \geq 0$. Polynomials of this form have two desirable properties of link prediction functions:

- Vertices connected by more paths are more likely to connect in the future (positivity of the coefficients).
- Vertices connected by shorter paths are more likely to connect in the future (ordering of the coefficients).

Note that we do not consider the zeroth and first powers of $\mathbf{A}$: They only contribute to additive constant for all node pairs.

5.3 Graph kernels

A kernel is a symmetric and positive-semidefinite function of two variables that denotes a similarity or proximity between objects. A graph kernel is defined for a given graph and denotes the similarity between any two vertices of that graph. Many graph kernels exist, most being derived from specific graph models and typically applied to different applications [15,21,27,54]. Since they denote a form of similarity between two vertices, graph kernels can be used for link prediction. Note that the null space of a matrix $\mathbf{A}$, that is, the set of vectors $\mathbf{x}$ such that $\mathbf{Ax} = \mathbf{0}$, is also called the *kernel* of $\mathbf{A}$. This meaning of the word *kernel* is not used in this text.

Many graph kernels are spectral transformations. Each of these graph kernels can be described by a symmetric positive-definite matrix $K(\mathbf{A})$ of the same size as the graph's adjacency matrix $\mathbf{A} = \mathbf{U\Lambda U}^{\mathrm{T}}$, and that can be written as $K(\mathbf{A}) = \mathbf{U}F(\mathbf{\Lambda})\mathbf{U}^{\mathrm{T}}$ for various functions $F(\mathbf{\Lambda})$ which apply a real function $f(\lambda)$ to each eigenvalue in $\mathbf{\Lambda}$. In addition to being spectral transformations, these graph kernels' functions $f(\lambda)$ are convex, which implies that large latent dimensions grow quicker than smaller latent dimensions. By construction, these graph kernels can be computed from the eigenvalue decomposition of $\mathbf{A}$. Because the graph kernels we describe here are based on the adjacency matrix $\mathbf{A}$, we will call them *adjacency kernels*.

Matrix Exponential The matrix exponential of the adjacency matrix is a graph kernel that can be used as a link prediction function [27]. A scale parameter α is typically inserted to balance the relative weight of short and long paths.

$$K_{\mathrm{EXP}}(\mathbf{A}) = \exp(\alpha\mathbf{A}) = \sum_{k=0}^{\infty} \alpha^k \frac{1}{k!}\mathbf{A}^k \tag{16}$$

It corresponds to the spectral transformation $f(\lambda) = e^{\alpha\lambda}$. When written as a power series, we see that paths are weighted by the inverse factorial of their length. The matrix exponential has nonnegative decreasing coefficients and is therefore suited to link prediction. The matrix exponential is also called the exponential diffusion kernel.

The Neumann Kernel The Neumann kernel is given by

$$K_{\mathrm{NEU}}(\mathbf{A}) = (\mathbf{I} - \alpha\mathbf{A})^{-1} = \sum_{k=0}^{\infty} \alpha^k \mathbf{A}^k, \tag{17}$$

where α is chosen such that $\alpha^{-1} > |\lambda_1|$, $|\lambda_1|$ being $\mathbf{A}$'s largest absolute eigenvalue, or equivalently the graph's spectral norm [23]. The resulting spectral transformation is $f(\lambda) = 1/(1 - \alpha\lambda)$. As the matrix exponential, the Neumann kernel can be written as a power series with nonnegative decreasing coefficients and is therefore a suitable link prediction function. The Neumann kernel is sometimes called the Neumann diffusion kernel [15]. The link prediction function resulting from the Neumann kernel is also called the Katz index [24].

Other Graph Kernels Some graph kernels such as the commute-time kernel are the spectral transformation of other characteristic graph matrices, such as the graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$ or the normalized adjacency matrix $\mathbf{N} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. These kernels are, however, better suited for spectral clustering and less for link prediction [52].

Using the Laplacian $\mathbf{L}$, several spectral transformations are known in the literature. In [8] for instance, they are called transfer functions and are used for semi-supervised learning, that is, learning the labels of unlabeled nodes in a graph, when some node labels are known. In these studies, only polynomial and stepwise linear transfer functions are considered.

5.4 Rank reduction

For large graphs, the eigenvalue and singular value decompositions can only be computed up to a small rank k, in practice not larger than about 100. The result is an approximation to the adjacency matrix $\mathbf{A}$ that uses only the k dominant eigenpairs.

$$R_k(\mathbf{A}) = \hat{\mathbf{U}}\hat{\mathbf{\Lambda}}\hat{\mathbf{U}}^{\mathrm{T}} \tag{18}$$

where $\hat{\mathbf{U}} = \mathbf{U}_{\cdot\, 1:k}$ and $\hat{\mathbf{\Lambda}} = \mathbf{\Lambda}_{1:k\; 1:k}$ are the corresponding eigenvector and eigenvalue matrices reduced to the largest k eigenvectors by absolute value, assuming that the diagonal elements $\lambda_i = \mathbf{\Lambda}_{ii}$ are in decreasing order by absolute value, that is, $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n|$. This is the best rank-$k$ approximation of $\mathbf{A}$ in a least-squares sense.

Although rank reduction appears to be a necessary but unwanted approximation when doing link prediction using spectral methods, some studies have used rank reduction itself as the link prediction method, showing that a lower rank than originally computed may result in more accurate link prediction [20,53]. To see how this is possible, consider the entry (i, j) in the adjacency matrix $\mathbf{A}$, which is zero if i and j are not connected. In the matrix $R_k(\mathbf{A})$, however, this entry is not zero and can be used as a link prediction score for the edge $\{i, j\}$. In methods such as latent Dirichlet allocation [3] and nonnegative matrix factorization [34] where a latent model is considered, rank reduction is also implicit. Only k latent dimension of the eigenvalue decomposition can be computed due to the size of the problem, but computing all n would produce an error-free approximation of the original matrix $\mathbf{A}$, making any score for nonedges equal to zero.

Reduction to a rank k can be interpreted as the spectral transformation

$$f(\lambda) = \begin{cases} \lambda & \text{if } |\lambda| \geq |\lambda_k| \\ 0 & \text{otherwise.} \end{cases} \tag{19}$$

A theoretical ambiguity arises when several eigenvalues have absolute value $|\lambda_k|$, which is usually ignored by ordering corresponding eigenvectors arbitrarily. In practice, the size of networks and the small value of k make that case very unlikely. This k is chosen to yield a similar, reasonable runtime for each dataset. As a result, k is larger for smaller datasets.

5.5 Latent preferential attachment

Preferential attachment is a simple link prediction model based on the idea that the probability of a new link being formed is proportional to the degrees of the nodes it connects. This idea can be extended to the decomposition of a graph's adjacency matrix, resulting in the latent preferential attachment model, which we describe here. In this section, we show that the latent preferential attachment model is equivalent to the spectral evolution model.

In a graph with adjacency matrix $\mathbf{A}$, the number of neighbors of a node i is called the degree of i and can be written as $d(i) = \sum_j \mathbf{A}_{ij}$. In the preferential attachment model, the probability of an edge appearing between the vertices i and j is proportional to both $d(i)$ and $d(j)$. In other words, links are formed with preference to nodes that already have a high number of neighbors. The preferential attachment model leads to networks where few nodes have many neighbors and many nodes have few neighbors. The distributions of node degrees in such networks can be described by *power laws*, meaning that the probability that a node has n neighbors is proportional to $n^{-\gamma}$ for a constant $\gamma > 1$. Typical values for γ are in the range [2, 3], although many other values larger than one are observed in practice. Such networks are called scale-free networks [2], and most real-world networks are of this form.

Now let us consider the eigenvalue decomposition of the adjacency matrix $\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}$. This decomposition can be used to write $\mathbf{A}$ as a sum of rank-one matrices:

$$\mathbf{A} = \sum_{k=1}^{r} \lambda_k \mathbf{u}_k \mathbf{u}_k^{\mathrm{T}} \tag{20}$$

where r is the rank of the decomposition, $\lambda_k = \mathbf{\Lambda}_{kk}$ are the eigenvalues, and $\mathbf{u}_k = \mathbf{U}_{\cdot k}$ are the eigenvectors of $\mathbf{A}$.

The usual interpretation of a matrix factorization is that each latent dimension k represents a *topic* in the network. Then $\mathbf{U}_{ik}$ represents the importance of vertex i in topic k, and λ_k represents the overall importance of topic k. Each rank-one matrix $\mathbf{A}^{(k)} = \lambda_k \mathbf{u}_k \mathbf{u}_k^{\mathrm{T}}$ can now be interpreted as the adjacency matrix of a weighted graph. This graph G_k will be the complete graph on n vertices, since $\mathbf{u}_k$ is typically not sparse when the network is connected. Therefore, all information about this graph is contained in the weights of the edges. Now, assume that preferential attachment is happening in the network, but restricted to one latent dimension k. Then the probability of the edge $\{i, j\}$ appearing will be proportional to $d_k(i)d_k(j)$, where $d_k(i)$ is the degree of node i in the graph G_k. This degree can be written as the sum over edge weights in G_k:

$$d_k(i) = \sum_j \mathbf{A}_{ij}^{(k)} = \sum_j \lambda_k \mathbf{U}_{ik} \mathbf{U}_{jk} = \mathbf{U}_{ik} \lambda_k \sum_j \mathbf{U}_{jk} \sim \mathbf{U}_{ik} \tag{21}$$

In other words, $d_k(i)$ is proportional to $\mathbf{U}_{ik}$ only, since $\lambda_k \sum_j \mathbf{U}_{jk}$ is independent of i. Therefore, the preferential attachment value is proportional to the corresponding entry in $\mathbf{A}^{(k)}$:

$$d_k(i)d_k(j) \sim \mathbf{U}_{ik}\mathbf{U}_{jk} \tag{22}$$

These values can be aggregated into a matrix $\mathbf{P}^{(k)}$ giving the preferential attachment values for all pairs (i, j):

$$\mathbf{P}^{(k)} \sim \mathbf{u}_k \mathbf{u}_k^{\mathrm{T}} \tag{23}$$

If we now assume that preferential attachment is happening in each latent dimension separately, with a weight ϵ_k depending on the topic k, then the overall preferential attachment prediction can be written as:

$$\mathbf{P} = \sum_k \epsilon_k \mathbf{u}_k \mathbf{u}_k^{\mathrm{T}} \tag{24}$$

Here, we replace proportionality by equality since the proportionality constants are absorbed by the constants ϵ_k. The matrix $\mathbf{P}$ can then be written in the following form, giving its eigenvalue decomposition:

$$\mathbf{P} = \mathbf{U}\mathbf{E}\mathbf{U}^{\mathrm{T}} \tag{25}$$

where $\mathbf{E}$ is the diagonal matrix containing the individual topic weights $\mathbf{E}_{kk} = \epsilon_k$. This prediction matrix is a spectral transformation of the adjacency matrix $\mathbf{A}$. Under this model, network growth can be interpreted as the replacement of the eigenvalues $\mathbf{\Lambda}$ by $\mathbf{\Lambda} + \mathbf{E}$:

$$\mathbf{A} + \mathbf{P} = \mathbf{U}(\mathbf{\Lambda} + \mathbf{E})\mathbf{U}^{\mathrm{T}} \tag{26}$$

