Pattern Recognition 44 (2011) 1212–1224

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Semi-supervised classification and betweenness computation on large, sparse, directed graphs

Amin Mantrach [a,*], Nicolas van Zeebroeck [a], Pascal Francq [b], Masashi Shimbo [c], Hugues Bersini [a], Marco Saerens [b]

[a] IRIDIA Laboratory, Université Libre de Bruxelles, 50 Av. Fr. Roosevelt, B-1050 Brussels, Belgium
[b] ISYS/LSM & Machine Learning Group, Université de Louvain, Place des Doyens 1, B-1348 Louvain-la-Neuve, Belgium
[c] Graduate School of Information Science, Nara Institute of Science and Technology, 8916-5 Takayama, Ikoma, Nara, Japan

ARTICLE INFO

Article history:
Received 7 December 2009
Received in revised form
20 October 2010
Accepted 25 November 2010
Available online 30 November 2010

Keywords:
Graph mining
Semi-supervised classification
Within-network classification
Betweenness centrality
Graph-based classification
Kernel methods
Kernel on a graph
Large-scale graphs

ABSTRACT

This work addresses graph-based semi-supervised classification and betweenness computation in large, sparse, networks (several millions of nodes). The objective of semi-supervised classification is to assign a label to unlabeled nodes using the whole topology of the graph and the labeling at our disposal. Two approaches are developed to avoid explicit computation of pairwise proximity between the nodes of the graph, which would be impractical for graphs containing millions of nodes. The first approach directly computes, for each class, the sum of the similarities between the nodes to classify and the labeled nodes of the class, as suggested initially in [1,2]. Along this approach, two algorithms exploiting different state-of-the-art kernels on a graph are developed. The same strategy can also be used in order to compute a betweenness measure. The second approach works on a trellis structure built from biased random walks on the graph, extending an idea introduced in [3]. These random walks allow to define a biased bounded betweenness for the nodes of interest, defined separately for each class. All the proposed algorithms have a *linear computing time* in the number of edges while providing good results, and hence are applicable to large sparse networks. They are empirically validated on medium-size standard data sets and are shown to be competitive with state-of-the-art techniques. Finally, we processed a novel data set, which is made available for benchmarking, for multi-class classification in a large network: the *U.S. patents citation network* containing 3M nodes (of six different classes) and 38M edges. The three proposed algorithms achieve competitive results (around 85% classification rate) on this large network–they classify the unlabeled nodes within a few minutes on a standard workstation.

1. Introduction

Within-network semi-supervised classification has received a growing focus in recent years (see [4,5] for a comprehensive survey of the topic). In such a setting, one tries to assign a label to the unlabeled nodes of a graph. Since the topology of the entire graph is used (including the unlabeled nodes), the problem is semi-supervised. Despite the growing need for dealing with huge real-world networks, few of the existing methods scale up to large graphs[1] so that semi-supervised classification on large graphs has become one of the current central issues; see the survey [4, Section 6.3]. Indeed, the techniques that scale well [6] are not always competitive when compared to state-of-the-art graph-based metrics [7] such as the regularized Laplacian kernel [8], the sum-over-paths (SoP) covariance [9], the random walk with restart similarity and its normalized version [10,11,7], or the Markov diffusion kernel [12]. A naive application of these graph kernel-based approaches does not scale well since it relies on the computation of a dense similarity matrix, which usually requires a matrix inversion. Techniques approximating the inverse of the matrix usually require some strong properties on the matrix, like the positive semi-definiteness [13], and are only conceivable for medium-size graphs (up to 50,000 nodes)—for larger graphs, a dense similarity matrix cannot be computed and stored into memory.

This paper tackles this problem with two different approaches. The first approach is based on existing, competitive, kernels on a graph, but it explicitly avoids the computation of the pairwise similarities between the nodes (following an idea suggested by Zhou et al. [1,2]). Indeed, as opposed to [11,14,9], Zhou et al. suggest

* Corresponding author. Tel.: +32 650 27 79; fax: +32 650 27 15.
E-mail addresses: amantrac@ulb.ac.be (A. Mantrach), Nicolas.van.Zeebroeck@ulb.ac.be (N. van Zeebroeck), pascal.francq@uclouvain.be (P. Francq), shimbo@is.naist.jp (M. Shimbo), bersini@ulb.ac.be (H. Bersini), marco.saerens@uclouvain.be (M. Saerens).

[1] In this work, we consider that a graph is large scale when the number of nodes exceeds 10^5—one graph investigated in this work has more than 3×10^6 nodes.

0031-3203/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2010.11.019

to avoid computing each pairwise measure and solving a system of linear equations instead. We design two iterative algorithms along this approach, each based on a different state-of-the-art similarity metric: the SoP covariance kernel [9] and the normalized random walk with restart. This kernel on a graph was called regularized commute-time kernel in [7] and is closely related to the modified Laplacian matrix [15], as well as the random walk with restart similarity [10,11]. As suggested initially in [1,2], the algorithms directly approximate the sum of similarities to labeled nodes. The second approach takes its inspiration from the randomized shortest path framework of [16,9] and the $\mathcal{D}-$walk algorithm based on bounded random walks [3]. In this case, a random walk betweenness [17], measuring how well a node is "in-between" each class, is derived from a trellis structure constructed from a biased random walk on the graph.

1.1. Contribution and organization of the paper

This work makes three main contributions:

- It provides three algorithms to address within-network semi-supervised classification tasks on large, sparse, directed graphs. All these algorithms have a *computing time linear in the number of edges* of the graph. Moreover, an algorithm allowing to compute the SoP betweenness centrality [9] is also proposed.
- It validates the three proposed algorithms on eight medium-size standard data sets and compares them to state-of-the-art techniques. Their performances are shown to be competitive in comparison with the other techniques. Results are also computed on a standard large-scale data set [18].
- It introduces a *novel benchmark data set*: The U.S. patents citation network, on which our three algorithms obtain competitive results.

The subsequent part, Section 2, introduces the necessary background and notations. Then, in Section 3, two iterative algorithms are derived from the assumptions of local and global consistency. Further, Section 4 defines a biased bounded betweenness and proposes a forward/backward algorithm to compute it. Section 5 applies our three algorithms to semi-supervised classification tasks and compares the results to various state-of-the-art techniques. A novel benchmark data set is also introduced: the U.S. patents citation network on which our three algorithms are assessed. Section 6 discusses the related work. Finally, the last part of the article includes conclusions and remarks as well as further extensions.

2. Background and notations

Consider a weighted directed graph or network, G, not necessarily strongly connected, with a set of n nodes V (or vertices) and a set of arcs E (or edges). Also consider a set of classes, $\mathcal{L}$. It is assumed that each node belongs to exactly one class—but the class label can be unknown. Moreover, let us define an n-dimensional indicator vector, $\mathbf{y}^c$, containing as entries 1 when the corresponding node belongs to class c and 0 otherwise (in which case the node is unlabeled or belongs to another class). To each arc linking node k and k' a positive number $c_{kk'}>0$, representing the *immediate cost* of following this arc, is associated. The *cost matrix* $\mathbf{C}$ is the matrix containing the immediate costs $c_{kk'}$ as elements.

A random walk on this graph is defined in the standard way. In node k, the random walker chooses the next arc to follow according to transition probabilities representing the probability of jumping from node k to node $k'\in S(k)$, the set of successor nodes (successors S). These transition probabilities will be denoted as $p_{kk'}$ with $k'\in S(k)$. If there is no arc between k and k', we simply consider that $c_{kk'}$ takes an arbitrary large value, denoted by ∞; in this case, the corresponding transition probability will be set to zero, $p_{kk'}=0$. The *natural random walk* on this graph is defined in the following way: it corresponds to a random walk with transition probabilities

$$p_{kk'}=\frac{1/c_{kk'}}{\sum_{k'\in S(k)}(1/c_{kk'})} \tag{1}$$

The corresponding transition matrix will be denoted as $\mathbf{P}$. In other words, the random walker chooses to follow an arc with a probability proportional to the inverse of the immediate cost (apart from the sum-to-one normalization), therefore locally favoring arcs having a low cost. Instead of $\mathbf{C}$, we might be given an adjacency matrix $\mathbf{A}$ with elements $a_{kk'}\geq 0$ indicating the affinity between node k and node k'. In this case, the corresponding costs could be computed from $c_{kk'}=1/a_{kk'}$ and the transition probabilities associated to each node are simply proportional to the affinities (and normalized).

3. Kernel-based semi-supervised classification on large sparse graphs

Three approaches for semi-supervised classification on large sparse graphs are investigated in this paper. The first two approaches (detailed in Sections 3.2 and 3.3) are based on approximating, or bounding, standard kernel-based techniques, and are developed in this section. They will therefore be referred to as *approximate approaches*. The third approach, discussed in detail in Section 4, is a generalization of the discriminative random walks classifier ($\mathcal{D}-$walks, [3]).

3.1. Kernel-based classification

The approximate approaches are kernel-based and adopt the simple following classification procedure (the consistency method), initially proposed by Zhou et al. in [1,2] (see also [19–22]). Based on a regularization framework for the optimization a loss function, this classification procedure takes both available class labels and smoothness into account. The resulting decision procedure is based on a simple sum of similarities (each similarity being provided by an element of the graph kernel matrix) with the labeled nodes (as described in [1,2] for instance). This technique has been used with other kernels than those initially proposed by Zhou et al. with competitive results [9,7]. It corresponds to a simple alignment between the kernel matrix and the class membership vector.

More precisely, suppose that we are given a meaningful proximity matrix $\mathbf{K}$ (usually a graph kernel matrix; see e.g. [23]) providing similarities k_{ij} between each pair of nodes of the graph G. For each node, its total similarity with nodes belonging to class c is contained in the column vector $\mathbf{s}^c=\mathbf{K}\mathbf{y}^c$. Then, each node is assigned to the class showing the largest similarity; the predicted class index is thus provided by $\mathrm{argmax}_c(\mathbf{s}^c)$ for all nodes. In this section, we propose to directly estimate this sum of similarities $\mathbf{s}^c$ for two different metrics, i.e. the *sum-over-paths* (SoP) covariance [9] and the so-called *regularized commute-time kernel* [7], called in this paper the *normalized random walk with restart*.

The SoP covariance kernel is related to other, already available, kernels on a graph [23,12,7], such as, e.g., the commute-time kernel [24,25]. The commute-time kernel is the natural kernel associated to the commute-time distance (also called resistance distance [26]), the average number of steps that a random walker, starting from a given node, takes for entering another node for the first time and afterward going back to the starting node. As explained in [7,24], most of these kernels on a graph define a similarity measure

taking into account all paths – both direct and indirect – between graph nodes. They have the nice property of increasing when the number of paths connecting two nodes increases and when the "length" of any path decreases. In short, the more short paths connect two nodes, the more similar those nodes are. On the contrary, the usual "shortest path" (also called "geodesic" or "Dijkstra" distance) between nodes of a graph does not necessarily decrease when connections between nodes are added and thus it does not capture the fact that strongly connected nodes are at a smaller distance than weakly connected ones. These similarity measures are usually easy to compute and have an intuitive interpretation, although they do not scale well for large networks.

The regularized commute-time kernel is closely related to the modified Laplacian matrix [15] and the random walk with restart (RWR) similarity measure [10,11], that have been shown to be competitive on such tasks [9,7,2]. Because of the widespread use and the popularity of the RWR measure, the regularized commute-time kernel will be called the *normalized random walk with restart* kernel in this paper. Notice that such iterative updates for semi-supervised classification have been seen earlier in [1,27–29]; for a good review of these techniques see [18, Chapter 11]. In the remainder, we propose to apply such iterative procedure to two recently introduced graph kernels – that have been shown to perform well previously on medium size graphs [9,7] – not yet applied on large-scale networks. Finally, notice also that column i of the kernel matrix, $\mathbf{Ke}_i$ (the similarities from node i), can be approximated with the same method by using $\mathbf{e}_i$, i.e. column i of the identity matrix, instead of $\mathbf{y}^c$.

