PAPER • OPEN ACCESS

Mercator: uncovering faithful hyperbolic embeddings of complex networks

To cite this article: Guillermo García-Pérez *et al* 2019 *New J. Phys.* **21** 123033

New Journal of Physics
The open access journal at the forefront of physics

Deutsche Physikalische Gesellschaft DPG
IOP Institute of Physics

Published in partnership with: Deutsche Physikalische Gesellschaft and the Institute of Physics

OPEN ACCESS

RECEIVED
30 April 2019

REVISED
5 November 2019

ACCEPTED FOR PUBLICATION
14 November 2019

PUBLISHED
16 December 2019

PAPER

Mercator: uncovering faithful hyperbolic embeddings of complex networks

Guillermo García-Pérez[1,2,8], **Antoine Allard**[3,4,8], **M Ángeles Serrano**[5,6,7] **and Marián Boguñá**[5,6]

[1] QTF Centre of Excellence, Turku Centre for Quantum Physics, Department of Physics and Astronomy, University of Turku, FI-20014 Turun Yliopisto, Finland
[2] Complex Systems Research Group, Department of Mathematics and Statistics, University of Turku, FI-20014 Turun Yliopisto, Finland
[3] Département de physique, de génie physique et d'optique, Université Laval, Québec (Québec), G1V 0A6, Canada
[4] Centre de modélisation mathématique, Université Laval, Québec (Québec), G1V 0A6, Canada
[5] Departament de Física de la Matèria Condensada, Universitat de Barcelona, Martí i Franquès 1, E-08028 Barcelona, Spain
[6] Universitat de Barcelona Institute of Complex Systems (UBICS), Universitat de Barcelona, Barcelona, Spain
[7] Institució Catalana de Recerca i Estudis Avançats (ICREA), Passeig Lluís Companys 23, E-08010 Barcelona, Spain
[8] Both authors contributed equally to this work.

E-mail: marian.boguna@ub.edu

Keywords: complex networks, network geometry, statistical inference

Abstract

We introduce Mercator, a reliable embedding method to map real complex networks into their hyperbolic latent geometry. The method assumes that the structure of networks is well described by the popularity $\times$ similarity $\mathbb{S}^1/\mathbb{H}^2$ static geometric network model, which can accommodate arbitrary degree distributions and reproduces many pivotal properties of real networks, including self-similarity patterns. The algorithm mixes machine learning and maximum likelihood (ML) approaches to infer the coordinates of the nodes in the underlying hyperbolic disk with the best matching between the observed network topology and the geometric model. In its fast mode, Mercator uses a model-adjusted machine learning technique performing dimensional reduction to produce a fast and accurate map, whose quality already outperforms other embedding algorithms in the literature. In the refined Mercator mode, the fast mode embedding result is taken as an initial condition in a ML estimation, which significantly improves the quality of the final embedding. Apart from its accuracy as an embedding tool, Mercator has the clear advantage of systematically inferring not only node orderings, or angular positions, but also the hidden degrees and global model parameters, and has the ability to embed networks with arbitrary degree distributions. Overall, our results suggest that mixing machine learning and ML techniques in a model-dependent framework can boost the meaningful mapping of complex networks.

1. Introduction

The main hypothesis of network geometry states that the architecture of real complex networks has a geometric origin [1–3]. The nodes of the complex network can be characterized by their positions in an underlying metric space so that the observable network topology—abstracting their patterns of interactions—is then a reflection of distances in this space. This simple idea led to the development of a very general framework able to explain the most ubiquitous topological properties of real networks [1, 2], namely, degree heterogeneity, the small-world property, and high levels of clustering. Network geometry is also able to explain in a very natural way other non-trivial properties, like self-similarity [1, 4] and community structure [5–7], their navigability properties [8–10], and is the basis for the definition of a renormalization group in complex networks [11]. The geometric approach has also been successfully extended to weighted networks [12] and multiplexes [13, 14].

Beyond being a formal theoretical framework to explain the topology of real networks, network geometry can be used to develop practical applications for real systems, including routing of information in the Internet [10], community detection [10, 15, 16], prediction of missing links [3, 17, 18], a precise definition of hierarchy in

networks [16], and downscaled network replicas [11], to name a few. However, applications require faithful embeddings of real-world networks into the hidden metric space using only the information contained in their topology. Several algorithms have been proposed to solve this problem, most of which either use maximum likelihood (ML) estimation techniques [10, 19–22], machine learning [23–25], or a combination of both [25, 26].