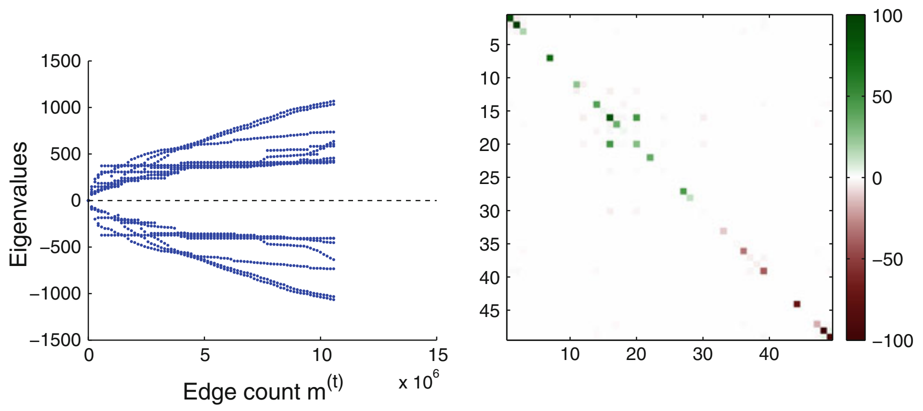


Fig. 6 Example of irregular but spectral network evolution in the Twitter user–user "@" mention network (Ws). **a** The evolution of the networks' eigenvalues over time, **b** the spectral diagonality test applied to a 75 %/25 % split of the network. Although the diagonality tests show that growth is spectral in this case, the spectral growth is irregular. The growth of this network can thus not be well described by any of the common graph kernels

Since the values **E** are not predicted by the latent preferential attachment model, every spectral transformation can be interpreted as latent preferential attachment, and thus, the latent preferential attachment model is equivalent to the spectral evolution model.

5.6 Irregular growth

We have seen that the spectral evolution model is a generalization of several graph growth models such as graph kernels. If these existing graph growth models already describe the growth of graphs, why do we then need to generalize them? The answer to that question is that most graph kernels and other growth models are only valid to a certain extent. By inspecting the growth of actual networks, we can see why these graph kernels cannot be universally applied for link prediction. Several cases of observed irregular spectral evolution are shown in Fig. 6.

In these plots, we see that every eigenvalue grows at its own speed. As a result, the spectral transformation function $f(\lambda)$ that gives new eigenvalues as a function of old ones is not regular. This kind of spectral growth cannot be well described by a function f that is monotonous like a graph kernel. Instead, each latent dimension is observed to grow at a specific speed. Despite this, the evolution of these networks is still spectral. It is only the spectral transformation that is irregular.

From this and other examples, we observe different possible behaviors of eigenvalue evolution. Some eigenvalues grow linearly, and others do not grow. In the same network, different eigenvalues may grow at different speeds. However, we did not encounter any case of an eigenvalue shrinking over time, even though this is not excluded by the addition of a single edge.[1] This behavior may have many explanations, which are usually hard to guess. In most network datasets, nodes are not labeled, and so we cannot inspect the eigenvectors in

[1] This can be seen by noting that a single edge has the adjacency matrix [0, 1; 1, 0], which is indefinite, and in fact only by adding a positive-semidefinite component to a matrix can it be guaranteed that eigenvalues do not shrink; this follows from the interlacing theorem given in [62, p. 97].

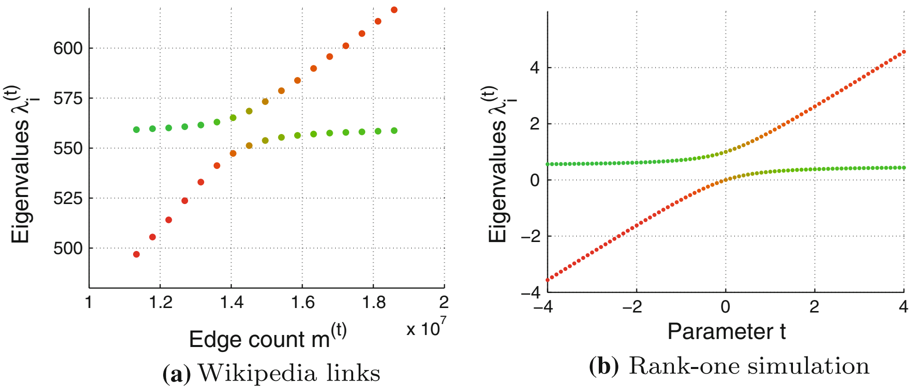


Fig. 7 An avoided crossing as observed in the growth of the Wikipedia hyperlink network's (WP) two dominant eigenvalues and eigenvectors in (**a**), and in a rank-one model in (**b**). The *red* and *green* color components of the points denote the cosine distance of eigenvectors to initial eigenvectors (color figure online)

question to see which vertices have the highest value in them. We may, however, conjecture that stagnating latent dimensions correspond to communities that die out, or other aspects of the network that do not grow anymore, for whatever reason. Although the reason for each latent dimension's growth is unknown, it can still be learned, as we will do in the next section. In fact, these properties make link prediction algorithms based on spectral transformations more universal, since they do not rely on domain-specific knowledge of the network.

5.7 Avoided crossings

As observed in several examples, a latent dimension can overtake another. In these cases, our model predicts *crossings* in the spectral plot. Looking at actual spectra, we, however, observe something slightly different: The smaller, growing eigenvalue slows down growth before it reaches the larger eigenvalue, and the larger eigenvalue starts growing at about the same rate. We observe this behavior in several datasets, as shown in Fig. 7.

This phenomenon is called an *avoided crossing* and can be explained as follows [33, 8.5]. Let $\mathbf{A}$ and $\mathbf{B}$ be two square symmetric rank-one matrices of the same size. Then the eigenvalues of $\mathbf{A} + t\mathbf{B}$ for a real parameter t display the avoided crossing behavior. This behavior is explained by the fact that the two eigenvectors of $\mathbf{A}$ and $\mathbf{B}$ are not orthogonal in the general case, and by considering the dimension of matrices with multiple eigenvalues.

In the rank-one case, let $\mathbf{A} = \alpha\mathbf{a}\mathbf{a}^{\mathrm{T}}$ and $\mathbf{B} = \beta\mathbf{b}\mathbf{b}^{\mathrm{T}}$, then α and $t\beta$ are only eigenvalues of $\mathbf{A} + t\mathbf{B}$ if $\mathbf{a}$ and $\mathbf{b}$ are collinear or orthogonal. Otherwise, an avoided crossing is observed as in Fig. 7(a). Thus, an avoided crossing indicates that a decomposition into outer products of orthogonal vectors is not the natural representation for some networks. If the true decomposition using $\mathbf{a}$ and $\mathbf{b}$ is used, the crossing would be *unavoided*. Alternatively, an avoided crossing may result from a perturbation of orthogonal latent dimensions in the same manner.

6 Control tests

We have seen in the previous sections that spectral growth is observed in actual networks and that it can be explained by several common graph growth models. To make sure that the validity of the spectral evolution model is in itself a nontrivial property of real networks, we have to verify that it does *not* follow from other random processes. We will verify this for two fundamental random processes: The random addition of edges and the random sampling of edges. To show that the spectral evolution model is not trivial, it must not be observed in these trivial models. Note that the property of nontriviality is not observed for all graph characteristics. Some advanced characteristics such as modularity can be observed in the simplest of random graph models, the Erdős–Rényi model [18].

6.1 Random graph growth

The first random growth model we investigate consists of adding edges randomly to a given graph. Let $\mathbf{A}_{(t)} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}$ be the adjacency matrix of a network, and $\mathbf{A}_{(t+1)} = \mathbf{A}_{(t)} + \mathbf{E}$ a perturbation of $\mathbf{A}_{(t)}$ where $\|\mathbf{E}\|_F = \epsilon$ is small and $\mathbf{E}$ can be thought of as an infinitesimally small edge added to the network. A perturbation argument then shows that the eigenvalue decomposition of $\mathbf{A}_{(t+1)} = \tilde{\mathbf{U}}\tilde{\mathbf{\Lambda}}\tilde{\mathbf{U}}^{\mathrm{T}}$ has the following bounds [55]:

$$\|\mathbf{\Lambda} - \tilde{\mathbf{\Lambda}}\|_{\mathrm{F}} = O(\epsilon^2) \tag{27}$$

$$|\mathbf{U}_{\cdot k}^{\mathrm{T}}\tilde{\mathbf{U}}_{\cdot k}| = O(\epsilon) \tag{28}$$

These two expressions can be derived from theorems by Weyl and Wedin [61,60].

As a result, eigenvectors are expected to change faster than eigenvalues for random additions to the adjacency matrix. We were able to confirm this prediction in synthetic random tests, which shows that spectral growth is not explained by a random graph model and is thus a nontrivial emergent property of real-world networks.

6.2 Random sampling

In this test, we generate an Erdős–Rényi random graph and split its edges randomly into two sets. We then test how well a spectral transformation maps one edge set to the other.

We use the following setup: Let $G = (V, E)$ be a randomly generated graph with $|V| = 1{,}000$, and the probability of each possible existing edge is chosen; thus, each vertex has on average three neighbors. We make an 75 %/25 % random split into vertex sets E_a and E_b represented by the adjacency matrices $\mathbf{A}$ and $\mathbf{B}$. These proportions correspond to the training/test split used in all experiments in this paper. We then compute the reduced eigenvalue decomposition $\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}$ of rank 100.

Figure 8 shows two tests. First, we plot the eigenvalues of $\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}$ against the diagonal elements of $\mathbf{U}^{\mathrm{T}}\mathbf{B}\mathbf{U}$. Then, we perform the spectral diagonality test of Sect. 4.4, showing the matrix $\mathbf{U}^{\mathrm{T}}\mathbf{B}\mathbf{U}$.

Observations In the spectral transformation plot (a), there is clearly only one eigenvalue with a significant nonzero value after random sampling. This single latent dimension can be interpreted as a form of preferential attachment (see Sect. 5.5) and is explained by the fact that choosing edges at random will return edges adjacent to a node i with probability proportional to the degree of i. Apart from that, no structure of the original network is preserved spectrally. Indeed, the diagonality test (b) shows that no linear combination of the leading eigenvectors can predict the sampled edge set, except for the dominant eigenvector which represents preferential attachment. We conclude that a spectral transformation of rank greater than one is a nontrivial emergent property of actual network growth.