3.2. The approximate sum-over-paths covariance

This first approach starts from the SoP covariance kernel. According to this measure [9], two nodes are considered as highly correlated if they often co-occur together on the same – preferably short – paths. This leads to the definition of a covariance kernel capturing similarities between pairs of nodes.

Let us first introduce the matrix $\mathbf{W}$, corresponding to Eq. (23) of [9],

$$\mathbf{W}=\mathbf{P}\circ\exp[-\theta\mathbf{C}]=\exp[-\theta\mathbf{C}+\ln\mathbf{P}] \tag{2}$$

where $\mathbf{P}$ is the transition matrix containing the $p_{kk'}$, and the logarithm/exponential functions are taken elementwise. Moreover, $\circ$ is the elementwise (Hadamard) matrix product. $\mathbf{W}$ contains as elements $w_{kk'}=p_{kk'}\exp[-\theta c_{kk'}]$. If we set $\mathbf{Z}=(\mathbf{I}-\mathbf{W})^{-1}$, then the *SoP covariance* between node k and node l (see [9, Eq. (41)]) is

$$\begin{aligned}\mathrm{cov}(k,l)=\frac{1}{\mathcal{Z}}\{&(z_{\bullet k}-1)z_{k\bullet}\delta_{kl}+z_{k\bullet}(z_{\bullet l}-1)(z_{lk}-\delta_{lk})\\&+z_{l\bullet}(z_{\bullet k}-1)(z_{kl}-\delta_{kl})-\frac{z_{k\bullet}z_{l\bullet}(z_{\bullet k}-1)(z_{\bullet l}-1)}{\mathcal{Z}}\}\end{aligned} \tag{3}$$

where $z_{k\bullet}=\sum_{k'=1}^{n}z_{kk'}$, $z_{\bullet k}=\sum_{k'=1}^{n}z_{k'k}$, $z_{\bullet\bullet}=\sum_{k,k'=1}^{n}z_{kk'}$, $z_{kk'}$ is element k,k' of $\mathbf{Z}$, $\mathcal{Z}=z_{\bullet\bullet}-n$, and δ_{kl} is the Kronecker delta whose value is 1 if $k=l$ and 0 otherwise. On the other hand, the *SoP betweenness centrality* measure (see [9, Eq. (40)]), is

$$\mathrm{bet}(k)=\frac{(z_{\bullet k}-1)z_{k\bullet}}{\mathcal{Z}} \tag{4}$$

and corresponds to the expected number of times node k appears on a path through the network.

As already mentioned, the goal here is to directly approximate $\mathbf{s}^c=\mathbf{K}\mathbf{y}^c$ where $\mathbf{K}$ is the *SoP covariance kernel matrix* containing the elements $\mathrm{cov}(k,l)$ (see Eq. (3)). For the sake of readability, let us fix a specific class c in the remainder of this Section 3, and omit the superscript c from $\mathbf{y}^c$. Now the sum of similarities between node k and the labeled nodes (of class c) is

$$\begin{aligned}\sum_l\mathrm{cov}(k,l)y_l=\frac{1}{\mathcal{Z}}&\left(z_{\bullet k}\sum_l z_{\bullet l}z_{lk}y_l+z_{\bullet k}\sum_l z_{l\bullet}z_{kl}y_l-z_{k\bullet}\sum_l z_{lk}y_l\right.\\&\left.-\sum_l z_{l\bullet}z_{kl}y_l-z_{\bullet k}z_{k\bullet}y_k+z_{k\bullet}y_k\right)\\&-\frac{1}{\mathcal{Z}^2}\left(z_{k\bullet}z_{\bullet k}\sum_l z_{l\bullet}z_{\bullet l}y_l-z_{k\bullet}z_{\bullet k}\sum_l z_{l\bullet}y_l-z_{k\bullet}\sum_l z_{l\bullet}z_{\bullet l}y_l\right.\\&\left.+z_{k\bullet}\sum_l z_{l\bullet}y_l\right)\end{aligned} \tag{5}$$

If we denote element k of a vector $\mathbf{x}$ as $[\mathbf{x}]_k$ and the diagonal matrix constructed from the vector $\mathbf{y}$ as $\mathbf{Diag}(\mathbf{y})$ and define

$$\mathbf{x}_0=\mathbf{Ze}\quad\text{thus } z_{k\bullet}=\sum_l z_{kl}=[\mathbf{x}_0]_k \tag{6}$$

$$\mathbf{x}_1=\mathbf{Z}^{\mathrm{T}}\mathbf{e}\quad\text{thus } z_{\bullet k}=\sum_l z_{lk}=[\mathbf{x}_1]_k \tag{7}$$

$$\mathbf{x}_2=\mathbf{Z}^{\mathrm{T}}\mathbf{Diag}(\mathbf{y})\mathbf{x}_1\quad\text{thus }\sum_l z_{\bullet l}z_{lk}y_l=[\mathbf{x}_2]_k \tag{8}$$

$$\mathbf{x}_3=\mathbf{ZDiag}(\mathbf{y})\mathbf{x}_0\quad\text{thus }\sum_l z_{l\bullet}z_{kl}y_l=[\mathbf{x}_3]_k \tag{9}$$

$$\mathbf{x}_4=\mathbf{Z}^{\mathrm{T}}\mathbf{y},\quad\text{thus }\sum_l z_{lk}y_l=[\mathbf{x}_4]_k \tag{10}$$

$$x_5=\mathbf{x}_1^{\mathrm{T}}\mathbf{Diag}(\mathbf{y})\mathbf{x}_0=\sum_l z_{l\bullet}z_{\bullet l}y_l \tag{11}$$

$$x_6=\mathbf{e}^{\mathrm{T}}\mathbf{Diag}(\mathbf{y})\mathbf{x}_0=\sum_l z_{l\bullet}y_l \tag{12}$$

then Eq. (5) can be rewritten as

$$\begin{aligned}\sum_l\mathrm{cov}(k,l)y_l=\frac{1}{\mathcal{Z}}&([\mathbf{x}_0]_k[\mathbf{x}_2]_k+[\mathbf{x}_1]_k[\mathbf{x}_3]_k-[\mathbf{x}_0]_k[\mathbf{x}_4]_k-[\mathbf{x}_3]_k\\&-[\mathbf{x}_1]_k[\mathbf{x}_0]_k[\mathbf{y}]_k+[\mathbf{x}_0]_k[\mathbf{y}]_k)-\frac{1}{\mathcal{Z}^2}([\mathbf{x}_0]_k[\mathbf{x}_1]_k x_5\\&-[\mathbf{x}_0]_k[\mathbf{x}_1]_k x_6-[\mathbf{x}_0]_k x_5+[\mathbf{x}_0]_k x_6)\end{aligned} \tag{13}$$

where the partition function $\mathcal{Z}$ is computed by $z_{\bullet\bullet}-n=\mathbf{x}_0^{\mathrm{T}}\mathbf{e}-n$ ($\mathbf{e}$ is a column vector full of 1's). The algorithm for computing this quantity consists in first solving the two systems of linear equations (6) and (7), which may be solved iteratively (Indeed, $\rho(\mathbf{W})<1$, see [9].). For example, here is the way Eq. (6) is solved:

$$\begin{aligned}\mathbf{Ze}=\mathbf{x}_0&\Longrightarrow(\mathbf{I}-\mathbf{W})^{-1}\mathbf{e}=\mathbf{x}_0\\&\Longrightarrow\mathbf{e}=(\mathbf{I}-\mathbf{W})\mathbf{x}_0\\&\Longrightarrow\mathbf{x}_0=\mathbf{W}\mathbf{x}_0+\mathbf{e}\end{aligned} \tag{14}$$

from which we obtain the iterative updating scheme

$$\begin{cases}\widehat{\mathbf{x}}_0(0)\leftarrow\mathbf{e}\\ \widehat{\mathbf{x}}_0(t+1)\leftarrow\mathbf{W}\widehat{\mathbf{x}}_0(t)+\mathbf{e}\end{cases} \tag{15}$$

Of course, more sophisticated methods, like conjugate gradient techniques [30], could be used instead. Afterwards, using the same trick, we solve the systems of linear equations (8)–(10) which are again solved iteratively. Finally, Eqs. (11) and (12) are computed directly. The complexity of each iteration is $\mathcal{O}(|E|)$ since the matrix is assumed to be sparse. So the global complexity is approximately $\mathcal{O}(\tau|E||\mathcal{L}|)$ where τ is the number of iterations. Note that we might directly compute the results for all the $\mathbf{y}^c$ labeling vector classes by

using (instead of a column vector $\mathbf{y}^c$) the concatenation of these vectors in a matrix $\mathbf{Y}$. In this case, depending on the architecture, solving the equations may be transparently parallelized. Indeed, we can easily compute the covariance for each class independently of the others. The computation may thus be parallelized directly on different cores, or CPUs. Finally, the assigned class index is provided by $\text{argmax}_c(\mathbf{K}\mathbf{y}^c)$ for all nodes.

Moreover, from Eq. (4), the SoP betweenness centrality of node k can be computed from $\text{bet}(k) = ([\mathbf{x}_1]_k - 1)[\mathbf{x}_0]_k / \mathcal{Z}$.

3.3. The bounded normalized random walk with restart

In this second approach, we will approximate a second kernel-based method, referred to as the normalized random walk with restart (and called the regularized commute-time kernel in [7]), by bounding the underlying random walk. The notion of approximating stationary quantities for random walks by only considering paths up to a specified length τ has been already investigated in [3,31,27,32] (for more details, see the related work in Section 6). We propose to apply this idea on the normalized random walk with restart kernel, and provide an interesting interpretation of the bounding. This kernel on a graph is closely related to the modified Laplacian matrix [15], the commute-time kernel [24,25] and the well-known random walk with restart similarity [10,11]. The normalized random walk with restart matrix $\mathbf{K}$ is given by

$$\mathbf{K} = (\mathbf{D} - \alpha\mathbf{A}^{\mathrm{T}})^{-1} \tag{16}$$

where $\mathbf{A}$ is the adjacency matrix, $\mathbf{D} = \mathbf{Diag}(\mathbf{A}\mathbf{e})$, and $\mathbf{e}$ is a column vector full of 1's. If matrix $\mathbf{A}$ is symmetric, Eq. (16) defines a valid kernel on a graph. As shown now, the parameter $\alpha \in]0,1[$ denotes, at each time step of a random walk, the probability that the random walker continues his walk. We are looking for a way to bound the sum-of-similarities $\mathbf{s} = \mathbf{K}\mathbf{y}$ up to a a priori-specified walk length τ. Following [7], since the transition matrix of the natural random walk on the graph is $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$, we may note that

$$\mathbf{K}\mathbf{y} = (\mathbf{D} - \alpha\mathbf{A}^{\mathrm{T}})^{-1}\mathbf{y} \tag{17}$$

$$= \mathbf{D}^{-1}\mathbf{D}(\mathbf{D} - \alpha\mathbf{A}^{\mathrm{T}})^{-1}\mathbf{y} \tag{18}$$

$$= \mathbf{D}^{-1}((\mathbf{D} - \alpha\mathbf{A}^{\mathrm{T}})\mathbf{D}^{-1})^{-1}\mathbf{y} \tag{19}$$

$$= \mathbf{D}^{-1}(\mathbf{I} - \alpha\mathbf{P}^{\mathrm{T}})^{-1}\mathbf{y} \tag{20}$$

$$= \mathbf{D}^{-1}\left(\sum_{t=0}^{\infty}(\alpha\mathbf{P}^{\mathrm{T}})^t\right)\mathbf{y} = \widehat{\mathbf{s}}(\infty) \tag{21}$$

Thus, intuitively, random walkers start from the labeled nodes with an initial distribution $\mathbf{x}(0) = \mathbf{y}$. Then, they diffuse with transition matrix $\alpha\mathbf{P}$, which is substochastic. Therefore, these random walkers have a non-zero probability of disappearing (giving up the walk) at each time step. Let us denote by $\mathbf{x}(t)$ the column vector containing the expected number of random walkers in a specific node of the network after t steps of the random walk; thus, $\mathbf{x}(t) = \alpha\mathbf{P}^{\mathrm{T}}\mathbf{x}(t-1)$ and $\mathbf{x}(0) = \mathbf{y}$. Eq. (21) tells us that the sum of similarities $\mathbf{K}\mathbf{y}$ is simply the normalized (by $\mathbf{D}^{-1}$) cumulated sum of expected visits to each node, $\mathbf{K}\mathbf{y} = \mathbf{D}^{-1}\sum_{t=0}^{\infty}\mathbf{x}(t)$. Bounding the walks aims to truncate this series up to term τ, $\widehat{\mathbf{s}}(\tau) = \mathbf{D}^{-1}\sum_{t=0}^{\tau}\mathbf{x}(t)$. The normalizing factor $\mathbf{D}^{-1}$ has the effect of weighting the intrinsic importance of popular nodes [33,34]. Since $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$, it can easily be shown that $\widehat{\mathbf{s}}(t)$ may be computed using the following recurrence equation:

$$\begin{cases} \widetilde{\mathbf{s}}(0) \leftarrow \mathbf{y} \\ \widetilde{\mathbf{s}}(t) \leftarrow \alpha\mathbf{P}^{\mathrm{T}}\widetilde{\mathbf{s}}(t-1) + \widetilde{\mathbf{s}}(0) \quad \text{for } t = 1,\ldots,\tau \\ \widehat{\mathbf{s}}(\tau) \leftarrow \mathbf{D}^{-1}\widetilde{\mathbf{s}}(\tau) \end{cases} \tag{22}$$

This recurrence scheme can be iterated until convergence, but, in our experiments, we stop the iteration at $t = \tau$ steps, which is equivalent to bounding the random walk up to τ steps. The parameter τ will be tuned by an internal cross-validation.