ML techniques assume that the network under study has been produced by a given model—a geometric one—and finds the values of its parameters that maximize the probability for the model to generate the observed topology. This technique requires finding the coordinates of every node in the latent geometry that maximize the likelihood function: a task that, in general, is NP-hard and consequently must rely on heuristics to obtain a reasonable approximate solution. ML methods are therefore generally slow, and their accuracy depends strongly on the chosen heuristic as well as on the quality of the underlying theoretical model.

In contrast, machine learning techniques are fast and model independent, so they can be used to find embeddings of large networks. A promising and accurate method is based on Laplacian eigenmaps (LE) [23, 24], a method originally designed to find dimensional reductions of a set of points embedded in $\mathbb{R}^n$ to an arbitrary dimensional space $\mathbb{R}^m$ with $m < n$ [27]. The LE method requires the definition of Euclidean distances between nodes in $\mathbb{R}^n$, but since no information is available about the 'real Euclidean' distances between connected pairs of nodes in networks, the use of heuristic arguments is necessary to estimate these distances [24]. A more fundamental problem with machine learning methods is that the embeddings are performed on Euclidean spaces. However, as shown in [2, 3], the geometry of real complex networks is better described by hyperbolic rather than Euclidean geometry, where angular coordinates on a circle are a proxy for the similarity between nodes, and their radial coordinates account for their popularity, which is typically measured by their degrees [1]. Machine learning methods are only able to infer the angular coordinates corresponding to the similarity sub-space while radial coordinates have to be inferred using some geometric model. Hence, these methods end up being model dependent as well.

Consequently, both types of methods are very sensitive to the model used to describe the network. The approaches introduced in references [19, 20, 23–25], which are based on the popularity $\times$ similarity optimization (PSO) model described in [3], which uses a simple mechanism to explain the emergence of an effective hyperbolic geometry in growing networks. However, this model can only generate pure power-law degree distributions $P(k) \sim k^{-\gamma}$ with $\gamma > 2$, whereas the degree distribution in many real networks shows important deviations from such pure power laws. Moreover, the model does not spontaneously generate the nested hierarchy of self-similar subgraphs with increasing average degree, as observed in real systems [1, 4].

In this paper, we introduce Mercator, a ready-to-use C++ code[9] that mixes the best of the ML and machine learning approaches. The mixing of the two techniques was explored in [26] using the PSO model to maximize the likelihood function. Instead, we use the static version of the same type of popularity $\times$ similarity geometric models, the $\mathbb{S}^1/\mathbb{H}^2$ model [1, 2], that can accommodate arbitrary degree distributions and can reproduce the self-similarity patterns observed in real networks. The first step in Mercator is to apply a LE approach, as in [24], but using the $\mathbb{S}^1/\mathbb{H}^2$ model instead of the PSO to infer the weights of the Laplacian matrix. Doing so yields a first (and fast) embedding method that already outperforms the one of [24]. The resulting embedding uses only information about pairs of connected neighbors, and can be further improved by using it as a starting point in a ML optimization—based again on the $\mathbb{S}^1/\mathbb{H}^2$ model—that uses information from both connected and not-connected pairs of nodes. The final result is the most accurate embedding method currently available in the literature. Yet, the final complexity of the method is $\mathcal{O}(N^2)$ for sparse networks with N nodes, which makes it competitive for real applications.

2. Methodological background

2.1. The $\mathbb{S}^1/\mathbb{H}^2$ model

The $\mathbb{S}^1$ model is the simplest among the class of geometric models [1]. The similarity space is a one dimensional sphere—a circle of radius R—where N nodes are distributed with a fixed density, set to one without loss of generality, so that $N = 2\pi R$[10]. Each node is also given a hidden variable $\kappa \in [\kappa_0, \infty)$ proportional to its expected degree. In general, κ and the angular position θ can be correlated and distributed according to an arbitrary distribution $\rho(\kappa, \theta)$. In such case, the model is able to generate community structure [5–7] and can reproduce different degree–degree correlation patterns and clustering spectra.

Once all nodes are assigned a tuple (κ, θ), each pair of nodes is connected with probability

$$p_{ij} = \frac{1}{1 + \left(\frac{d_{ij}}{\mu\kappa_i\kappa_j}\right)^{\beta}}, \tag{1}$$

[9] The code will be available at https://github.com/networkgeometry/mercator upon publication.