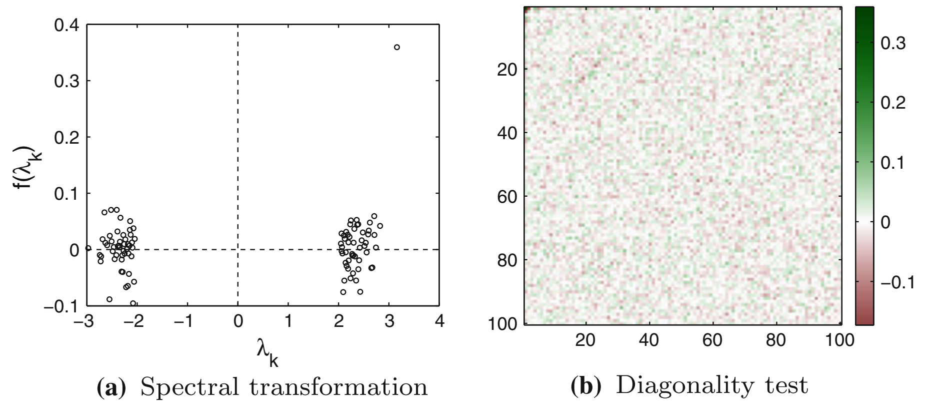


Fig. 8 The spectral transformation generated by random sampling from an Erdős–Rényi graph. **a** The spectral transformation resulting from random sampling, **b** diagonality test resulting from random sampling. Both tests show that only one latent dimension is preserved, showing that random sampling does not result in spectral transformations

7 Learning spectral transformations

In the last section, we have observed network growth to follow the *spectral evolution model*. In this and the next section, we present two ways to exploit the spectral evolution model for solving the problem of link prediction.

The main idea for link prediction under the spectral evolution model consists in predicting future values of a network's eigenvalues, and retaining its eigenvectors. The first method we present takes the approach that graph kernels are correct descriptions of network growth and learns the parameters of these graph kernels using curve fitting. The second method assumes that the growth of eigenvalues does not follow any pattern, and therefore, graph kernels cannot give accurate link prediction results. Instead, the second method extrapolates the change of eigenvalues into the future. We will describe both methods for unweighted, undirected, unipartite networks.

To find a spectral transformation that is best, we must first define at what task it should be best. The task we consider in this paper is the task of link prediction. Thus, we first introduce the link prediction problem and then describe our two learning methods.

An important class of applications using networks is recommender system. In recommender systems, the task at hand can usually be formulated as a problem of link prediction. In a general sense, *link prediction* denotes the problem of predicting links in a network. Link prediction is usually considered to be the most general type of problem that can be formulated on networks and is typically hard to solve. Recommender systems in the scope of this work are found in many data mining applications:

– A search engine finds documents matching a query. By modeling documents and the words they contain as a bipartite network, matching documents to a query corresponds to finding highly scored links between the query words and documents.
– A recommender system can be modeled as a network containing both users and items. Recommendations are then found by predicting links from users to items. The recommender network may connect users and items directly such as with ratings, or indirectly, for example, through topics, categories, user history, sensors, etc.

- Context-aware recommender systems additionally include a context for each query, which can be modeled as links between the query and the contextual elements. The task is then to find links between the query and entities to recommend.
- Rating prediction is a special case of link prediction, where edges are weighted. An important application is collaborative filtering, with users rating items.
- Finding related items can be achieved by predicting links between items connected to a network, even if there are no direct links between the items.
- Finding similar nodes in the network, for e.g. to recommend new friends in a social network. In this case edges are unweighted, and the links to be predicted describe the similarity nodes.
- Predicting edge weights, for e.g. to predict whether a user will like or not like a given movie. In this case the network is a bipartite rating graph, and missing links in the network have to be predicted.
- Predicting future interaction, for e.g. predicting e-mails or scientific coauthorship. In this case link prediction must be performed in a network with multiple edges.

In all these problems, there is a network optionally weighted by real numbers, and the problem consists of predicting future edges. In particular, we can distinguish the following types of link prediction:

- Given a network, predict which edges will appear: link detection or link completion [16]
- Given a network and a specific node, predict to which other nodes it will connect: recommendation [19]
- Given a network and a pair of unconnected nodes, determine whether it will be connected: link prediction proper [39]
- Given a weighted network and two unconnected nodes, determine the weight of an edge connecting them, knowing that an edge will appear between them: collaborative filtering [5], predicting trust [17], link sign prediction [30]

These problems can be solved by considering link prediction functions: functions of node pairs returning a numerical score. The different machine learning problems given above can then be distinguished by the way these numbers are interpreted, and how they are compared to the actual networks. Computed scores for a node pair can be used as prediction values directly or used to find a ranking, which is then compared with the actual known ranking.

7.1 Local link prediction methods

This section describes several simple link prediction methods that do not make use of spectral graph theory. Due to their simplicity, these methods are very widely used and will serve as a baseline against which complex link prediction methods are evaluated. These link prediction functions compute a link prediction score for a node pair using only information about the immediate neighborhood of the two nodes. We will therefore call these functions local link prediction functions [39].

Let i and j be two nodes in the graph for which a link prediction score is to be computed. To compute a link prediction score for the edge $\{i, j\}$, local link prediction functions depend only on the neighbors of i and j. In contrast to this, the spectral link prediction methods described in the main part of this work take into account the whole network. In other words, they are global.

Common Neighbors The number of common neighbors can be used in itself as a link prediction function [39]. In the example of social recommendations in a user–user network, this implements the *friend of a friend* principle and is equivalent to recommending nonfriends that have the highest number of common friends.

$$\text{CommonNeighbors}(i, j) = |\{k|k \sim i \wedge k \sim j\}| \tag{29}$$

The number of common neighbor is a spectral transformation. It is equivalent to the triangle closing model described in Sect. 5.1.

Adamic–Adar The measure of Adamic and Adar counts the number of common neighbors, weighted by the inverse logarithm of each neighbor's degree [1].

$$\text{AdamicAdar}(i, j) = \sum_{k \sim i,j} \log(d(k))^{-1} \tag{30}$$

This can be interpreted as a weighted variant of the common neighbors model.

Jaccard The Jaccard coefficient measures the amount of common neighbors divided by the number of neighbors of either vertex [39].

$$\text{Jaccard}(i, j) = \frac{|\{k|k \sim i \wedge k \sim j\}|}{|\{k|k \sim i \vee k \sim j\}|} \tag{31}$$

The Jaccard coefficient too can be considered a weighted variant of the common neighbors model.

Preferential Attachment Taking only the degree of i and j into account for link prediction leads to the *preferential attachment* model [2], which can be used as a model for more complex methods such as modularity kernels [50,64]. If $d(i)$ is the number of neighbors of node i, the preferential attachment model gives a prediction between i and j of

$$\text{PreferentialAttachment}(i, j) = \frac{1}{2|E|} d(i)d(j). \tag{32}$$

The factor $1/(2|E|)$ normalizes the sum of predictions for a vertex to its degree. As we saw in Sect. 5.5, a generalization of the preferential attachment model is equivalent to the spectral evolution model.

7.2 Normalization

Both local and global link prediction algorithms usually benefit from normalization. Normalization consists of applying a transformation to the data before applying a link prediction algorithm. A typical example of normalization is performed for rating prediction: The overall mean rating is subtracted from all ratings. After predictions have been computed, the overall mean is then added to all predicted ratings. We will call this type of procedure *additive normalization*. The adjacency matrix $\mathbf{A}$ is in many applications replaced by $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. This amounts to replacing each entry $\mathbf{A}_{ij} = 1$ by $1/\sqrt{d(i)d(j)}$. We will call this procedure *multiplicative normalization*.

Both types of normalization can be applied to almost all link prediction algorithms. In fact, they can even be applied to the trivial link prediction algorithm that always predicts 1 or to the trivial rating prediction algorithm that always predicts 0:

– Always predicting 1 in combination with multiplicative normalization leads to the preferential attachment model: Predict $d(i)d(j)$ for the edge $\{i, j\}$.
– Always predicting 0 in combination with additive normalization leads to the global mean prediction: Always predict the global mean rating.

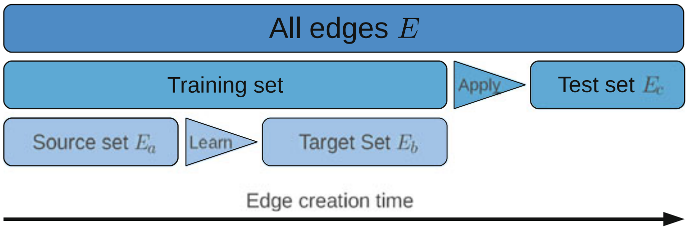


Fig. 9 The three-way split used in all experiments. The dataset is first split into a training and test set by edge creation time. Then, the training set is again split into a source and target set of edges. Then, a spectral transformation is learned from the source to the target set. This spectral transformation is then applied to the training set. Finally, the predicted edges are compared with the edges in the test set

In this work, additive normalization is always used implicitly for all weighted networks. Multiplicative normalization is explicitly used by using the normalized matrix $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ instead of $\mathbf{A}$.

7.3 Evaluating link prediction methods

To evaluate a link prediction algorithm, we must know which links actually appeared in a network and compare them to the links that were predicted. In networks where we know the time at which each edge has appeared, we can split the set of edges by creation time. We can then apply a given link prediction algorithm to the training set, use it to predict links, and compare the predicted links with the test set.

Each of the link prediction algorithms that we introduce in this work is based on a spectral transformation. This spectral transformation is specific to each dataset, and therefore, it has to be learned separately for each dataset. Thus, we split the training set again by edge creation time and use this split to learn a spectral transformation, which is then applied to the whole test set. Finally, the predicted edges are compared with the edges in the test set. Figure 9 gives a summary of the method we use.

The split is performed in the following way. First, the training/test split is made, using 75 % of all edges in the training set and 25 % of all edges in the test set. This split is done by edge creation times: Every edge in the training set is older than every edge in the test set. At the time where the split occurs, the network contains all edges in the training set, and nothing more. The training set is then split into a source and target set, also of sizes 75 and 25 %.

Then, the giant connected component is found in the source set, and all vertices not in this giant connected component are removed in the source, target, and test sets. The rationale here is the following: If an edge in the test or target set connects two vertices that are not connected by a path in the source set, then there is no point in trying to learn this edge. By only keeping the giant connected component in the source set, we make sure that edges in the test and target sets are indirectly connected in the source set. This pruning, however, results in splits that are not exactly 75 %/25 % for all datasets. If a dataset does not have a connected component in the source set of at least 10 % of all edges, then we did not include that dataset in our collection.

We use the following notation to denote the source, target, and test sets. In the graph $G = (V, E)$, the set of edges is partitioned as $E = E_a \dot{\cup} E_b \dot{\cup} E_c$, where E_a is the source set, E_b is the target set, and E_c is the test set. The training set is $E_a \dot{\cup} E_b$. The adjacency matrix $\mathbf{A}$ is then decomposed as $\mathbf{A} = \mathbf{A}_a + \mathbf{A}_b + \mathbf{A}_c$, where the indices a, b, c represent the source, target, and test set, respectively.

7.4 Measuring link prediction accuracy

To compare the experimental results of link prediction algorithms, a measure of link prediction accuracy has to be used. Several such measures are used in the literature, of which we only mention the mean average precision (MAP) [49], the area under the curve (AUC) [4], and the normalized discounted cumulated gain (NDCG) [22]. These measures of accuracy are similar to each other, and our tests indicate that they are interchangeable for our purposes. Therefore, we will only report prediction accuracy using a single one of these measures, the mean average precision, which we now define.