Notice that the matrix $(\mathbf{I} - \alpha\mathbf{P}^{\mathrm{T}})^{-1}$ in Eq. (20) coincides with the well-known random-walk with restart similarity matrix; it was used, e.g., in [10] for computing similarities between nodes and was inspired from Page et al.'s famous PageRank algorithm [35]. Eq. (22) holds for directed graphs as well, but in this case the similarity matrix $\mathbf{K}$ is no more a valid kernel. The time complexity of this algorithm is $\mathcal{O}(\tau|E||\mathcal{L}|)$, which is the same as that of the SoP approximation presented in Section 3.2. Note, however, that we only have one system of linear equations to solve, while the SoP approximation requires solving five systems of the same size. In terms of spatial complexity, we need to maintain one column vector at each time step for the current results and to store into memory the column vector $\mathbf{x}(0)$ and the sparse transition matrix $\mathbf{P}$. $\mathbf{x}(0)$ needs to be computed only once at the initialization time. Therefore, the space complexity is $\mathcal{O}(|E| + |\mathcal{V}|)$. Finally, the assigned class index is provided by $\text{argmax}_c(\mathbf{s}^c)$ for all nodes.

4. The biased $\mathcal{D}-$walks

Callut et al. [3] recently proposed still another random-walk based approach: the *discriminative random walks* ($\mathcal{D}-$walks), providing a class betweenness measure for classifying nodes in a graph. It computes a *group betweenness* index [36] with respect to a set of nodes—in this case, the labeled nodes belonging to the same class. This model performed well on a number of semi-supervised tasks [3]. In this section, we propose an extension of the $\mathcal{D}-$walks by using the randomized shortest path (RSP) framework introduced in [16,37,9]. By defining an entropy-related parameter θ that controls the global entropy spread by the random walker in the network, we may gradually bias the random walk towards short paths. For a parameter value of $\theta = 0$, our extension reduces to the $\mathcal{D}-$walks. On the other hand, for intermediate values of θ, the random walk is biased towards short paths, therefore avoiding, to a certain extend, loops or broad, irrelevant, walks. For additional motivations, the reader is advised to turn to [16].

A $\mathcal{D}-$walk relative to class c (see [3] for further details) is a random walk on a graph that starts (at $t = 0$) from a node belonging to class c and ends in a node of the same class c. Therefore, the nodes of class c are transformed into absorbing nodes when $t \geq 1$. More precisely, the approach considered here consists in applying the randomized shortest path framework on a lattice structure constructed from the original network whose weight matrix is referred to as $\widetilde{\mathbf{W}}^{\mathbf{c}}$. Any vector or matrix with a tilde will be n-dimensional. This matrix is directly computed from the exponential cost matrix $\mathbf{W}$ associated to the network (Eq. (2)), as will be explained below. Inspired by hidden Markov models [38], the main idea is the following: the original network G is *unfolded in time* in order to build a lattice L made of the network nodes at time steps $0,1,\ldots,\tau$. Transitions are only allowed from nodes at time t to successor nodes at time $t+1$ (see Fig. 1). The interpretation of this lattice is immediate: it represents a bounded random walk on the graph G with $t \in [0,\tau]$. The random walker starts in a node belonging to class c at time $t = 0$ and walks until either he reaches a node in class c, and is absorbed by this node, or stops at time $t = \tau$, that is, τ is the maximum walk length. The lattice L therefore contains $N = n \times (\tau + 1)$ nodes in total (Fig. 1).

This lattice will be considered as a new graph – which is acyclic this time – on which the randomized shortest-path framework can be applied [16]. Its $N \times N$ exponential costs matrix is denoted by $\boldsymbol{W}^c$

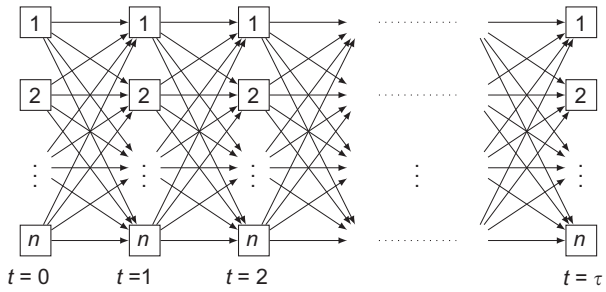

Fig. 1. A lattice L defined from the original graph G.

and is organized in $(\tau+1)\times(\tau+1)$ blocks of size $n\times n$:

$$W^c=\left(\begin{array}{c|c|c|c|c} 0 & \widetilde{\mathbf{W}}^c(1) & 0 & \dots & 0 \\ \hline 0 & 0 & \widetilde{\mathbf{W}}^c(2) & \dots & 0 \\ \hline \vdots & \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & 0 & \dots & \widetilde{\mathbf{W}}^c(\tau) \\ \hline 0 & 0 & 0 & \dots & 0 \end{array}\right)$$

where $\mathbf{0}$ is a matrix full of zeroes of the appropriate size, i.e. $n\times n$ in this case.

Most blocks are null matrices since the graph considered here is a lattice and only transitions between time steps t and $t+1$ are allowed. Hence, only submatrices at block $(t,\ t+1)$, (for $t=0,1,\dots,\tau$) may hold non-zero elements. Moreover, the matrices $\widetilde{\mathbf{W}}^c(t)$ are set equal to the $n\times n$ matrix $\mathbf{W}$ computed from Eq. (2), with one important modification: when $t=1$, $\widetilde{\mathbf{W}}^c(t)=\mathbf{W}$, but for $t>1$, the rows of the matrix corresponding to nodes labeled as class c are *replaced by zero rows*. This method aims at making these nodes absorbing: when a random walker hits one of these nodes, he stops his walk and disappears, following the strategy of $\mathcal{D}-$walks.

Two N-dimensional vectors, $\boldsymbol{h}_0^c$ and $\boldsymbol{h}_f^c$, are also defined,

$$h_0^c=\left(\begin{array}{c} \widetilde{\mathbf{h}}^c \\ \hline 0 \\ \hline \vdots \\ \hline 0 \\ \hline 0 \end{array}\right),\quad h_f^c=\left(\begin{array}{c} 0 \\ \hline \widetilde{\mathbf{h}}^c \\ \hline \vdots \\ \hline \widetilde{\mathbf{h}}^c \\ \hline \widetilde{\mathbf{h}}^c \end{array}\right)$$

These column vectors are indicator vectors defining the sets of starting and ending nodes of the random walks. The entries of the vector $\boldsymbol{h}_0^c$ ($\boldsymbol{h}_f^c$) are set to 1 if a path can start from (end into) the corresponding node, and 0 otherwise. They are also decomposed into $n\times 1$ blocks: $\widetilde{\mathbf{h}}^c$ is an $n\times 1$ vector whose values are equal to 1 if the node considered belongs to class c and 0 otherwise. In other words, $\widetilde{\mathbf{h}}^c=\mathbf{y}^c$. Since the paths (or walks) considered start at time step 0, the vector $\boldsymbol{h}_0^c$ holds zero elements for all time steps $t>0$. The vector $\boldsymbol{h}_f^c$ contains the $n\times 1$ vector $\widetilde{\mathbf{h}}^c$ for every time step larger than 0 since the random walker may stop at each time step $t=1,2,\dots,\tau$ (except $t=0$), when hitting a node of class c. For readability reasons, the superscript c of $\boldsymbol{h}_0^c$ and $\boldsymbol{h}_f^c$ is omitted in the remaining of this section.

The betweenness we are looking for is computed for each class c independently. Indeed, for each class, we consider all paths of length up to τ (in number of steps), starting from a node belonging to class c, and ending in a node of the same class. In other words, the betweenness measures to which extend a node is located in-between the nodes of the class of interest.

As detailed in [16,37,9] and already mentioned in Section 3.2, an important quantity appearing in the RSP framework is $\boldsymbol{Z}^c=(\mathbf{I}-\boldsymbol{W}^c)^{-1}$, called the fundamental matrix. Every quantity of interest can be obtained from this matrix (see [16] for details). For instance, the expected number of transitions through link $k\to k'$ (see [9, Appendix A, Eq. (49)]), from which our betweenness will be derived, is

$$\overline{\eta}_{kk'}^c=-\frac{1}{\theta}\frac{\boldsymbol{h}_0^{\mathrm{T}}\boldsymbol{Z}^c(\boldsymbol{W}^c)'_{kk'}\boldsymbol{Z}^c\boldsymbol{h}_f}{\mathcal{Z}^c} \tag{23}$$

where $(\boldsymbol{W}^c)'_{kk'}$ is the partial derivative of $\boldsymbol{W}^c$ with respect to $c_{kk'}$ and $\mathcal{Z}=\boldsymbol{h}_0^{\mathrm{T}}\boldsymbol{Z}^c\boldsymbol{h}_f$. If we define $(\boldsymbol{\alpha}^c)^{\mathrm{T}}=\boldsymbol{h}_0^{\mathrm{T}}\boldsymbol{Z}^c$ as the forward parameters vector by reference to hidden Markov models [38], then

$$(\boldsymbol{\alpha}^c)^{\mathrm{T}}=\boldsymbol{h}_0^{\mathrm{T}}\boldsymbol{Z}^c\Longrightarrow(\boldsymbol{\alpha}^c)^{\mathrm{T}}(\mathbf{I}-\boldsymbol{W}^c)=\boldsymbol{h}_0^{\mathrm{T}} \tag{24}$$

$$\Longrightarrow(\mathbf{I}-\boldsymbol{W}^c)^{\mathrm{T}}\boldsymbol{\alpha}^c=\boldsymbol{h}_0 \tag{25}$$

$$\Longrightarrow\boldsymbol{\alpha}^c=(\boldsymbol{W}^c)^{\mathrm{T}}\boldsymbol{\alpha}^c+\boldsymbol{h}_0 \tag{26}$$

Symmetrically, we define $\boldsymbol{\beta}^c=\boldsymbol{Z}^c\boldsymbol{h}_f$ as the backward parameters vector,

$$\boldsymbol{\beta}^c=\boldsymbol{Z}^c\boldsymbol{h}_f\Longrightarrow(\mathbf{I}-\boldsymbol{W}^c)\boldsymbol{\beta}^c=\boldsymbol{h}_f \tag{27}$$

$$\Longrightarrow\boldsymbol{\beta}^c=\boldsymbol{W}^c\boldsymbol{\beta}^c+\boldsymbol{h}_f \tag{28}$$

Decomposing $\boldsymbol{\alpha}^c$ into blocks in Eq. (26) yields

$$\left(\begin{array}{c} \widetilde{\boldsymbol{\alpha}}^c(0) \\ \widetilde{\boldsymbol{\alpha}}^c(1) \\ \vdots \\ \widetilde{\boldsymbol{\alpha}}^c(\tau-1) \\ \widetilde{\boldsymbol{\alpha}}^c(\tau) \end{array}\right)=\left(\begin{array}{ccccccc} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ (\widetilde{\mathbf{W}}^c(1))^{\mathrm{T}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & (\widetilde{\mathbf{W}}^c(2))^{\mathrm{T}} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & (\widetilde{\mathbf{W}}^c(\tau))^{\mathrm{T}} & \mathbf{0} \end{array}\right)\times\left(\begin{array}{c} \widetilde{\boldsymbol{\alpha}}^c(0) \\ \widetilde{\boldsymbol{\alpha}}^c(1) \\ \vdots \\ \widetilde{\boldsymbol{\alpha}}^c(\tau-1) \\ \widetilde{\boldsymbol{\alpha}}^c(\tau) \end{array}\right)+\left(\begin{array}{c} \widetilde{\mathbf{h}}^c \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{0} \end{array}\right)$$

which allows to obtain the following forward recurrence relations:

$$\begin{cases}\widetilde{\boldsymbol{\alpha}}^c(0)=\widetilde{\mathbf{h}}^c \\ \widetilde{\boldsymbol{\alpha}}^c(t+1)=(\widetilde{\mathbf{W}}^c(t+1))^{\mathrm{T}}\widetilde{\boldsymbol{\alpha}}^c(t)\end{cases} \tag{29}$$

Similarly, we do the same for $\boldsymbol{\beta}^c$ to obtain the backward recurrence relations:

$$\begin{cases}\widetilde{\boldsymbol{\beta}}^c(\tau)=\widetilde{\mathbf{h}}^c \\ \widetilde{\boldsymbol{\beta}}^c(t-1)=\widetilde{\mathbf{W}}^c(t)\widetilde{\boldsymbol{\beta}}^c(t)+\widetilde{\mathbf{h}}^c,\quad t>0.\end{cases} \tag{30}$$

Replacing $\boldsymbol{h}_0^T\boldsymbol{Z}^c$ (Eq. (24)) and $\boldsymbol{Z}^c\boldsymbol{h}_f$ (Eq. (27)) in Eq. (23) yields

$$\overline{\eta}_{kk'}^c=-\frac{1}{\theta}\frac{(\boldsymbol{\alpha}^c)^{\mathrm{T}}(\boldsymbol{W}^c)'_{kk'}\boldsymbol{\beta}^c}{\mathcal{Z}^c} \tag{31}$$

The partition function $\mathcal{Z}^c$ corresponds to the contribution of all the paths starting and ending in class c. As already stated, $\mathcal{Z}^c=\boldsymbol{h}_0^{\mathrm{T}}\boldsymbol{Z}^c\boldsymbol{h}_f$ and $\boldsymbol{\beta}^c=\boldsymbol{Z}^c\boldsymbol{h}_f$; hence, $\mathcal{Z}^c=\boldsymbol{h}_0^{\mathrm{T}}\boldsymbol{\beta}^c$. Since the $N\times 1$ column vector $\boldsymbol{h}_0$ is formed by an $n\times 1$ block which is $\widetilde{\mathbf{h}}^c$ and 0

values in the remaining positions, the product $\boldsymbol{h}_0^{\mathrm{T}}\boldsymbol{\beta}^c$ equals $(\widetilde{\mathbf{h}}^c)^{\mathrm{T}}\widetilde{\boldsymbol{\beta}}^c(0)$. Intuitively, it means that only paths starting at time step 0 contribute in the partition function. Therefore,

$$\overline{\eta}^c_{kk'} = -\frac{1}{\theta}\frac{(\boldsymbol{\alpha}^c)^{\mathrm{T}}(\boldsymbol{W}^c)'_{kk'}\boldsymbol{\beta}^c}{(\widetilde{\mathbf{h}}^c)^{\mathrm{T}}\widetilde{\boldsymbol{\beta}}^c(0)} \tag{32}$$

Now, differentiating $\boldsymbol{W}^c$ with respect to $c_{kk'}$ and assuming that node k does not belong to class c – in which case we obtain the trivial result $\overline{\eta}^c_{kk'}=0$ since k is absorbing – yields

$$(\boldsymbol{W}^c)'_{kk'} = -\theta p_{kk'}\exp[-\theta c_{kk'}]\begin{pmatrix} \mathbf{0} & \widetilde{\mathbf{e}}_k\widetilde{\mathbf{e}}_{k'}^{\mathrm{T}} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \widetilde{\mathbf{e}}_k\widetilde{\mathbf{e}}_{k'}^{\mathrm{T}} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \widetilde{\mathbf{e}}_k\widetilde{\mathbf{e}}_{k'}^{\mathrm{T}} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \widetilde{\mathbf{e}}_k\widetilde{\mathbf{e}}_{k'}^{\mathrm{T}} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \end{pmatrix}$$

where $\widetilde{\mathbf{e}}_k$ is an $n\times 1$ column vector containing a 1 in position k, and 0's everywhere else. Therefore, the expected number of transitions through link $k\to k'$ is simply the sum of the expected number of transitions over all time steps,

$$\overline{\eta}^c_{kk'} = \frac{\sum_{t=0}^{\tau-1}(\widetilde{\boldsymbol{\alpha}}^c(t))^{\mathrm{T}}\widetilde{\mathbf{e}}_k\widetilde{\mathbf{e}}_{k'}^{\mathrm{T}}\widetilde{\boldsymbol{\beta}}^c(t+1)}{(\widetilde{\mathbf{h}}^c)^{\mathrm{T}}\widetilde{\boldsymbol{\beta}}^c(0)}w_{kk'} \tag{33}$$

with $w_{kk'}=\exp(-\theta c_{kk'}+\ln p_{kk'})$. Consequently, the betweenness of node k' with respect to class c is simply the sum of incoming transitions

$$\mathrm{bet}^c_{k'} = \sum_{k=1}^{n}\overline{\eta}^c_{kk'} = \frac{\sum_{t=0}^{\tau-1}\sum_{k=1}^{n}\widetilde{\mathbf{e}}_k^{T}\widetilde{\boldsymbol{\alpha}}^c(t)\omega_{kk'}\widetilde{\boldsymbol{\beta}}^c(t+1)\widetilde{\mathbf{e}}_{k'}}{(\widetilde{\mathbf{h}}^c)^{\mathrm{T}}\widetilde{\boldsymbol{\beta}}^c(0)} \tag{34}$$

By observing from Eq. (29) that $\sum_{k=1}^{n}\widetilde{\mathbf{e}}_k^{\mathrm{T}}\widetilde{\boldsymbol{\alpha}}^c(t)\omega_{kk'}=\sum_{k=1}^{n}\widetilde{\alpha}^c_k(t)\omega_{kk'}=\widetilde{\alpha}^c_{k'}(t+1)$, we finally obtain for the biased $\mathcal{D}-$walk betweenness vector

$$\mathbf{bet}^c = \frac{\sum_{t=0}^{\tau-1}\widetilde{\boldsymbol{\alpha}}^c(t+1)\circ\widetilde{\boldsymbol{\beta}}^c(t+1)}{(\widetilde{\mathbf{h}}^c)^{\mathrm{T}}\widetilde{\boldsymbol{\beta}}^c(0)} \tag{35}$$

where $\circ$ is the elementwise (Hadamard) matrix product. This betweenness is very similar to the $\boldsymbol{\gamma}(t)$ variable computed in hidden Markov models [38,39] where it can be interpreted as the probability of being in some node after a walk of t steps without having visited any node of class c. Once the class betweenness has been computed, each node is assigned to the class showing the largest betweenness. The algorithm is detailed in Algorithm 1. The time complexity is $\mathcal{O}(\tau|E||\mathcal{L}|)$ since for each class $c\in|\mathcal{L}|$ we have to compute the forward and backward vectors which require τ steps each. Moreover, each vector computation imply a product between a sparse matrix and a column vector which is $\mathcal{O}(|E|)$. The space complexity is $\mathcal{O}(|E|+\tau|V|)$ since we have to store in memory both the matrix $\widetilde{\mathbf{W}}$ and the matrix $\widetilde{\mathbf{W}}^c$ and at the same time we have to keep in memory τ times the forward and backward vectors.

Algorithm 1. Computation of the biased $\mathcal{D}-$walk betweenness.

Input:

–A graph G containing n nodes.
–$\theta>0$: the parameter controlling the degree of exploration.
–$\mathbf{C}$: the $n\times n$ cost matrix associated to G, containing elements $c_{kk'}>0$.
–$\mathbf{P}$: the $n\times n$ transitions-probabilities matrix of a natural random walk.
–$\widetilde{\mathbf{y}}^c$: the $n\times 1$ class membership vector containing 1 for nodes belonging to the class c and 0 otherwise.
–τ: the maximum walk length considered.
–$|\mathcal{L}|$: the number of classes.

Output:

–The *betweenness* matrix $\mathbf{B}$ containing $|\mathcal{L}|$ columns where each column contains the betweenness of each node relatively to a class.

1. $\widetilde{\mathbf{W}}=\mathbf{P}\circ\exp[-\theta\mathbf{C}]$, where the exponential is taken elementwise and $\circ$ is the elementwise (Hadamard) matrix product.
2. **for** $c=1$ to $|\mathcal{L}|$ **do**
3. $\widetilde{\mathbf{W}}^c=\widetilde{\mathbf{W}}$ and set the rows for which $\widetilde{\mathbf{y}}^c=1$ to $\mathbf{0}^{\mathrm{T}}$.
4. $\widetilde{\boldsymbol{\beta}}^c(\tau)=\widetilde{\mathbf{y}}^c$ (initialization of the backward vector).
5. **for** $t=\tau$ to 1 **do**
6. $\widetilde{\boldsymbol{\beta}}^c(t-1)=\widetilde{\mathbf{W}}^c\widetilde{\boldsymbol{\beta}}^c(t)+\widetilde{\mathbf{y}}^c$
7. **end for**
8. $\widetilde{\boldsymbol{\alpha}}^c(1)=\widetilde{\mathbf{W}}^{\mathrm{T}}\widetilde{\mathbf{y}}^c$ (initialization of the forward vector).
9. **for** $t=1$ to $\tau-1$ **do**
10. $\widetilde{\boldsymbol{\alpha}}^c(t+1)=(\widetilde{\mathbf{W}}^c)^{\mathrm{T}}\widetilde{\boldsymbol{\alpha}}^c(t)$
11. **end for**
12. $\mathbf{B}(:,c)=\frac{\sum_{t=1}^{\tau}\widetilde{\boldsymbol{\alpha}}^c(t)\circ\widetilde{\boldsymbol{\beta}}^c(t)}{(\widetilde{\mathbf{y}}^c)^{\mathrm{T}}\widetilde{\boldsymbol{\beta}}^c(0)}$ where $\circ$ is the elementwise multiplication.
13. **end for**
14. **return B**

5. Experiments

This experimental section has two main goals. Firstly, the three approximate algorithms introduced in this paper are compared to their exact counterpart (without approximating or bounding) and to some state-of-the art techniques on several graph-based semi-supervised classification tasks over medium-size data sets. Secondly, the performance of our three proposed algorithms are further assessed and compared to two state-of-the art techniques on (i) a large 6-classes sparse network, consisting of the graph of citations between about 3 million U.S. patents and (ii) a standard large-scale protein secondary structure data set.

5.1. First experiment: validation of the approximate approaches

In this first part of the experiments, we address the task of classifying unlabeled nodes in partially labeled graphs on eight medium-size data sets (up to 5000 nodes). The goal is to compare the approximate approaches to the exact kernel-based techniques in terms of classification rate. This comparison is performed on medium-size networks only since kernel approaches cannot be computed on large networks.