The mean average precision is based on the ranking implied by link prediction scores. In other words, giving predicted scores for all edges, we find the actual edges among them and then their ranking determines a mean average precision value. Since the total number of possible edges is quadratic in the vertex count, this cannot be computed in practice. Instead, we generate a *zero test set*, that is, a set of edges of the same size as the test, containing only edges not contained in the original networks. We call this zero test set $E_{\bar{c}}$.

Let $E_{c\bar{c}} = E_c \dot{\cup} E_{\bar{c}}$ be the total test set of edges with $|E_{c\bar{c}}| = r$. For each test edge $e \in E_{c\bar{c}}$, $\mathbf{A}_e$ is the actual edge weight in the test set, and p_e the computed link prediction value. The average precision is then defined in terms of rankings: Edges are ranked by their predicted weight and evaluated by their actual weight. For each edge in E_c, we compute the precision of the results up to that edge. The average of these precision values then gives the average precision. The average precision is therefore in the range [0, 1], and the value 1 denotes perfect prediction.

Let $R(i)$ be the edge of rank i defined such that $p_{R(i)} \leq p_{R(j)}$ if $i < j$. Then the average precision of the prediction vector p is defined as

$$\operatorname{ap}(p) = \left(\sum_{i=1}^{r} \left[\mathbf{A}_{R(i)} \right]_0^1 \right)^{-1} \sum_{i=1}^{r} \left(\left[\mathbf{A}_{R(i)} \right]_0^1 \frac{1}{i} \sum_{j=1}^{i} \left[\mathbf{A}_{R(j)} \right]_0^1 \right), \tag{33}$$

where $[x]_0^1 = 1$ if $x > 1/2$ and $[x]_0^1 = 0$ otherwise.

The mean average precision is then given by the mean of average precision values computed over each node separately [44]. Let $\operatorname{ap}_i(p)$ be the average precision computed only over all edges adjacent to vertex i. Then the mean average precision is given by:

$$\operatorname{map}(p) = \frac{1}{|V|} \sum_{i \in V} \operatorname{ap}_i(p) \tag{34}$$

In information retrieval, the average precision is averaged over all queries. Here, we consider each vertex in the test set a query and consider the predict of its incident edges a query. Both definitions are equivalent.

8 Learning by curve fitting

We now describe a general learning technique for estimating the parameters of any graph kernel or other link prediction function that can be expressed as a spectral transformation. The method described in this section was originally published in [29]. This method works by reducing the learning problem to a one-dimensional curve-fitting problem. As mentioned in Sect. 5, a certain number of graph kernels and other link prediction methods can be expressed as spectral transformations, confirming the spectral evolution model. Our contribution in this section is a method for learning the parameters of these link prediction functions by reducing the resulting high-dimensional parameter learning problem to a one-dimensional curve-fitting problem, which can be solved efficiently. The runtime of this method only depends on the chosen reduced rank and is independent of the original network size.

We show that this generalization is possible under the assumption that the chosen training and test sets are simultaneously diagonalizable, an assumption which, as described in Sect. 5, is implicit in the spectral evolution model. The method presented in this section can be used to learn the parameters of these kinds of link prediction functions. This includes not only the parameters of graph kernels such as the matrix exponential, but also the reduced rank in rank reduction methods and weights used for individual path lengths in path-counting methods. Since we reduce the parameter estimation problem to a one-dimensional curve-fitting problem that can be plotted and inspected visually to compare the different prediction algorithms, an informed choice can be made about them, in addition to measuring the precision of each algorithm on a test set.

Let $\mathbf{A}$ be the adjacency matrix of an undirected, unipartite graph, and $\mathbf{A} = \mathbf{U\Lambda U}^{\mathrm{T}}$ its eigenvalue decomposition. Then the spectral evolution model states that new links can be predicted by computing a spectral transformation $F(\mathbf{A}) = \mathbf{U}F(\mathbf{\Lambda})\mathbf{U}^{\mathrm{T}}$. As described in Sect. 5, several link prediction functions are of this form, in particular all graph kernels based on the adjacency matrix such as the matrix exponential $F(\mathbf{A}) = \exp(\alpha\mathbf{A})$. Most of these graph kernels have parameters, for instance α for the matrix exponential. The method in this section constitutes a way to learn these parameters, and at the same time to compare different graph kernels in more depth than simply comparing their performances at link prediction.

Other graph kernels than those presented in the last section are not spectral transformations of the adjacency matrix $\mathbf{A}$, but of the Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{A}$, or the normalized adjacency matrix $\mathbf{N} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. The method described in this section can also be applied to these graph kernels, as we will show.

8.1 Link prediction as an optimization problem

Given a graph $G = (V, E)$, we want to find a spectral transformation F that performs well at link prediction for this particular graph. To that end, we divide the set of edges E into a training set E_a and a test set E_b and then look for a function F that maps the training set to the test set with minimal error.

Formally, let $\mathbf{A}_a$ and $\mathbf{A}_b$ be the adjacency matrices of the source and target set, respectively, such that $\mathbf{A}_a + \mathbf{A}_b$ is the complete adjacency matrix of G. In other words, we know $\mathbf{A}_a$ and want to find a function that maps $\mathbf{A}_a$ to $\mathbf{A}_b$. We will call $\mathbf{A}_a$ the source matrix and $\mathbf{A}_b$ the target matrix. Let $\mathcal{S}$ be the set of all spectral transformations from $\mathbb{R}^{|V|\times|V|}$ to $\mathbb{R}^{|V|\times|V|}$. The solution to the following optimization problem gives the optimal spectral transformation for the task of predicting the edges in the test set.

Problem 1 Let $\mathbf{A}_a, \mathbf{A}_b \in \mathbb{R}^{|V|\times|V|}$ be two symmetric adjacency matrices over the same vertex set V. A spectral transformation that maps $\mathbf{A}_a$ to $\mathbf{A}_b$ with minimal error is given by a solution to

$$\begin{aligned} \min_F \quad & \|F(\mathbf{A}_a) - \mathbf{A}_b\|_{\mathrm{F}} \\ \text{s.t.} \quad & F \in \mathcal{S} \end{aligned} \tag{35}$$

The Frobenius norm $\|\cdot\|_{\mathrm{F}}$ corresponds to the root mean squared error (RMSE) of the mapping from $\mathbf{A}_a$ to $\mathbf{A}_b$. While the root mean squared error is common in link prediction problems, other error measures exist, but give more complex solutions to our problem, making it impossible to solve the resulting problem efficiently. We will therefore restrict ourselves to the Frobenius norm in the rest of the section.

Problem 1 can be solved by computing the eigenvalue decomposition $\mathbf{A}_a = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}$ and using the fact that the Frobenius norm is invariant under multiplication by an orthogonal matrix.

$$\begin{aligned} &\left\| F(\mathbf{A}_a) - \mathbf{A}_b \right\|_{\mathrm{F}} \\ &= \left\| \mathbf{U} F(\mathbf{\Lambda}) \mathbf{U}^{\mathrm{T}} - \mathbf{A}_b \right\|_{\mathrm{F}} \\ &= \left\| F(\mathbf{\Lambda}) - \mathbf{U}^{\mathrm{T}} \mathbf{A}_b \mathbf{U} \right\|_{\mathrm{F}} \end{aligned} \tag{36}$$

The Frobenius norm in Expression (36) can be decomposed into the sum of squares of off-diagonal entries of $F(\mathbf{\Lambda}) - \mathbf{U}^{\mathrm{T}}\mathbf{A}_b\mathbf{U}$, which is independent of F, and into the sum of squares of its diagonal entries. This leads to the following least-squares problem equivalent to Problem 1:

Problem 2 If $\mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}$ is the eigenvalue decomposition of $\mathbf{A}_a$, then the solution to Problem 1 is given by $F(\mathbf{\Lambda})_{ii} = f(\mathbf{\Lambda}_{ii})$, where $f(\lambda)$ is a solution to the following minimization problem.

$$\min_f \quad \sum_i \left(f(\mathbf{\Lambda}_{ii}) - \mathbf{U}_{\cdot i}^{\mathrm{T}} \mathbf{A}_b \mathbf{U}_{\cdot i} \right)^2 \tag{37}$$

This is a one-dimensional least-squares curve-fitting problem of size n. Since each function $F(\mathbf{A}_a)$ corresponds to a function $f(\lambda)$, we can choose a link prediction function F and learn its parameters by inspecting the corresponding curve fitting problem.

8.2 Curve fitting

We have reduced the general matrix regression problem of Eq. (35) to a one-dimensional least-squares curve-fitting problem of size n. In practice, computing the full eigenvalue decomposition $\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}$ is not possible, first because the decomposition has runtime $O(n^3)$, but also because the dense $n \times n$ matrix $\mathbf{U}$ would take too much memory to store, that is, $O(n^2)$. Instead, the rank-reduced eigenvalue decomposition $\mathbf{A} \simeq \hat{\mathbf{U}}\hat{\mathbf{\Lambda}}\hat{\mathbf{U}}^{\mathrm{T}}$ of rank k can be computed. The result is that in Eq. (37), the curve-fitting problem goes only over k points. Since k is usually much smaller than n, this curve-fitting problem is computationally very easy to solve.

We now take each spectral link prediction function $F(\mathbf{A})$ described in Sect. 5 and, for each one, derive its underlying real function $f(\lambda)$ that applies to every eigenvalue separately. Most of the link prediction functions have parameters, for example, the parameter α in the exponential kernel $\exp(\alpha\mathbf{A})$. These parameters are kept in the real function. Additionally, we insert a multiplicative parameter $\beta > 0$ into every real function $f(\lambda)$, giving $\beta f(\lambda)$, that is to be learned along with the other parameters. Although this parameter will not change the relative link prediction scores and therefore has no influence on the final link prediction accuracy, including it allows the learned curves to better fit the observed eigenvalues and thus results in more sensible values for the other parameters. The resulting functions are summarized in Table 1.