Data sets. The different classification models, referred to as classifiers, are compared on eight data sets that were used previously for semi-supervised classification: the four universities `WebKB cocite` data sets [2,6], the two `industry` data sets [6], the `IMDb prodco` data set [6] and the `CoRA cite` data set [6].[2]

IMDb: The collaborative Internet Movie Database (`IMDb`, [6]) has several applications such as making movie recommendation or movie category classification. The classification problem focuses on the prediction of the movie notoriety (whether the movie is a box-office or not). It contains a graph of movies linked together

[2] The preprocessed version in Matlab/Octave format of the data sets used in this first experiment is available from http://iridia.ulb.ac.be/~amantrac/pub/SoP_TPAMI.zip [9].

whenever they share the same production company. The weight of an edge in the resulting graph is the number of production companies two movies have in common. The IMDb-proco graph contains 1169 movies and has the class distribution shown in Table 1.

Industry: Industry regroups two data sets [6]. The industry-pr data set is based on 35,318 Newswire press releases. The companies mentioned in each press release were extracted and an edge was placed between two companies if they appeared together in a press release. The same applies to the industry-yh data set based on 22,170 business news stories collected from the web. The weight of an edge is the number of such co-occurrences found in the complete corpus. To classify a company, Yahoo!'s 12 industry sectors have been used in the two industry data sets. The details about the two industry data sets are reported in Table 2.

CoRA: CoRA cite is a graph of 3583 nodes collected from machine learning research papers labeled into seven different topics [6]. Papers are linked if they share a common author, or if one cites the other. The composition of the CoRA cite data set is reported in Table 3.

Table 1
Class distribution for the IMDb-proco data set.

Category	Size
High-revenue	572
Low-revenue	597
Total	1169
Majority class proportion (%)	50.67

Table 2
Class distribution for the industry-yh and industry-pr data sets.

Category	Size	
	industry-yh	industry-pr
Basic materials	104	83
Capital Goos	83	78
Conglomerates	14	13
Consumer cyclical	99	94
Consumer noncyclical	60	59
Energy	71	112
Financial	170	268
Heathcare	180	279
Services	444	478
Technology	505	609
Transportation	38	47
Utilities	30	69
Total	1798	2189
Majority class proportion (%)	28.1	27.8

Table 3
Class distribution for the CoRA cite data set.

Category	Size
Case-based	402
Genetic algorithms	551
Neural networks	1064
Probabilistic methods	529
Reinforcement learning	335
Rule learning	230
Theory	472
Total	3583
Majority class proportion (%)	29.70

WebKB: WebKB consists of sets of web pages gathered from four computer science departments (one for each university), with each page manually labeled into six categories: course, department, faculty, project, staff, and student [6]. Two pages are linked by co-citation (if x links to z and y links to z, then x and y are co-citing z). The composition of the data sets is shown in Table 4.

Classification methods: The standard kernel-based classifiers compared in this experiment are based on (1) the SoP covariance (SoP) kernel introduced in [9] (see Section 3.2, Eq. (3)), (2) the normalized random walk with restart (NRWR) kernel [7], a normalized version of the random walk with restart [10,11] (see Section 3.3, Eq. (16)). The approximate classification methods proposed in this work are (3) the approximate sum-over-paths classifier (aSoP, see Section 3.2), based on the SoP covariance kernel, (4) the bounded normalized random walk with restart classifier (bNRWR, see Section 3.3) and (5) the biased $\mathcal{D}-$walk (b$\mathcal{D}$WALK, see Section 4). Moreover, as a baseline, we report the results obtained by using (6) the normalized, regularized, Laplacian (NRL) kernel, $(\mathbf{I}+\alpha\tilde{\mathbf{L}})^{-1}$, where $\tilde{\mathbf{L}}=\mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$ is the normalized Laplacian matrix [2] that achieved a competitive performance in [9], (7) the approximate normalized, regularized, Laplacian (aNRL) kernel which is the approximate counterpart of the NRL directly computing sum of the similarities by solving iteratively (up to τ steps) the system of linear equation, (8) the hitting time (hit Time) $h(k|\{i\in c\})$ (also called the average first-passage time) which in this application measures the average time (i.e. average steps) a random walker starting from any node i of a specified class c will take to reach a unlabeled node k for the first time [31]. The hitting time for each possible diffusion class has been approximated by solving iteratively a system of linear equations until convergence. Afterwards, we assign each node k to the class for which its hitting time is a minimum ($\text{argmin}_{c\in\mathcal{L}}h(k|\{i\in c\})$). Finally, the results of a simple (9) k-nearest-neighbor (KNN) are also reported. Our implementation of the KNN consists in taking all neighbors at maximum k steps of the considered node. An unlabeled node will be labeled with the tag that is most represented in the set of nodes located at maximum k steps (where duplicates have been removed). Each vote is weighted by the similarity in terms of number of steps ($1/k$) with the node of interest.

Notice that for all the bounded methods (bNRWR, b$\mathcal{D}$WALK, KNN), the maximum walk length τ is tuned during cross-validation. On the other hand, the approximate methods (aSoP, aNRL, hit Time) are iterated until convergence (RMSE $<$ 1.0e$-$04).

Remember that for all the kernel-based methods, the class label is obtained by computing the sum of similarities between the node of interest and the labeled nodes, as detailed in Section 3.1.

Experimental methodology: The classification accuracy will be reported for several labeling rates (5%, 10%, 20%, 35%, 50%, 65%, 80% and 95%), i.e. proportions of nodes for which the label is known. The labels of remaining nodes are used as test data. For each considered labeling rate, 10 random node label deletions (test sets) were

Table 4
Class distribution for the four WebKB cocite data sets.

Category	Size			
	Cornell	Texas	Washington	Wisconsin
Course	54	51	170	83
Department	25	36	20	37
Faculty	62	50	44	37
Project	54	28	39	25
Staff	6	6	10	11
Student	145	163	151	155
Total	346	334	434	348
Majority class proportion (%)	41.9	48.8	39.2	44.5

performed (10 runs), on which performances are averaged. For each unlabeled node, the various classifiers predict the most suitable category according to the procedures described previously. During each run, a 10-fold nested cross-validation is performed. The external folds are obtained by 10 successive rotations of the nodes and the performance of one run is averaged on these 10 folds. For each fold of the external cross-validation, a 5-fold internal cross-validation is performed on the remaining labeled nodes in order to tune the hyper-parameters of each classifier (i.e. the parameter θ for SoP, aSoP and bDWALK, the parameters α for NRWR, bNRWR, NRL and aNRL, the walk length τ for bDWALK, bNRWR and aNRL, the parameter k of KNN). For each method and each labeling rate, we report the average classification rate averaged on the 10 runs.

Results and discussion: Comparative results for each method on the eight different data sets are reported in Figs. 2(a), (b), 3(a) and (b). Clearly, whatever the labeling rate considered, the approximate methods achieve almost the same performance as their exact kernel-based counterpart on all data sets (i.e. there is no significative difference in terms of accuracy according to a sign test for a labeling rate of 10% with a p-value of 0.01). By analyzing the statistical significance of the results (Table 5), we observe that the best methods overall are the NRWR (and its bounded counterpart), the biased $\mathcal{D}-$walk, and the NRL (and its approximate counterpart) since they range among the top methods on all the benchmarked data sets. The SoP approach obtains good results in general, but it achieves sometimes least competitive results for a low labeling rate. This is the case on the two industries data set (Fig. 3(a)) and on the washington-cocite data set (Fig. 2(a)). Finally the hitting time approach results seems to be least competitive on these data sets except for the texas-cocite.

5.2. Second experiment: application to large-scale networks

In this second part of the experiments, we address the same task of classification of unlabeled nodes in partially labeled graphs, but this time on two large-scale data sets. The goal is to compare the results obtained by the three new algorithms introduced in this paper. The first data set considered is the US patents network which consists of more than 3 millions of nodes interconnected by 38 millions of links. The second data set consists of an amino acid sequence window associated to a target secondary structure [18]. This data set has already been used to investigate how far state-of-the-art semi-supervised methods can cope with large-scale application in [18].

Data sets: *The patent data set* introduced in this work is based on two publicly available databases: the NBER data set [40,41] and the PatStat database [42]. The resulting set is made of 3,416,966 U.S. patents granted between 1963 and 2002 and contains bibliographic data on each patent such as filing and grant dates, priority numbers (in case a U.S. patent was filed following preceding national or international applications), number of claims, and the list of countries of extension (countries other than the U.S. where the same patent was filed). In the economic literature, these

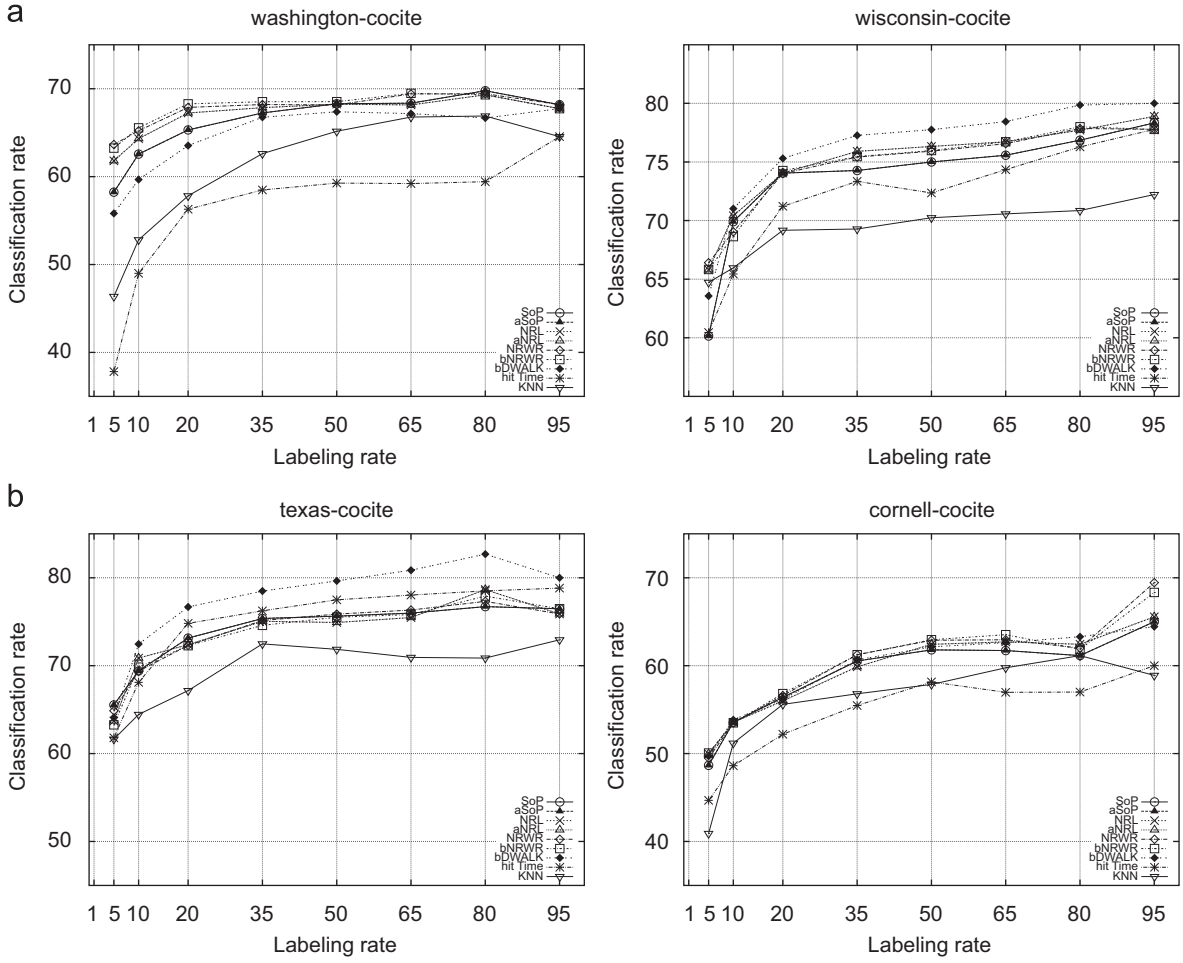

Fig. 2. Classification rates in percent, averaged over 10 runs, obtained on partially labeled graphs, for an increasing labeling rate of 5%, 10%, 20%, 35%, 50%, 65%, 80% and 95%. Results are reported for the SoP, aSoP, NRWR, bNRWR, NRL, aNRL, b$\mathcal{D}$WALK, hit Time and KNN classification methods. The graphs show the results obtained on the `washington`, `wisconsin`, `texas` and `cornell` `WebKB` data sets.