Since both $\mathbf{\Lambda}$ and $\mathbf{U}^{\mathrm{T}}\mathbf{A}_b\mathbf{U}$ are available after having computed the eigenvalue decomposition of $\mathbf{A}_a$, the main computational part of our method lies in the rank-reduced eigenvalue decomposition of $\mathbf{A}_a$. The computation of $\mathbf{U}^{\mathrm{T}}\mathbf{A}_b\mathbf{U}$ has runtime complexity $O(kr)$ where r is the number of nonzeroes in $\mathbf{A}_b$ and the curve-fitting runtime is only dependent on k. By rank reduction, the final least-squares problem has only size k. Since the rank-reduced

Table 1 Link prediction functions and their corresponding real functions

	Method	Matrix function	Real function
Section 5.1	Triangle closing	$\mathbf{A}^2$	$f(\lambda) = \lambda^2$
Section 5.2	Path counting	$p(\mathbf{A}) = \sum_{k=0}^{d} \alpha_k \mathbf{A}^k$	$f(\lambda) = \sum_{k=0}^{d} \alpha_k \lambda^k$
Section 5.3	Exponential kernel	$K_{\mathrm{EXP}}(\mathbf{A}) = \exp(\alpha \mathbf{A})$	$f(\lambda) = e^{\alpha\lambda}$
	Neumann kernel	$K_{\mathrm{NEU}}(\mathbf{A}) = (\mathbf{I} - \alpha \mathbf{A})^{-1}$	$f(\lambda) = 1/(1 - \alpha\lambda)$
Section 5.4	Rank reduction	$R_k(\mathbf{A}) = \hat{\mathbf{U}}\hat{\mathbf{\Lambda}}\hat{\mathbf{U}}^{\mathrm{T}}$	$f(\lambda) = \begin{cases} \lambda & \text{if } \lvert\lambda\rvert \geq \lvert\boldsymbol{\lambda}_k\rvert \\ 0 & \text{otherwise} \end{cases}$

The outer multiplicative coefficient β (see explanation in the text) is not shown, for clarity

eigenvalue decomposition of $\mathbf{A}_a$ is computed anyway for spectral link prediction methods and because $r \ll n^2$ since $\mathbf{A}_b$ is sparse, it follows that this curve fitting method has only negligible overhead.

A related idea is to use $\mathbf{A}_a + \mathbf{A}_b$ instead of $\mathbf{A}_b$ for optimization, in the assumption that a link prediction algorithm should return high values for known edges. With this modification, we compute $\mathbf{U}^{\mathrm{T}}(\mathbf{A}_a + \mathbf{A}_b)\mathbf{U}$ instead of $\mathbf{U}^{\mathrm{T}}\mathbf{A}_b\mathbf{U}$. Our test showed that this variant gave worse results in almost all cases. We therefore do not consider this idea any further.

Finally, we must consider the problem of eigenvalue scaling. Since we only train F on a source matrix $\mathbf{A}_a$ but intend to apply the learned function to $\mathbf{A}_a + \mathbf{A}_b$, the function F would be applied to spectra of different extent. Therefore, we normalize the spectra we encounter by dividing all eigenvalues by the largest absolute eigenvalue, replacing $\mathbf{\Lambda}$ by $|\lambda_1|^{-1}\mathbf{\Lambda}$. Note that dividing the eigenvalue by $|\lambda_1|$ is dependent on the factor β, since the β is applied after the spectral transformation.

8.3 Laplacian kernels

The method of learning a link prediction kernel by curve fitting also applies to the Laplacian matrix. Instead of using the adjacency matrix $\mathbf{A}_a$, we use the Laplacian $\mathbf{L}$ as the source matrix. Here, the Laplacian $\mathbf{L}$ is understood to be computed over the source edge set E_a.

The Laplacian matrix of an undirected, unipartite graph with n vertices is an $n \times n$ matrix that characterizes the connections between vertices in the graph. $\mathbf{L} = \mathbf{D} - \mathbf{A}$ is the combinatorial Laplacian of the graph, and $\mathcal{L} = \mathbf{I} - \mathbf{N} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$ is the normalized Laplacian. The Laplacian matrices are symmetric and positive-semidefinite. When the network is unweighted, the multiplicity of $\mathbf{L}$ and $\mathcal{L}$ equals the number of connected components in the network. The Laplacian matrices are thus singular for nonempty networks. The eigenvalues of the normalized Laplacian matrix $\mathcal{L}$ lie in the interval [0, 2].

Using the eigenvalue decomposition of the Laplacian, several Laplacian graph kernels can be computed. These Laplacian graph kernels are known in the literature and have also be used for link prediction. These graph kernels are all spectral transformations of the Laplacian, and therefore, they can be learned using curve fitting.

Commute-time Kernels Since the Laplacian matrix is positive-semidefinite, its Moore–Penrose pseudoinverse is also positive-semidefinite and can be used as a graph kernel [14].

The Moore–Penrose pseudoinverse of a matrix is a generalization of the matrix inverse. For a square, symmetric matrix $\mathbf{X}$, the Moore–Penrose pseudoinverse is denoted $\mathbf{X}^+$. Given the eigenvalue decomposition $\mathbf{X} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}$, the Moore–Penrose pseudoinverse can be computed as $\mathbf{X}^+ = \mathbf{U}\mathbf{\Lambda}^+\mathbf{U}^{\mathrm{T}}$, where the Moore–Penrose pseudoinverse of the diagonal $\mathbf{\Lambda}$ is given by inverting each nonzero element of $\mathbf{\Lambda}$ separately, that is, $(\mathbf{\Lambda}^+)_{ii} = \mathbf{\Lambda}_{ii}^{-1}$ if $\mathbf{\Lambda}_{ii} \neq 0$ and 0 otherwise. Note that the Moore–Penrose pseudoinverse is a spectral transformation, called the commute-time kernel.

$$K_{\mathrm{COM}}(\mathbf{L}) = \mathbf{L}^+ \tag{38}$$

$$K_{\mathrm{COM}}(\mathcal{L}) = \mathcal{L}^+ \tag{39}$$

The pseudoinverse $\mathbf{L}^+$ can be used to derive a metric:

$$D(i, j) = (\mathbf{L}^+)_{ii} + (\mathbf{L}^+)_{jj} - (\mathbf{L}^+)_{ij} - (\mathbf{L}^+)_{ji} \tag{40}$$

This metric has two interpretations. If the network is interpreted as an electrical network where each edge is a resistor, this metric gives the equivalent resistance between any two nodes [25]. The kernel $\mathbf{L}^+$ is thus known as the resistance distance kernel and $\mathbf{L}$ as the Kirchhoff matrix. When considering a random walk on the graph G, the average commute time between any two nodes equals the resistance distance. The equivalence between the two formalisms is studied in [13].

By regularization of the commute-time kernel, we arrive at the regularized Laplacian kernels [54]:

$$K_{\mathrm{COMR}}(\mathbf{L}) = (\mathbf{I} + \alpha\mathbf{L})^{-1} \tag{41}$$

$$K_{\mathrm{COMR}}(\mathcal{L}) = (\mathbf{I} + \alpha\mathcal{L})^{-1} \tag{42}$$

For the parameter α, we require $\alpha^{-1} > \lambda_1$, where λ_1 is the largest eigenvalue of $\mathbf{L}$, or $0 < \alpha < 1$ in the normalized case. As a special case, the nonnormalized regularized Laplacian kernel is called the random forest kernel for $\alpha = 1$ [9] and characterizes the proximity of two vertices by the number of ways they can be connected by random forests. The normalized regularized Laplacian is equivalent to the normalized Neumann kernel by noting that $(1 + \alpha\mathcal{L})^{-1} = (1 + \alpha)(\mathbf{I} - \alpha\mathbf{N})^{-1}$.

Heat Diffusion By considering a process of heat diffusion on the network, a Laplacian heat diffusion kernel can be derived [21]:

$$K_{\mathrm{HEAT}}(\mathbf{L}) = \exp(-\alpha\mathbf{L}) \tag{43}$$

$$K_{\mathrm{HEAT}}(\mathcal{L}) = \exp(-\alpha\mathcal{L}) \tag{44}$$

The normalized heat diffusion kernel is equivalent to the normalized exponential kernel: $\exp(-\alpha\mathcal{L}) = e^{-\alpha}\exp(\alpha\mathbf{N})$ [54]. The heat diffusion kernel is also known as the Laplacian exponential diffusion kernel [15].

The normalized Laplacian can be derived from the normalized adjacency matrix by the spectral transformation $\mathcal{L} = F(\mathbf{N}) = \mathbf{I} - \mathbf{N}$ corresponding to the real function $f(\lambda) = 1 - \lambda$. Thus, the normalized commute-time kernel reduces to the normalized Neumann kernel, and the heat diffusion kernel reduces to the normalized exponential kernel.

Table 2 shows the underlying real functions for Laplacian graph kernels. A larger set of Laplacian kernels is described in [15], most of which contain additional parameters. Note that the Laplacian kernels apply decreasing functions to the Laplacian spectrum. In the resulting kernels, the dominant eigenvectors are thus the eigenvectors of the smallest nonzero eigenvalue of $\mathbf{L}$ or $\mathcal{L}$. As we compute only k eigenvectors and eigenvalues of the Laplacian matrices, we have to make sure that we find the smallest k eigenvalues and their eigenvectors.

Table 2 Laplacian link prediction functions and their corresponding real functions

Method	Matrix function	Real function
Commute-time kernel	$K_{\mathrm{COM}}(\mathbf{L}) = \mathbf{L}^+$	$f(\lambda) = \begin{cases} \lambda^{-1} & \text{if } \lambda > 0 \\ 0 & \text{otherwise} \end{cases}$
Regularized commute-time kernel	$K_{\mathrm{COMR}}(\mathbf{L}) = (\mathbf{I} + \alpha\mathbf{L})^{-1}$	$f(\lambda) = 1/(1 + \alpha\lambda)$
Heat diffusion	$K_{\mathrm{HEAT}}(\mathbf{L}) = \exp(-\alpha\mathbf{L})$	$f(\lambda) = e^{-\alpha\lambda}$

9 Learning by spectral extrapolation

We now derive another method for link prediction based on the spectral evolution model. According to the spectral evolution model established in the previous section, the spectrum of a network evolves over time, while the eigenvectors remain constant. We saw that this assumption is true to a certain point in large real-world networks and used it in the last section to reduce the link prediction problem to a simple curve-fitting problem. To do this, we chose a time in the past and learned a mapping from the network's structure in the past to the network's structure in the present. Instead, we will now consider infinitesimal changes in the present network and extrapolate them into the future. The method described in this section was originally published in [28].

According to the spectral evolution model, only the eigenvalues of a network change significantly over time. Therefore, we will look at the change in eigenvalues of a network and extrapolate it into the future. Because we consider each eigenvalue separately, this method is suitable for cases where eigenvalues of a single network grow at very different speeds. In such a case, graph kernels as learned in the previous section are not accurate, since they assume all eigenvalues to grow according to a single real function, the spectral transformation function. The method presented here can therefore be considered as an extension of graph kernels to irregular grow patterns, while still being conformant to the spectral evolution model.

Link prediction functions such as the matrix exponential assume there is a real function $f(\lambda)$ that applies to all eigenvalues. In the link prediction functions described in Sect. 5, this function is very regular, that is, it is positive and grows monotonically. Observed spectral transformations such as the one in Fig. 10b are, however, irregular, and no simple function solves the curve-fitting problem well. This is explained by the fact that eigenvalues may cross each other, indicating that a nonmonotonous function is needed. Note that the irregularity here is only observed in the growth of eigenvalues. The overall growth is still spectral, and eigenvectors are nearly constant.

For these reasons, we must consider the growth of each latent dimension separately. If we knew that eigenvalues did not cross each other, we could compare the ith eigenvalue at two points in time and extrapolate a third value. But because eigenvalues pass each other, we must first learn the relation between new and old eigenvalues.