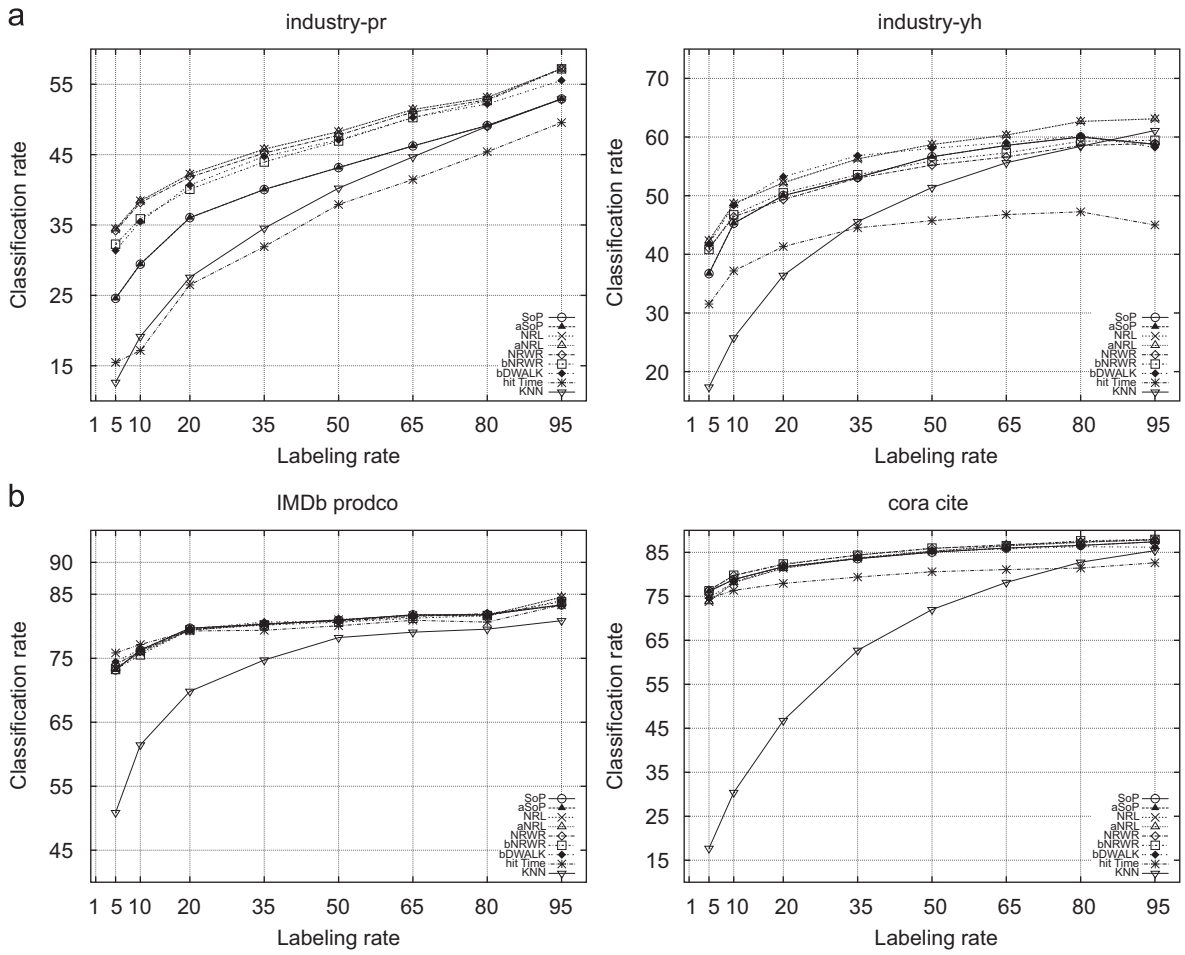

Fig. 3. Classification rates in percent, averaged over 10 runs, obtained on partially labeled graphs, for an increasing labeling rate of 5%, 10%, 20%, 35%, 50%, 65%, 80% and 95%. Results are reported for the SoP, aSoP, NRWR, bNRWR, NRL, aNRL, b$\mathcal{D}$WALK, hit Time and KNN classification methods. The graphs show the results obtained on the two `industries` data sets and on the `IMDb` and `CoRA` data sets [6].

Table 5
Compilation of statistical sign tests computed for the aSoP, bNRWR, b$\mathcal{D}$WALK and hit Time classification methods. A signed test has been performed on each of the eight medium-size data sets, based on the results of the 10 runs of the semi-supervised classification task, for a fixed labeling rate of 10%. Each entry of the table shows the number of times the row method is respectively significantly (i.e. p-value <0.01) better ($>$), the same ($=$) or worse ($<$) than the column method. For instance, bNRWR performs significantly better than hit Time on seven data sets.

Algorithm	aSoP	bNRWR	b$\mathcal{D}$WALK	hit Time	aNRL
aSoP	–	$=(7), <(1)$	$=(8)$	$>(5), =(3)$	$=(7), <(1)$
bNRWR	–	–	$>(1), =(6), <(1)$	$>(7), =(1)$	$=(7), <(1)$
b$\mathcal{D}$WALK	–	–	–	$>(6), =(2)$	$=(7), <(1)$
hit Time	–	–	–	–	$=(1), <(7)$

data are intensively used to measure innovation, the output of research and development activities, or to estimate the value of patents.

In addition, the data set includes the names and residence address of all inventors and assignees (i.e. companies) listed on each patent. These data can be used to analyze the geographical origin of patents and inventions. Companies listed as assignees can be matched with additional economic data such as turnover, profits, or stock value for the sake of economic analysis. The corpus is complemented with textual data including the English title and abstract of each patent. This information could be used in further work to label the patents based on both citations and text.

More importantly, patents are classified according to different U.S. and international classifications. The main U.S. class used in this work contains six broad industrial areas (chemicals, ICT, drugs and medical, electrical and electronic, mechanical, others), each of which contains up to nine subclasses. On the other hand, the international patent classification (IPC), maintained by the World Intellectual Property Organization (WIPO), provides a hierarchical representation of all technological fields. The first level contains eight classes (labeled from A to H: human necessities, transporting, chemistry and metallurgy, textiles and paper, fixed constructions, mechanical engineering, physics, electricity). The second level contains up to 20 subclasses (from 01 to 20). At the third level, the IPC class is made of four digits (from A01A to H10G). Additional levels are available up to 11 digits. It is to be noted however that patents are frequently assigned to several classes. In this case, one class is referred to as the main category. Although it is difficult to link such technological classes to industrial sectors, the economic literature has used them intensively to analyze innovation activities across industries (see [43]).

The graph structure in the data is made of *bibliographic references* between patents. In order to obtain a patent, an inventor must provide the list of references to patents and scholarly publications upon which the invention is based. This list is then completed by an examiner at the patent office in an attempt to delineate as clearly as possible the territory covered by each patent. What had been published earlier can indeed no longer be patented. The network of citations between patents can be seen as indicative

Table 6
Class distribution for the `U.S. patents` data set.

Category	Size	Proportion (%)
Chemicals	630,107	19.42
ICT	381,537	11.76
Drugs and medical	245,595	7.57
Electrical and electronic	575,369	17.73
Mechanical	724,022	22.31
Others	688,375	21.21
Total	3,245,005	100
Majority class proportion (%)	22.31	

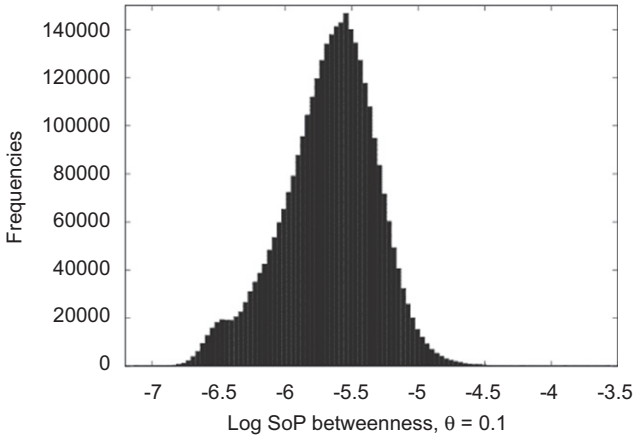

Fig. 4. Distribution of the logarithm of the SoP betweenness computed on the U.S. patents network. The θ parameter was fixed to 0.1.

of knowledge flows between companies, countries or industries. They have therefore been intensively used as indicators of knowledge spillovers in the economic literature (see e.g. [44–48]). In addition, since patent citations indicate downstream research activities, hence investments around the same technology, they suggest that an intensively cited patent denotes a particularly important or valuable invention. Citation counts have therefore also been used to produce value-weighted counts of patents (see [49,40,41]). One can naturally expect patents to preferably cite patents from the same technological class. An interesting topic, not covered in this paper, concerns clustering applied to patents (for more information, see the following survey on community detection [50]).

However, not all patents have citations to or from other patents, and many citations actually refer to pre-1963 patents for which most data are lacking. Excluding unconnected patents and records with missing values, the graph is left with 3,245,005 interconnected nodes and a total of 19,423,243 links.

The graph of patent citations is directed by nature, since patents can, in principle, only cite earlier publications. However, for the sake of the algorithms implemented in this paper, the matrix of links has been made symmetrical (as if each citing-cited pair of patents was a set of mutually citing patents). This matrix, in addition to the main class assigned to each patent in each classification scheme, is provided along with this paper.[3] Note that official patent numbers have been replaced by sequential numbers (from 1 to 3,245,005) to ease computations. The class distribution of nodes in the U.S. patents network is shown in Table 6.

Furthermore, the SoP betweenness has been computed according to Eq. (4). The histogram of the base-10 logarithm of this betweenness is displayed in Fig. 4, for a θ parameter equal to 0.1 according to previous tests (see [9]). The betweenness scores are located in the interval [7.3172e−08, 2.7947e−04]. For information, the patents that obtain the highest betweenness scores are, respectively, the U.S. patent 4340581 on *DSCG binding protein and process for preparing same* of July 20, 1982; the U.S. patent 4683202 on *Process for amplifying nucleic acid sequences* of July 28, 1987 and the U.S. patent 4723129 on *Bubble jet recording method and apparatus in which a heating element generates bubbles in a liquid flow path to project droplets* of February 2, 1988.

The secondary structure (*SecStr*) is a large data set that has been benchmarked in [18]. It consists of 83,679 amino acids around which an amino acids window $[-7,+7]$ is centered. The target is composed of two main classes: the $\alpha-$helical and $\beta-$sheet secondary structure form one class of 47,856 protein positions while the remaining coil structural motif positions form the other class. Hence, the main task is to predict the secondary structure of a given amino acid in a protein based on a sequence window. The class distribution of this data set is shown in Table 7.

Classification models: The tested classifiers are (1) the SoP approximation (aSoP), (2) the bounded normalized random walk with restart kernel (bNRWR), (3) the bounded $\mathcal{D}-$walk (b$\mathcal{D}$WALK) and (4) the hitting time (hit Time). As baseline, the results of (5) a simple k nearest neighbor (KNN), as well as (6) the approximate normalized regularized Laplacian (aNRL), are also reported. In this experiment, the number of maximum considered steps for the KNN was limited to $k=2$ because of computational issues.

Experimental methodology: The classification accuracy will be reported for increasing labeling rates (1%, 5%, 10%, 20%, 35%, 50%, 65%, 80% and 95%), i.e. proportions of nodes for which the label is known. The labels of remaining nodes are used as test data. For each considered labeling rate, 10 random node label deletions (test sets) were performed (10 runs), on which performances are averaged. For each run, a 10-fold cross-validation is performed on the remaining labeled nodes in order to tune the hyper-parameters of each classifier (see the first experiment in Section 5.1 for details). Thus, the performance on each run is assessed on the remaining unlabeled nodes with the hyper-parameter tuned during the 10-fold cross-validation.

Results and discussion: The results for each method and each labeling rate are reported in Fig. 5(a) for the U.S. patents and in Fig. 5(b) for the amino acids sequence data set.

For the U.S. patents (Fig. 5(a)), we observe that the results are very stable across the 10 runs. The bounded normalized random walk with restart kernel-based method (bNRWR) achieves the best performance (significant, $p<0.01$, at a labeling rate of 5% and 10%, according to a t-test performed on the 10 runs) for all labeling rates, very closely followed by the b$\mathcal{D}$WALK (except for a very low labeling rate, where the b$\mathcal{D}$WALK performance drops), and closely followed by the aSoP. It can be observed that these three proposed techniques are not very sensitive to the labeling rate. Indeed, the drop in performance from 95% to 5% labeling rate is lower than 6%,

Table 7
Class distribution for the `Secondary Structure` data set.

Category	Size	Proportion (%)
$\alpha-$helical and $\beta-$sheet	35,823	42.81
coil	47,856	57.19
Total	83,679	100
Majority class proportion (%)	57.19	

[3] This preprocessed data set in Matlab/Octave format is available from http://iridia.ulb.ac.be/~amantrac/pub/patents.mat.tar.gz.

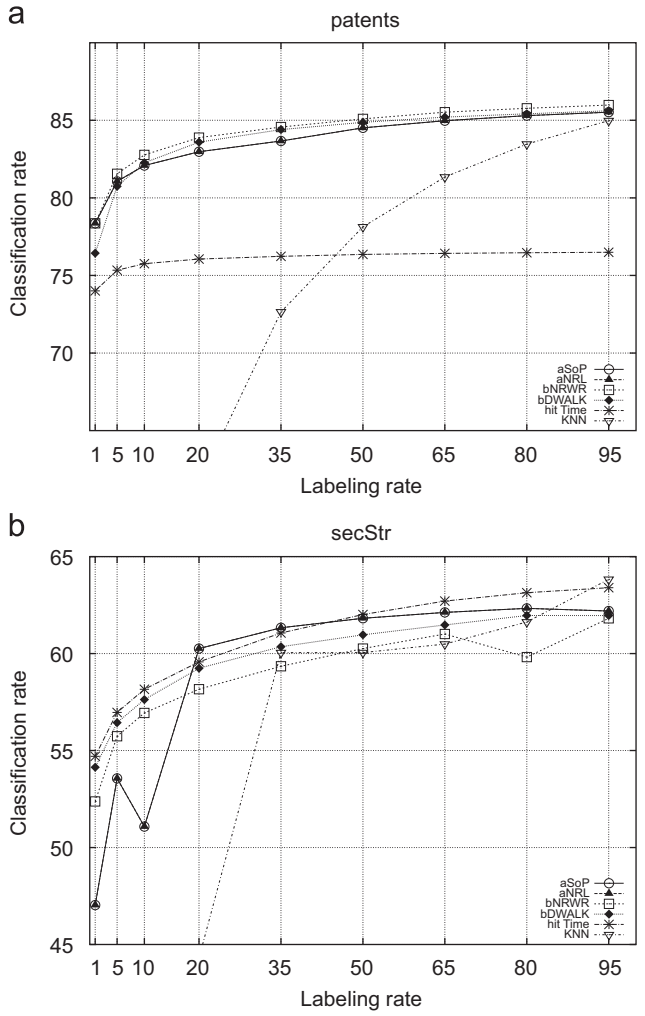

Fig. 5. Classification rates in percent, averaged over 10 runs, obtained on partially labeled graphs, for an increasing labeling rate of 1%, 5%, 10%, 20%, 35%, 50%, 65%, 80% and 95%. Results are reported for the aSoP (approximate sum-over-paths), the bNRWR (bounded normalized random walk with restart), the b$\mathcal{D}$WALK (biased $\mathcal{D}$−walk), the aNRL (approximate normalized regularized Laplacian), the hit Time and the KNN methods. The graphs show the results obtained on (a) the `U.S. patents` data set and (b) the `secStr` data set.

whatever the method considered. The hitting time approach is also stable but it underperforms in comparison with the three top leading methods. Actually, we already observed on medium size data sets that the hitting time performance was somewhat below the top methods (see experiments on medium size data set Section 5.1). In contrast, we observe a significant drop in performance for the aNRL and the KNN when the labeling rate decreases (for the sake of readability the results of the KNN and the aNRL are not shown for low labeling rates). Actually, both methods obtain a classification rate around 30% and around 8% for respectively a labeling rate of 5% and 1%.

On the SecStr data set (Fig. 5(a)), we observe a sudden large drop of the aNRL curve below 5% of labeling rate while the drop is more gentle on the U.S. patents network. The hitting time (hit Time) may be considered as the leading method on this data set, very closely followed by the b$\mathcal{D}$WALK and the bNRWR—all these three algorithms are rather robust with respect to changes in labeling rate (i.e. the drop in performance is lower than 10%). Actually, only the bNRWR and the b$\mathcal{D}$WALK achieved consistently very stable results on the `U.S. patents` graph as well as on the eight medium-size networks. Thus, according to our benchmarks, these two approaches seem to be quite robust. Finally, the aSoP results are

Table 8
Averaged accuracy drop obtained when labeling rate decreases from 95% to 5% on medium-size networks, and from 95% to 1% on large-scale networks. The accuracy, averaged over 10 runs, were obtained for medium-size networks on: `washington, wisconsin, texas, cornell WebKB, industries, IMDb, CoRA` data sets, and for large-scale network on: `U.S. patents citation network` and `secStr` data set. The results are reported for the aSoP, the bNRWR, the b$\mathcal{D}$WALK, the aNRL, the hit Time and the KNN.

Size	aSoP	aNRL	bNRWR	b$\mathcal{D}$WALK	hit Time	KNN
Medium-size	15.90	14.43	14.22	15.09	17.44	28.09
Large-scale	11.17	68.66	8.53	8.49	5.50	64.30

Table 9
Overview of cpu time in seconds needed for running an algorithm (and thus classifying all the unlabeled nodes), averaged over 10 runs, obtained on the `U.S. patents network` for labeling rates of 1%, 5%, 10%, 20%, 35%, 50%, 65%, 80% and 95%. Results are reported for the aSoP, the bNRWR, the b$\mathcal{D}$WALK and the aNRL. The cpu used is an Intel(R) Xeon(R) CPU E5335 at 2.00 GHz, with 4096 KB of cache size and 8 GB of RAM.

Algorithm	1%	5%	10%	20%	35%	50%	65%	80%	95%
aSoP	769	749	972	883	658	291	313	351	337
aNRL	45	15	17	25	46	77	134	179	246
bNRWR	41	42	31	82	118	178	261	380	505
b$\mathcal{D}$WALK	55	58	63	79	120	184	271	379	511

a bit disappointing on this data set since its performance curve drops more sharply than the leading algorithms for low labeling rates (as before, for the sake of readability the results of the KNN and the aNRL are not reported for low labeling rates).

An analysis of the stability of the algorithms on all benchmarked data sets (medium-size and large-scale) is provided in Table 8. It shows, for all the implemented methods, the averaged accuracy drop when the labeling rate decreases from 95% to 5% on medium-size graphs, and from 95% to 1% on large-scale networks.

Table 9 provides a comparison of the running time of all methods, averaged on 10 runs for the U.S. patents network. We observe that the classification task only takes a few minutes, whatever the method used. The aSoP method is significantly slower, but is still able to classify the whole graph in a few minutes.

6. Related work

Graph-based semi-supervised classification has been the subject of intensive research in recent years. A wide range of approaches have been developed in order to address the problem. Among them, we may cite random walks [51,29,3], graph mincuts [52], spectral methods [53–56], regularization frameworks [19,21,22,1,2], transductive and spectral SVMs [57]. For a comprehensive survey of semi-supervised classification, including graph-based approaches, see [4,5]. Some of these approaches tackle the problem by random-walk-based methods. However, it has been shown that hitting times and commute times approaches suffer from several problems; for example, they take too long paths (hence irrelevant) into account so that popular entities are intrinsically favored [33,34]. On the other hand, random-walk-based techniques often require to inverse a matrix in order to compute measures on walks that are potentially of infinite length.

These shortcomings led some authors to consider bounded (or truncated) walks [58,31,3]. Sarkar et al. [58,31], for instance, used a truncated commute-time approach and showed experimentally that the truncation boosts the results while providing a competitive computing time in a proximity search task on a large graph with 600 K nodes. In the same spirit, Callut et al. suggested to bound

walks for tackling graph-based semi-supervised problems [3]. Their approach offered a time complexity $\mathcal{O}(\tau|\mathcal{L}||E|)$, but no experimental results on large graphs were presented. In this paper, we propose precisely a generalization of the algorithm introduced by Callut et al. using the randomized shortest path framework [16,9,37]. In addition, we present experimental results on the large-scale U.S. patents data set and on a large-scale protein secondary structure data set.

Tong et al. suggested a method avoiding to inverse the complete matrix for computing the random walk with restart measure [59]. Their idea is to reduce the computing time by partitioning the input graph into smaller communities. Then, by applying a low rank approximation, they were able to approximate the random walk with restart. The approximated matrix obtained is sparse and hence can be kept into memory. Furthermore, they reduce the computing time by the number of communities initially found. Thanks to the Sherman–Morrison lemma, the precomputed inverse is updated on-the-fly for a new query. This method suffers from the fact that it adds an hyperparameter k, e.g. the number of communities, that depends on the network. Furthermore, the computing time is reduced by the factor k which is still untractable for large graphs with millions of nodes.

Moreover, Tong et al. recently developed a direction-aware proximity method based on the concept of escape probability [14,11]. They presented two methods to compute efficiently this proximity measure. The first one, FastAllDAP, allows to compute directly the proposed measure between all pairs of nodes by reducing the cost of solving a large number of linear systems to one matrix inversion. This method can only be applied on medium-size graphs (i.e. <10K nodes). The second method, FastOneDAP, may be applied to large graphs, but only computes one measure. In this case, the matrix inversion problem is approximated by a Taylor expansion of the concerned matrix and the complexity is reduced to $\mathcal{O}(\tau|E|)$. In this work, we avoid the direct computation of the proximity measure by taking advantage of the property of our targeted task, i.e. semi-supervised classification. Using the FastOneDAP directly for this task would require computing the measure $|V|$ times (i.e., for each node), so that its time complexity would be $\mathcal{O}(\tau|E||V|)$ (recall that the time complexity of our proposed method is $\mathcal{O}(\tau|E||\mathcal{L}|)$, since the number $|\mathcal{L}|$ of classes is usually much smaller than the number $|V|$ of nodes, we save an important amount of computing time).

Finally, Herbster et al. [60] proposed a technique for fast label prediction on a generic graph through the approximation of the graph with either a minimum spanning tree or a shortest path tree. Once the tree has been extracted, they are able to compute the pseudoinverse of the Laplacian matrix – a well-known kernel on a graph [25,24,61] – efficiently. The fast computation of the pseudoinverse enables to address prediction problems on large graphs.

7. Conclusion

This work investigated several approaches for tackling semi-supervised problems as well as betweenness computation on large, sparse, graphs. While this paper focuses on semi-supervised classification and betweenness computation, the same approaches (bounding or approximating random walks) could easily be applied in order to compute other graph measures, such as group degree centrality, closeness centrality, etc. [36]. This will undoubtedly be the subject of further work.

We will also, as future work, pursue the analysis of the U.S. patents data set introduced in this work. For instance, the correlation between various measures of importance of nodes (such as centrality/prestige) and econometric indicators of the value of a patent will be investigated. We will also study the impact of using additional patent information such as the abstract, various econometric indexes, etc., on the results of semi-supervised classification. Another interesting issue in this respect is how to combine the information provided by the graph and the node features in a clever, preferably optimal, way.