9.1 Method

Given a network dataset, we sort all its edges by edge arrival time. Then, we split the set of edges into three sets of edges: the source set of edges, the target set of edges, and the test set of edges. This split is visualized in Fig. 9. The source and target sets taken together are called the training set. The goal of the evaluation will then be to learn a spectral transformation

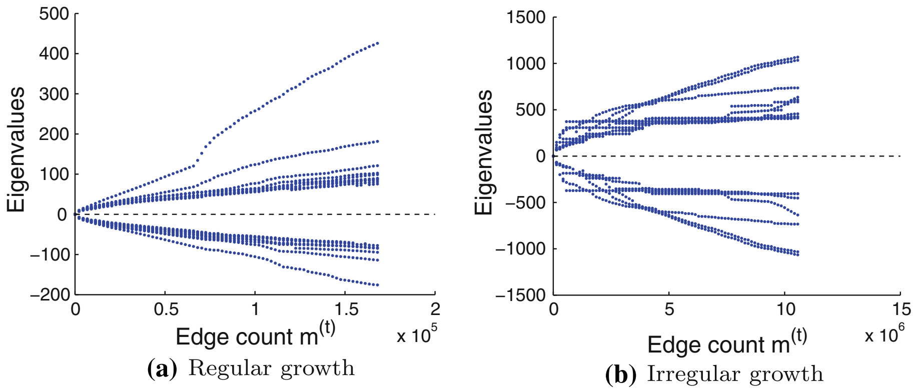


Fig. 10 Examples of **a** regular and **b** irregular spectral growth. In the first case, the eigenvalues grow regularly, and a graph kernel can be used to model network evolution. In the second case, spectral growth is irregular and graph kernels are not suitable. In both cases, growth is spectral. **a** Internet topology (TO), **b** Twitter user–user @ mentions (Wa)

set going from the source to the target set, to apply that learned spectral transformation to the training set, and to compare the predicted edges with the edges in the test set. The two timepoints at which the edge set is split are defined as follows: t_a is the time of split between the source and target sets, and t_b is the time of split between the training and the test set.

Let $\mathbf{A}_a$ be the adjacency matrix containing all edges in the source set and $\mathbf{A}_b$ the adjacency matrix containing all edges in the target set. The adjacency matrix of the training set is then $\mathbf{A}_a + \mathbf{A}_b$. We now compute the eigenvalue decompositions of the adjacency matrices at times t_a and t_b:

$$\mathbf{A}_a = \mathbf{U}_a \mathbf{\Lambda}_a \mathbf{U}_a^{\mathrm{T}} \tag{45}$$
$$\mathbf{A}_a + \mathbf{A}_b = \mathbf{U}_b \mathbf{\Lambda}_b \mathbf{U}_b^{\mathrm{T}} \tag{46}$$

To find out which latent dimension i at time t_a corresponds to the latent dimension j at time t_b, we compute a mean of eigenvalues at time t_a, weighted by the similarity between the two latent dimensions at time t_a and at time t_b. As a similarity measure, we use the dot product between the corresponding normalized eigenvectors. Thus, if $(\lambda_b)_j$ is an eigenvalue of $\mathbf{A}_a + \mathbf{A}_b$ at time t_b, its conjectured previous value at time t_a is as follows:

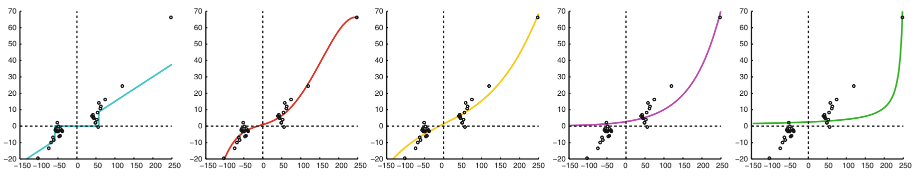
$$(\hat{\lambda}_a)_j = \left(\sum_i (\mathbf{U}_a)_{\cdot i}^{\mathrm{T}} (\mathbf{U}_a)_{\cdot j} \right)^{-1} \sum_i (\mathbf{U}_a)_{\cdot i}^{\mathrm{T}} (\mathbf{U}_a)_{\cdot j} (\lambda_a)_i \tag{47}$$

where $(\lambda_a)_i$ and $(\mathbf{U}_a)_{\cdot i}$ are the eigenvalues and eigenvectors at time t_a, respectively. We then perform linear extrapolation to predict an eigenvalue $\hat{\lambda}_j$ in the future.

$$\hat{\lambda}_j = 2(\lambda_b)_j - (\hat{\lambda}_a)_j \tag{48}$$

Table 3 Comparison of the two learning methods

Curve fitting	Extrapolation
Uses any characteristic matrix	Uses only the adjacency matrix
Considers only two timepoints	Considers all timepoints
Works well when growth is regular	Works well when growth is irregular



(a) Rank reduction **(b)** Polynomial **(c)** Nonnegative polynomial **(d)** Exponential **(e)** Neumann Kernel

Fig. 11 Curve-fitting individual graph kernels, using the Internet topology network (TO)

Using the matrix $\hat{\mathbf{\Lambda}} = (\hat{\lambda}_j)$, the predicted edge weights are then $\mathbf{U}_b \hat{\Lambda} \mathbf{U}_b^{\mathrm{T}}$. The computed eigenvalues $\hat{\lambda}_j$ in the matrix $\hat{\mathbf{\Lambda}}$ are not necessarily ordered, which is not a problem for link prediction scores.

10 Experiments

We will now compare both new link prediction methods experimentally with each other, and against baseline algorithms. First, we recall the basic differences in the two methods in Table 3. The list of datasets is given in Table 5 in the "Appendix." All network datasets are part of the Koblenz Network Collection (KONECT, konect.uni-koblenz.de). In the following experiments, all matrix decompositions are computed up to the rank given in Table 5 in the "Appendix".

To evaluate the performance of the two link predictions methods, we use the unipartite, unweighted networks of Table 5 and compute the link prediction accuracy using the mean average precision as defined in Sect. 7.4.

We apply the curve-fitting methods and spectral extrapolation methods described in the previous sections to the source and target edge sets of each network, learning several link prediction functions. We then apply these link prediction functions to the training set of each network to compute predictions for the edges in the test set. Since the networks in these experiments are unweighted, the test set contains edges and nonedges, representing the zero test set as described in Sect. 7.3. As a baseline, we use the local link prediction functions described in Sect. 7.1.

The results of our evaluation are shown in Table 4. Figure 11 shows the curve-fitting method applied to individual combinations of base matrices and graph kernels for the Internet Topology network (TO).

Figure 12 shows the extrapolation method applied to three selected networks. The numerical results are in Fig. 13.

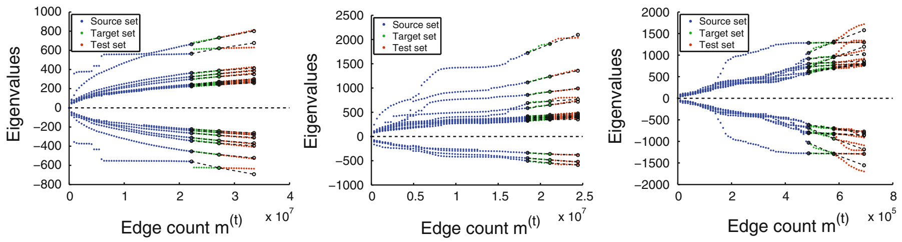
Observations We make the following observations:

– The overall best performing graph kernel based on the adjacency matrix **A** is the exponential kernel $\exp(\alpha \mathbf{A})$. The other graph kernels perform worse, but still have acceptable

Table 4 The best performing spectral transformation for each dataset

Dataset	Best method	MAP
arXiv hep-ph (PH)	Spectral extrapolation	0.758
arXiv hep-th (TH)	Spectral extrapolation	0.756
Haggle (CO)	$(\mathbf{I} - \alpha\mathbf{A})^{-1}$	0.738
DBLP (Pc)	$\exp(\alpha\mathbf{A})$	0.759
English Wikipedia (EL)	$\exp(\alpha\mathbf{A})$	0.828
Enron (EN)	Spectral extrapolation	0.876
Epinions (EP)	Spectral extrapolation	0.310
Facebook New Orleans (Ol)	Spectral extrapolation	0.669
Facebook New Orleans (Ow)	$p(\mathbf{A})$	0.802
Flickr (FL)	$p(\mathbf{A})$	0.655
Digg (DG)	$R_k(\mathbf{A})$	0.871
Twitter (Wa)	$\exp(\alpha\mathbf{A})$	0.771
Internet topology (TO)	$\exp(\alpha\mathbf{A})$	0.726
English Wikipedia (WP)	$p(\mathbf{N})$	0.666
YouTube (YT)	$\exp(\alpha\mathbf{A})$	0.687

We show the link prediction method that performs best, and its mean average precision (MAP)



(a) Wikipedia hyperlinks (WP) **(b)** Flickr friends (FL) **(c)** Facebook wall posts (Ow)

Fig. 12 The result of applying spectral extrapolation to the Wikipedia, Flickr, and Facebook networks. The actual evolution of the spectrum of each network is shown in dots. Spectral extrapolation is computed using only the decompositions at the split points between the source, target, and test sets and is shown as dotted lines overlaid on the target and test set

prediction accuracy. We interpret the better performance of the matrix exponential by the fact that the coefficients in its Taylor expansion get small much faster than the coefficients in the Taylor expansions of the other graph kernels, meaning that long paths are much less important than short paths.

– Spectral transformations of the Laplacian $\mathbf{L}$ have significantly worse link prediction accuracy than functions of $\mathbf{A}$. This result is consistent with the observed absence of Laplacian spectral evolution, and also with the result from Radl et al. [52] that the resistance distance (which is based on the commute-time kernel $\mathbf{L}^{+}$) is meaningless for large random geometric graphs. Among functions of $\mathbf{L}$, the regularized commute-time kernel $(\mathbf{I} + \alpha\mathbf{L})^{-1}$ performs best, but not significantly.
– The spectral extrapolation method performs better than the curve-fitting methods only in a few cases. The spectral extrapolation method suffers from the case where eigenvalues are

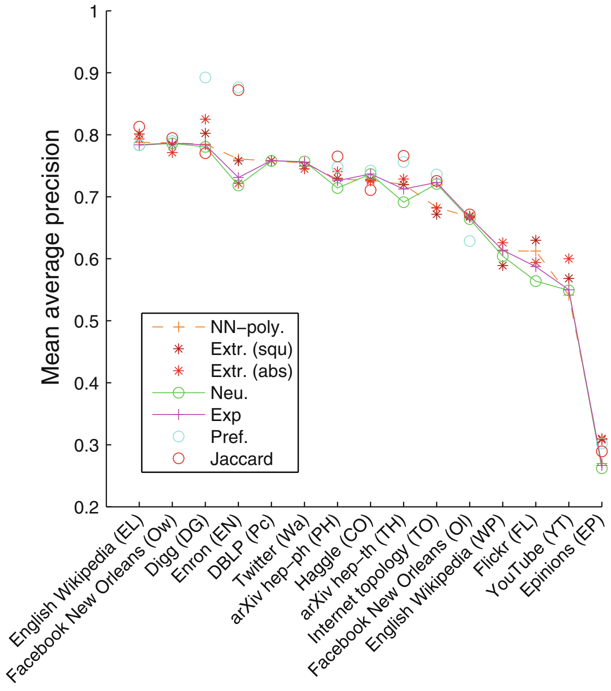


Fig. 13 Evaluation results for unweighted, unipartite networks. Directed networks are included, but edge directions are ignored in them

very near each other. In practice, this is not common, but can be observed in very decentral networks, that is, networks without a clear core component, such as road networks.

– Among local link prediction functions, the Jaccard coefficient and the Adamic–Adar measure correlate to a high degree. They both have a prediction accuracy comparable to that of preferential attachment. All local link prediction methods perform significantly worse than the best curve-fitting methods.
– We also found that if a dataset is so big that only few eigenvectors and eigenvalues can be computed, the resulting points are not numerous enough to make a meaningful distinction between the performance of the different curve-fitting models. This can be interpreted as a case of overfitting: Using only two or three eigenpairs, a linear function of the spectrum is probably the best prediction we can make. By comparing the results with various types of network datasets, we found that the various link prediction methods in current use are justified in specific cases.

A general note should be made about the overall performance of all algorithm for specific networks. For the Epinions trust network (EP) for instance, and other network datasets not shown in this evaluation, the mean average precision value is small compared to those of other datasets. This is explained in most cases by the fact that these networks have low clustering, that is, a comparatively low amount of triangles.

As an overall conclusion, we recommend using curve fitting to learn the parameters of the matrix exponential as a candidate for link prediction applications. However, we recommend to generate the curve fitting plot described in Sect. 8 in any case to catch eventual new spectral transformations patterns. Also, we recommend to perform the spectral diagonality test described in Sect. 4.4 beforehand, to catch those datasets where spectral transformations do not work.

11 Related work

Several previous attempts have been made at learning spectral transformations. These attempts have been restricted to specific datasets and problem types and did not consider the validity of the spectral evolution model.

Joint Diagonalization Simultaneously diagonalizable matrices have been considered in the context of *joint diagonalization*, where given a set of matrices $\{\mathbf{A}_i\}$, a matrix $\mathbf{U}$ is searched that makes the matrices $\{\mathbf{U}\mathbf{A}_i\mathbf{U}^{\mathrm{T}}\}$ as diagonal as possible according to a given matrix norm. See for instance [59]. In that context, the task consists of finding an orthogonal matrix $\mathbf{U}$, whereas our approach consists of optimizing the fit by varying the spectrum.

Joint diagonalization methods have been used for link prediction [56]. By construction, the resulting network evolution is spectral, and the approximation at each timepoint is less precise than the eigenvalue decomposition as described in this work. Equivalently, this can be described as a decomposition of the vertex-vertex-time tensor [51]. In fact, our spectral evolution model provides a justification for these advanced methods, since joint diagonalization implies constant or near-constant eigenvectors.

Linear and Quadratic Programming In [65], spectral transformations of the Laplacian matrix are considered, where they are called *spectral transforms*. The spectral transformations in that work are learned using quadratically constrained quadratic programming (QCQP). A similar technique is used in [42]. In [40], a spectral transformation of the Laplacian matrix is learned using linear programming.

12 Discussion

The spectral evolution model states that in many real-world networks, growth can be described by a change of the spectrum, while the corresponding eigenvectors remain constant. This assumption is true to a large extent in most networks we analyzed, based on the observation of the change in eigenvalues and eigenvectors of networks over time. We also described a more sophisticated test of the spectral evolution model, the spectral diagonality test. This test measures to what extent a snapshot of the network at one point can be diagonalized by the eigenvectors of the same network at another time. The result is a matrix that is diagonal exactly when growth is perfectly spectral. When growth is not spectral, this matrix is not diagonal. The degree to which this matrix is spectral can therefore be used as an indicator of the spectrality of a network's growth.

The fact that the spectral evolution model can be observed in many networks is not in contradiction to previous results in the literature. In fact, a certain number of known link prediction methods described previously result in network growth that can be described by spectral transformations. This class of link prediction functions includes the triangle closing model, the weighted path-counting model, all graph kernels based on adjacency matrix, and rank reduction approaches. Another explanation of the spectral evolution model is given by an extension to the preferential attachment model, which we call the latent preferential attachment model. As we showed, the latent preferential attachment model is equivalent to the spectral evolution model, under the assumption that a preferential attachment process happens in each latent dimension separately.

To make sure that spectral evolution is a characteristic of real networks and not an artifact of sampling any network randomly, we performed two control tests. By adding edges randomly to an existing network and by sampling a random network, we could not observe any spectral growth. This indicates that the spectral evolution model is indeed a nontrivial property of real-world networks.

Having confirmed that the spectral evolution model applies to many real networks, we could exploit it to predict links in these networks. We have presented two methods for learning link prediction functions in large networks. The first method uses spectral curve fitting and the second one spectral extrapolation. Both methods make different assumptions: The curve-fitting method learns a spectral mapping from one matrix to another and therefore can also be applied to matrices other than the adjacency matrix, such as the Laplacian matrix. The spectral extrapolation method on the other hand considers infinitesimal changes in the adjacency matrix's spectrum. It is therefore suited for network datasets where exact edge creation times are known. However, it cannot be applied to the Laplacian matrix or other characteristic graph matrices other than the adjacency matrix.

Both methods suffer from the case where eigenvalues are very near each other. In practice, this is not common, but can be observed in very decentral networks, that is, networks without a clear core component, such as road networks. For the curve-fitting method, we may guess that using the normalized adjacency matrix leads to similar problems, since the normalized adjacency matrix has many eigenvalues slightly less than one. However, this is not the case.

We also found that if a dataset is so big that only few eigenvectors and eigenvalues can be computed, the resulting points are not numerous enough to make a meaningful distinction between the performance of the different curve-fitting models. This can be interpreted as a case of overfitting: Using only two or three latent dimensions, a linear function of the spectrum is probably the best prediction we can make. By comparing the result with various types of network datasets, we found that the various link prediction methods in current use are justified in specific cases. We hope that the methods presented here can help make an informed choice as to the right method. The advantage of the curve-fitting method lies in the fact that we do not have to rely on the mean average precision of a link prediction function. Instead, we can look at its curve-fitting plot and come up with new spectral transformations, if necessary.

In the networks we tested, the spectral extrapolation method provides more accurate link prediction in many cases, in particular in networks with irregular—but still spectral—growth. For networks with very regular growth, regular graph kernels perform better, however. Across all datasets, the performance of our method is better than any single link prediction method. Our new extrapolation method is also parameter free: Not only are there no parameters, as in various graph kernels, but our methods make the choice of a specific spectral growth model unnecessary.

Finally, the fact that we observed avoided crossings of eigenvalues during network growth, even if just very slightly, may be an indication that other, nonorthogonal, matrix decompositions may also be of interest to study in the context of dynamic network analysis.

In ongoing work building on the results presented in this paper, we are investigating other matrix decompositions such as those that apply specifically to directed networks, to bipartite networks, and to 3-regular hypergraphs, for example, folksonomies. We are also investigating the eigenvectors more closely, first by observing in what way they change (even if not by

Table 5 The list of network datasets

	Dataset	Node and edge types	*Nodes*	*Edges*	*Rank*
CO	Haggle	Person–person contact	274	28,244	5
DG	Digg	User–user reply	30,398	87, 627	40
EL	English Wikipedia	Editor–editor vote	8,297	107,071	114
EN	Enron	User–user email	87,365	1,149,884	30
EP	Epinions	User–user trust/distrust	131,828	841,372	70
FL	Flickr	User–user friendship	2,302,925	33,140,018	30
Ol	Facebook	User–user friendship	63,731	1,545,686	70
Ow	Facebook	User–user wall post	63,891	876,993	49
Pc	DBLP	Author–author collaboration	801,474	9,510,516	30
PH	arXiv hep-ph	Author–author collaboration	28,093	12,730,098	31
TH	arXiv hep-th	Author–author collaboration	22,908	11,209,368	38
TO	Internet topology	AS–AS connection	34,761	171,403	34
Wa	Twitter	User–user "@" tag mention	2,919,613	12,887,063	30
WP	English Wikipedia	Article–article link	1,870,709	39,953,145	30
YT	YouTube	User–user friendship	3,223,643	18,524,095	30

For each network dataset, we show the rank used for computing the matrix decomposition

much) and then by interpreting their components as a distribution, resulting in a random graph model parametrized by the spectrum only.

Acknowledgments The research leading to these results has received funding from the European Community's Seventh Frame Programme under grant agreement no 257859, ROBUST, and from the German Research Foundation (DFG) under the Multipla project (grant 38457858).

Appendix: list of datasets

This is the complete list of network datasets; all are from the Koblenz Network Collection (KONECT, konect.uni-koblenz.de). The Haggle dataset (CO) represents contacts between persons [7]. The Digg datasets (DG) contain message replies between users of the Web site Digg [11]. The English Wikipedia vote datasets (EL) represent administrator votes between users of the English Wikipedia [36]. The Enron dataset (EN) is a network of e-mail messages between employees of Enron [26]. Epinions (EP) is a trust/distrust network from the Web site of the same name [45]. The Flickr network (FL) contains user–user friendship links from the Flickr image sharing Web site [47]. The two Facebook datasets (Ol, Ow) contain the friendship links and wall messages between users of the social network Facebook in New Orleans [58]. The DBLP network (Pc) contains the coauthorship links between authors in the DBLP bibliography [38]. The arXiv hep-ph and hep-th networks (PH, TH) also contain the coauthorship links between authors in the corresponding sections of arXiv [37]. The Internet topology dataset (TO) contains the structural network of the Internet [63]. The Twitter network (Wa) contains the user–user mentions using the "@" syntax in Twitter [10]. The English Wikipedia hyperlink network (WP) contains all links between articles of the English Wikipedia [46]. The YouTube dataset (YT) contains the user–user friendships from video-sharing site YouTube [46] (Table 5).