Yet another interesting topic to be investigated is the clustering of large-scale graphs—we are currently working on extensions of graph kernel clustering applicable to large-scale graphs [62,63]. For a survey of community detection refer to [50].

Still another interesting application of the techniques presented in this paper is collaborative recommendation. Indeed, the same methods could be used for large-scale recommendation (like the Netflix challenge) using graph kernels in the same way as [25].

Acknowledgments

We thank the anonymous reviewers for their interesting remarks and suggestions that allowed to improve significantly the quality of the paper.

Part of this work has been funded by projects with the "Région wallonne" and the Belgian "Politique Scientifique Fédérale". We thank these institutions for giving us the opportunity to conduct both fundamental and applied research.

References

[1] D. Zhou, O. Bousquet, T. Lal, J. Weston, B. Scholkopf, Learning with local and global consistency, in: Proceedings of the Neural Information Processing Systems Conference (NIPS2003), 2003, pp. 237–244.
[2] D. Zhou, J. Huang, B. Scholkopf, Learning from labeled and unlabeled data on a directed graph, in: Proceedings of the 22nd International Conference on Machine Learning, 2005, pp. 1041–1048.
[3] J. Callut, K. Francoisse, M. Saerens, P. Dupont, Semi-supervised classification from discriminative random walks, Proceedings of the European Conference on Machine Learning (ECML 2008), Lecture Notes in Artificial Intelligence, vol. 5211, 2008, pp. 162–177.
[4] X. Zhu, Semi-supervised learning literature survey, Computer Sciences, University of Wisconsin-Madison, 2008.
[5] X. Zhu, A. Goldberg, Introduction to Semi-supervised Learning, Morgan & Claypool Publishers, 2009.
[6] S.A. Macskassy, F. Provost, Classification in networked data: a toolkit and a univariate case study, Journal of Machine Learning Research 8 (2007) 935–983.
[7] F. Fouss, L. Yen, A. Pirotte, M. Saerens, An experimental investigation of seven kernels on two collaborative recommendation tasks, submitted for publication.
[8] P. Chebotarev, E. Shamis, The forest metric for graph vertices, Electronic Notes in Discrete Mathematics 11 (2002) 98–107.
[9] A. Mantrach, L. Yen, J. Callut, K. Francoise, M. Shimbo, M. Saerens, The sum-over-paths covariance kernel: a novel covariance between nodes of a directed graph, IEEE Transactions on Pattern Analysis and Machine Intelligence 32 (6) (2010) 1112–1126.
[10] J.-Y. Pan, H.-J. Yang, C. Faloutsos, P. Duygulu, Automatic multimedia cross-modal correlation discovery, in: Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2004, pp. 653–658.
[11] H. Tong, Y. Koren, C. Faloutsos, Fast direction-aware proximity for graph mining, in: Proceedings of the ACM SIGKDD international conference on Knowledge Discovery and Data Mining (KDD), 2007, pp. 747–756.
[12] F. Fouss, L. Yen, A. Pirotte, M. Saerens, An experimental investigation of graph kernels on a collaborative recommendation task, in: Proceedings of the 6th International Conference on Data Mining (ICDM 2006), 2006, pp. 863–868.
[13] F. Bach, M. Jordan, Predictive low-rank decomposition for kernel methods, in: Proceedings of the 22nd International Conference on Machine Learning, vol. 22, 2005, pp. 23–40.
[14] H. Tong, C. Faloutsos, J.-Y. Pan, Random walk with restart: fast solutions and applications, Knowledge and Information Systems 14 (3) (2008) 327–346.
[15] T. Ito, M. Shimbo, T. Kudo, Y. Matsumoto, Application of kernels to link analysis, in: Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2005, pp. 586–592.
[16] M. Saerens, Y. Achbany, F. Fouss, L. Yen, Randomized shortest-path problems: two related models, Neural Computation 21 (8) (2009) 2363–2404.
[17] L. Freeman, A set of measures of centrality based on betweenness, Sociometry 40 (1) (1977) 35–41.
[18] O. Chapelle, B. Scholkopf, A. Zien, Semi-Supervised Learning, MIT Press, 2006.
[19] M. Belkin, I. Matveeva, P. Niyogi, Tikhonov regularization and semi-supervised learning on large graphs, in: Proceedings of the IEEE International

Conference on Acoustics, Speech, and Signal Processing (ICASSP2004), 2004, pp. 1000–1003.
[20] J. Wang, T. Jebara, S.-F. Chang, Graph transduction via alternating minimization, in: Proceedings of the 25th International Conference on Machine Learning 2008, pp. 1144–1151.
[21] J. Wang, F. Wang, C. Zhang, H. Shen, L. Quan, Linear neighborhood propagation and its applications, IEEE Transactions on Pattern Analysis and Machine Intelligence 31 (9) (2009) 1600–1615.
[22] Y. Yajima, T.-F. Kuo, Efficient formulations for 1-svm and their application to recommendation tasks, Journal of Computers 1 (3) (2006) 27–34.
[23] J. Shawe-Taylor, N. Cristianini, Kernel Methods for Pattern Analysis, Cambridge University Press, 2004.
[24] M. Saerens, F. Fouss, L. Yen, P. Dupont, The principal components analysis of a graph, Proceedings of the 15th European Conference on Machine Learning (ECML 2004), Lecture Notes in Artificial Intelligence, vol. 3201, Springer-Verlag, Berlin, 2004, pp. 371–383.
[25] F. Fouss, A. Pirotte, J.-M. Renders, M. Saerens, Random-walk computation of similarities between nodes of a graph, with application to collaborative recommendation, IEEE Transactions on Knowledge and Data Engineering 19 (3) (2007) 355–369.
[26] D.J. Klein, M. Randic, Resistance distance, Journal of Mathematical Chemistry 12 (1993) 81–95.
[27] D. Rao, D. Yarowsky, C. Callison-Burch, Affinity measures based on the graph Laplacian, in: Proceedings of the 3rd Textgraphs Workshop on Graph-Based Algorithms for Natural Language Processing, Association for Computational Linguistics, 2008, pp. 41–48.
[28] X. Zhu, Z. Ghahramani, J. Lafferty, Semi-supervised learning using gaussian fields and harmonic functions, in: Proceedings of the 20th International Conference on Machine Learning (ICML-2003), 2003, pp. 912–919.
[29] M. Szummer, T. Jaakkola, Partially labeled classification with Markov random walks, in: NIPS, 2001, pp. 945–952.
[30] G.H. Golub, C.F.V. Loan, Matrix Computations, third ed., Johns Hopkins University Press, 1996.
[31] P. Sarkar, A. Moore, A tractable approach to finding closest truncated-commute-time neighbors in large graphs, in: Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI), 2007, pp. 335–343.
[32] L. Lu, T. Zhou, Link Prediction in Complex Networks: A Survey, submitted for publication ArXiv e-prints arXiv:1010.0725.
[33] D. Liben-Nowell, J. Kleinberg, The link-prediction problem for social networks, Journal of the American Society for Information Science and Technology 58 (7) (2007) 1019–1031.
[34] M. Brand, A random walks perspective on maximizing satisfaction and profit, in: Proceedings of the 2005 SIAM International Conference on Data Mining, 2005.
[35] L. Page, S. Brin, R. Motwani, T. Winograd, The pagerank citation ranking: bringing order to the web, Technical Report 1999-0120, Computer Science Department, Stanford University, 1999.
[36] S. Wasserman, K. Faust, Social Network Analysis: Methods and Applications, Cambridge University Press, 1994.
[37] L. Yen, A. Mantrach, M. Shimbo, M. Saerens, A family of dissimilarity measures between nodes generalizing both the shortest-path and the commute-time distances, in: Proceedings of the 14th SIGKDD International Conference on Knowledge Discovery and Data Mining, 2008, pp. 785–793.
[38] L. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, Proceedings of the IEEE 77 (1989) 257–285.
[39] F. Jelinek, Statistical Methods for Speech Recognition, The MIT Press, 1997.
[40] B. Hall, A. Jaffe, M. Trajtenberg, The NBER patent citation data file: lessons, insights and methodological tools, NBER Working Paper 8498, 2001.
[41] B. Hall, A. Jaffe, M. Trajtenberg, Market value and patent citations, RAND Journal of Economics 36 (2005) 16–38.
[42] E.P. Office, PatStat, Statistical Patent Database (PatStat), 2007.
[43] N. van Zeebroeck, B. van Pottelsberghe de la Potterie, W. Han, Issues in measuring the degree of technological specialization with patent data, Scientometrics 66 (2006) 481–492.
[44] A. Jaff, M. Trajtenberg, R. Henderson, Geographic localization of knowledge spillovers as evidenced by patent citations, The Quarterly Journal of Economics 108 (3) (1993) 577–598.
[45] R. Cowan, D. Foray, he economics of codification and the diffusion of knowledge, Industrial and Corporate Change 6(3) (1997).
[46] F. Lichtenberg, B. van Pottelsberg de la Potterie, International r&d spillovers: a comment, European Economic Review 42 (8) (1996) 1483–1491.
[47] R. Cowan, D. Foray, The explicit economics of knowledge codification and tacitness, Industrial and Corporate Change 9(2) (2000).
[48] B. Cassiman, R. Veugelers, R. Henderson, R&D cooperation and spillovers: some empirical evidence, The American Economic Review 92 (4) (2002) 1169–1184.
[49] M. Trajtenberg, A penny for your quotes: patent citations and the value of innovations, The RAND Journal of Economics 21 (1990) 171–187.
[50] S. Fortunato, Community detection in graphs, Physics Reports 486 (3–5) (2010) 75–174.
[51] D. Zhou, B. Scholkopf, Learning from labeled and unlabeled data using random walks, in: Proceedings of the 26th DAGM Symposium, 2004, pp. 237–244.
[52] A. Blum, S. Chawla, Learning from labeled and unlabeled data using graph mincuts, in: ICML, 2001, pp. 19–26.
[53] O. Chapelle, J. Weston, B. Schölkopf, Cluster kernels for semi-supervised learning, in: NIPS, 2002, pp. 585–592.
[54] A.J. Smola, R. Kondor, Kernels and regularization on graphs, in: M. Warmuth, B. Schölkopf (Eds.), Proceedings of the Conference on Learning Theory COLT, 2003, pp. 144–158.
[55] R.I. Kondor, J. Lafferty, Diffusion kernels on graphs and other discrete structures, in: Proceedings of the 19th International Conference on Machine Learning, 2002, pp. 315–322.
[56] A. Kapoor, Y.A. Qi, H. Ahn, R.W. Picard, Hyperparameter and kernel learning for graph based semi-supervised classification, in: NIPS, 2005, pp. 627–634.
[57] T. Joachims, Transductive learning via spectral graph partitioning, in: ICML, 2003, pp. 290–297.
[58] P. Sarkar, A.W. Moore, A. Prakash, Fast incremental proximity search in large graphs, in: ICML, 2008, pp. 896–903.
[59] H. Tong, C. Faloutsos, J.-Y. Pan, Fast random walk with restart and its applications, in: Proceedings of 6th IEEE International Conference on Data Mining, 2006, pp. 613–622.
[60] M. Herbster, M. Pontil, S. Rojas-Galeano, Fast prediction on a tree, in: Proceedings of the 22th Neural Information Processing Conference NIPS 2008, 2008, pp. 657–664.
[61] H. Hirai, K. Murota, M. Rikitoku, Svm kernel by electric network, Pacific Journal of Optimization 1 (2005) 509–526.
[62] L. Yen, F. Fouss, C. Decaestecker, P. Francq, M. Saerens, Graph nodes clustering based on the commute-time kernel, in: Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2007, pp. 1037–1045.
[63] L. Yen, F. Fouss, C. Decaestecker, P. Francq, M. Saerens, Graph nodes clustering with the sigmoid commute-time kernel: a comparative study, Data and Knowledge Engineering 68 (3) (2009) 338–361.