References

1. Adamic L, Adar E (2001) Friends and neighbors on the web. Soc Netw 25:211–230
2. Barabási A-L, Albert R (1999) Emergence of scaling in random networks. Science 286(5439):509–512
3. Blei D, Ng A, Jordan M, Lafferty J (2003) Latent dirichlet allocation. J Mach Learn Res 3:993–1022
4. Bradley AP (1997) The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recogn 30:1145–1159
5. Breese JS, Heckerman D, Kadie C (1998) Empirical analysis of predictive algorithms for collaborative filtering. In: Proceedings of conference on uncertainty in artificial intelligence, pp 43–52
6. Brin S, Page L (1998) The anatomy of a large-scale hypertextual Web search engine. Comput Netw ISDN Syst 30(1–7):107–117
7. Chaintreau A, Hui P, Crowcroft J, Diot C, Gass R, Scott J (2007) Impact of human mobility on opportunistic forwarding algorithms. IEEE Trans Mobile Comput 6(6):606–620
8. Chapelle O, Weston J, Schölkopf B (2003) Cluster kernels for semi-supervised learning. Adv Neural Inform Process Syst 15:585–592
9. Chebotarev P, Shamis E (1997) The matrix-forest theorem and measuring relations in small social groups. Autom Remote Control 58(9):1505–1514
10. Choudhury MD, Lin Y-R, Sundaram H, Candan KS, Xie L, Kelliher A (2010) How does the data sampling strategy impact the discovery of information diffusion in social media? In: Proceedings of conference on weblogs and social media, pp 34–41
11. Choudhury MD, Sundaram H, John A, Seligmann DD (2009) Social synchrony: predicting mimicry of user actions in online social media. In: Proceedings of the international conference on computational science and engineering, pp 151–158
12. Chung F (1997) Spectral graph theory. American Mathematical Society, Providence
13. Doyle PG, Snell JL (1984) Random walks and electric networks. Mathematical Association of America, Washington, DC
14. Fouss F, Pirotte A, Renders J-M, Saerens M (2004) A novel way of computing dissimilarities between nodes of a graph, with application to collaborative filtering and subspace projection of the graph nodes. In: Proceedings of European conference on machine learning, pp 26–37
15. Fouss F, Yen L, Pirotte A, Saerens M (2006) An experimental investigation of graph kernels on a collaborative recommendation task. In: Proceedings of the international conference on data mining, pp 863–868
16. Goldenberg A, Kubica J, Komarek P (2003) A comparison of statistical and machine learning algorithms on the task of link completion. In Proceedings workshop on link analysis for detecting complex behavior
17. Guha R, Kumar R, Raghavan P, Tomkins A (2004) Propagation of trust and distrust. In: Proceedings of the international world wide web conference, pp 403–412
18. Guimerà R, Sales-Pardo M, Amaral LAN (2004) Modularity from fluctuations in random graphs and complex networks. Phys Rev E 70(2):025101
19. Huang Z, Chung W, Chen H (2004) A graph model for e-commerce recommender systems. Am Soc Inf Sci Technol 55(3):259–274
20. Huang Z, Zeng D, Chen H (2007) A comparison of collaborative-filtering recommendation algorithms for e-commerce. IEEE Intell Syst 22(5):68–78
21. Ito T, Shimbo M, Kudo T, Matsumoto Y (2005) Application of kernels to link analysis. In: Proceedings of the international conference on knowledge discovery in data mining, pp 586–592
22. Järvelin K, Kekäläinen J (2002) Cumulated gain-based evaluation of IR techniques. ACM Trans Inf Syst 20(4):422–446
23. Kandola J, Shawe-Taylor J, Cristianini N (2002) Learning semantic similarity. In Advances in neural information processing systems, pp 657–664
24. Katz L (1953) A new status index derived from sociometric analysis. Psychometrika 18(1):39–43
25. Klein DJ, Randić M (1993) Resistance distance. J Math Chem 12(1):81–95
26. Klimt B, Yang Y (2004) The Enron corpus: A new dataset for email classification research. In: Proceedings of European conference on machine learning, pp 217–226
27. Kondor R, Lafferty J (2002) Diffusion kernels on graphs and other discrete structures. In: Proceedings of the international conference on machine learning, pp 315–322
28. Kunegis J, Fay D, Bauckhage C (2010) Network growth and the spectral evolution model. In: Proceedings of the international conference on information and knowledge management, pp 739–748
29. Kunegis J, Lommatzsch A (2009) Learning spectral graph transformations for link prediction. In: Proceedings of the international conference on machine learning, pp 561–568
30. Kunegis J, Lommatzsch A, Bauckhage C (2009) The slashdot zoo: mining a social network with negative edges. In: Proceedings of the international world wide web conference, pp 741–750

31. Kunegis J, Schmidt S, Lommatzsch A, Lerner J (2010) Spectral analysis of signed graphs for clustering, prediction and visualization. In: Proceedings of SIAM international conference on data mining, pp 559–570
32. Lü L, Jin C-H, Zhou T (2009) Similarity index based on local paths for link prediction of complex networks. Phys. Rev. E 80(4):046122
33. Lax PD (1984) Linear algebra and its applications. Wiley, London
34. Lee DD, Seung SH (2000) Algorithms for non-negative matrix factorization. In: Advances in neural information processing systems, pp 556–562
35. Leskovec J, Backstrom L, Kumar R, Tomkins A (2008) Microscopic evolution of social networks. In: Proceedings of the international conference on knowledge discovery and data mining, pp 462–470
36. Leskovec J, Huttenlocher D, Kleinberg J (2010) Governance in social media: A case study of the Wikipedia promotion process. In: Proceedings of th international conference on weblogs and social media
37. Leskovec J, Kleinberg J, Faloutsos C (2007) Graph evolution: densification and shrinking diameters. ACM Trans Knowl Discov Data 1(1):1–40
38. Ley M (2002) The DBLP computer science bibliography: evolution, research issues, perspectives. In: Proceedings of the international symposium on string processing and information retrieval, pp 1–10
39. Liben-Nowell, D. and Kleinberg, J. (2003), The link prediction problem for social networks. In: Proceedings of the international conference on information and knowledge management, pp 556–559
40. Liu W, Qian B, Cui J, Liu J (2009) Spectral kernel learning for semi-supervised classification. In: Proceedings of the international joint conference on artificial intelligence, pp 1150–1155
41. Long B, Zhang Z, Yu PS (2010) A general framework for relation graph clustering. J Knowl Inf Syst 24(3):393–413
42. Lu Z, Jain P, Dhillon IS (2009) Geometry-aware metric learning In: Proceedings of the international conference on machine learning, pp 673–680
43. von Luxburg U (2007) A tutorial on spectral clustering. Stat Comput 17(4):395–416
44. Manning CD, Raghavan P, Schütze H (2008) Introduction to information retrieval. Cambridge University Press, Cambridge
45. Massa P, Avesani P (2005) Controversial users demand local trust metrics: an experimental study on epinions.com community. In: Proceedings of American association for artificial intelligence conference, pp 121–126
46. Mislove A (2009) Online social networks: measurement, analysis, and applications to distributed information systems. PhD thesis, Rice University
47. Mislove A, Koppula HS, Gummadi KP, Druschel P, Bhattacharjee B (2008) Growth of the Flickr social network. In: Proceedings of the workshop on online social networks, pp 25–30
48. Mohar B (1991) The Laplacian spectrum of graphs. Graph Theory Comb Appl 2:871–898
49. Najork MA, Zaragoza H, Taylor MJ (2007) HITS on the Web: how does it compare? In: Proceedings of the international conference on research and development in information retrieval, pp 471–478
50. Newman MEJ (2006) Finding community structure in networks using the eigenvectors of matrices. Phys Rev E 74(3):036104
51. Peng W, Li T (2011) Temporal relation co-clustering on directional social network and author-topic evolution. J Knowl Inf Syst 26(3):467–486
52. Radl A, von Luxburg U, Hein M (2009) The resistance distance is meaningless for large random geometric graphs. In: Proceedings of the workshop on analyzing networks and learning with graphs
53. Sarwar B, Karypis G, Konstan J, Riedl J (2000) Application of dimensionality reduction in recommender systems-a case study. In: Proceedings of ACM WebKDD Workshop
54. Smola A, Kondor R (2003) Kernels and regularization on graphs. In: Proceedings of the conference on learning theory and kernel machines, pp 144–158
55. Stewart GW (1990) Perturbation theory for the singular value decomposition, Technical report. University of Maryland, College Park
56. Sun J, Tao D, Faloutsos C (2006) Beyond streams and graphs: dynamic tensor analysis. In: Proceedings of the international conference on knowledge discovery and data mining, pp 374–383
57. Thurau C, Kersting K, Wahabzada M, Bauckhage C (2011) Convex non-negative matrix factorization for massive datasets. J Knowl Inf Syst 29(2):457–478
58. Viswanath B, Mislove A, Cha M, Gummadi KP (2009) On the evolution of user interaction in Facebook. FIn: Proceedings of the workshop on online social networks, pp 37–42

59. Wax M, Sheinvald J (1997) A least-squares approach to joint diagonalization. IEEE Signal Process Lett 4(2):52–53
60. Wedin P-Å (1972) Perturbation bounds in connection with singular value decomposition. BIT Numer Math 12(1):99–111
61. Weyl H (1912) Das asymptotische Verteilungsgesetz der Eigenwerte linearer partieller Differenzialgleichungen (mit einer Anwendung auf die Theorie der Hohlraumstrahlung). Math Ann 71(4):441–479
62. Wilkinson JH (1965) The algebraic eigenvalue problem. Oxford University Press, Oxford
63. Zhang B, Liu R, Massey D, Zhang L (2005) Collecting the internet AS-level topology. SIGCOMM Comput Commun Rev 35(1):53–61
64. Zhang D, Mao R (2008) Classifying networked entities with modularity kernels. In Proceedings of the conference on information and knowledge management, pp 113–122
65. Zhu X, Kandola J, Lafferty J, Ghahramani Z (2006) Semi-supervised learning, MIT Press, chapter graph kernels by Spectral Transforms

Author Biographies

Jérôme Kunegis is a postdoctoral researcher at the Institute for Web Science and Technologies (WeST) at the University of Koblenz-Landau, Germany. He graduated at the Berlin Institute of Technology (TU Berlin) in 2006 and received his Ph.D. at the University of Koblenz-Landau in 2011 for his work on the spectral evolution model under the supervision of Prof. Steffen Staab. With main research interest in Web science, network science, data mining, and machine learning, Dr. Kunegis is best known for his work on spectral network analysis, as well as for the KONECT Project (Koblenz Network Collection), a large-scale effort to assemble network datasets for use in Web Science, network science, and other disciplines.

Damien Fay is a research fellow at University College Cork, Ireland. He obtained a B.Eng from University College Dublin in 1995, an MEng in 1997, and a Ph.D. from Dublin City University in 2003 and worked as a mathematics lecturer at the National University of Ireland in Galway from 2003 to 2007 before joining the NetOS group at Computer Laboratory in Cambridge from 2007 to 2010 as a research associate. His research interests include applied graph theory and time series analysis.

Christian Bauckhage is a professor of media informatics at the University of Bonn, Germany, and lead scientist for multimedia pattern recognition at the Fraunhofer Institute for Intelligent Analysis and Information Systems (IAIS). He obtained a Ph.D. in computer Science from Bielefeld University, Germany, and was a postdoctoral researcher in the Centre for Vision Research of York University in Toronto. Later, he worked as a senior research scientist at Deutsche Telekom Laboratories (T-Labs) in Berlin, where he conducted and coordinated industrial ICT research. He is an expert in large-scale data mining and pattern recognition and researches computational approaches to knowledge extraction from social media. He is a regular reviewer for leading journals and conferences and has (co)authored more than 100 technical papers on pattern recognition, computer vision, Web data mining, and future Internet-related topics.