[10] Notice that in thermodynamic limit the curvature of the circle vanishes and the model is *effectively* defined on $\mathbb{R}^1$.

where $d_{ij} = R\Delta\theta_{ij}$ is the arc length of the circle between nodes i and j separated by an angular distance $\Delta\theta_{ij}$. Parameters μ and β control the average degree and the clustering coefficient, respectively. The model can be defined using any connection probability as long as it is an integrable function $p(\chi)$ with $\chi = \frac{d}{\mu\kappa\kappa'}$ [1]. The particular functional form that we chose here (the Fermi distribution) is the one that defines maximally random ensembles of geometric graphs that are simultaneously clustered, small-world, and with heterogeneous degree distributions.

If nodes are uniformly distributed over the circle, we have $\rho(\kappa, \theta) = \rho(\kappa)/2\pi$. In this case, the choice $\mu = \frac{\beta}{2\pi\langle k\rangle}\sin\frac{\pi}{\beta}$ guarantees that, in the thermodynamic limit, the expected degree of a node with hidden variable κ is $\bar{k}(\kappa) = \kappa$ and the network average degree is $\langle k\rangle = \langle\kappa\rangle$. It is therefore possible to associate unambiguously the hidden variable κ with the node degree. For finite systems, however, the values of the hidden variables κ must be evaluated numerically. It is also important to notice that the parameter μ is, in fact, superfluous since it can be absorbed in the definition of κ; κ would then be proportional, but not exactly equal, to the expected degree. As a result, the embedding task only requires the estimation of $2N + 1$ parameters: the hidden variables (κ_i, θ_i), $i = 1, \cdots, N$, and the parameter β.

2.1.1. Hyperbolic representation. The $\mathbb{H}^2$ model

Quite remarkably, the $\mathbb{S}^1$ model can be expressed as a purely geometric model in the hyperbolic plane. By mapping the expected degree of each node κ_i to a radial coordinate as

$$r_i = \hat{R} - 2\ln\frac{\kappa_i}{\kappa_0}, \tag{2}$$

with $\hat{R} \equiv 2\ln\frac{N}{\mu\pi\kappa_0^2}$, the connection probability becomes

$$p_{ij} = \frac{1}{1 + \mathrm{e}^{\frac{\beta}{2}(x_{ij} - \hat{R})}}, \tag{3}$$

where

$$x_{ij} = r_i + r_j + 2\ln\frac{\Delta\theta_{ij}}{2} \tag{4}$$

is a good approximation of the hyperbolic distance between two nodes separated by an angular distance $\Delta\theta_{ij}$ and with radial coordinates r_i and r_j [11]. The connection probability thus becomes a function of the hyperbolic distance alone, which turns the model into a purely geometric one and has important consequences for the global connectivity of the network. For instance, topological shortest paths closely follow geodesic curves in the hyperbolic plane, and can therefore be used to efficiently navigate the network [8, 10]. Furthermore, when the distribution of expected degrees follows a power law of exponent γ, the radial distribution in the hyperbolic plane is

$$\rho(r) = \alpha\frac{\sinh\alpha r}{\cosh\alpha\hat{R} - 1} \tag{5}$$

with $\gamma = 2\alpha + 1$ and $\alpha > 0$. Nodes are therefore homogeneously distributed in the hyperbolic plane for $\gamma = 3$ and are quasi-homogeneously distributed for other values of γ. In this paper, we use the $\mathbb{S}^1$ model for likelihood maximization, and its equivalent $\mathbb{H}^2$ version for visualization purposes.

2.2. Embedding techniques

Mercator exploits two different embedding techniques, based on ML and on LE, which are briefly outlined in this section.

2.2.1. Model-corrected LE

LE was originally designed as a method for dimensional reduction. Given a set of points $\{\mathbf{x}_i \in \mathbb{R}^n,\ i = 1,\cdots,N\}$ with the Euclidean metric, LE finds a mapping of these points $\{\mathbf{x}_i \mapsto \mathbf{y}_i \in \mathbb{R}^m\}$ with $m < n$ such that the loss function

$$\epsilon = \sum_{i,j}|\mathbf{y}_i - \mathbf{y}_j|^2\omega(|\mathbf{x}_i - \mathbf{x}_j|^2) \tag{6}$$

is minimized. Here, $|\mathbf{y}_i - \mathbf{y}_j|$ is the Euclidian distance between points i and j in $\mathbb{R}^m$ and $\omega(\,\cdot\,)$ is a decreasing function of the distance between the same pair of points in the original Euclidean space $\mathbb{R}^n$. Intuitively, placing pairs of points far apart in $\mathbb{R}^m$ if they were originally close in $\mathbb{R}^n$ increases the loss function equation (6). Minimizing ϵ should therefore yield a faithful dimensional reduction of the data.

[11] This approximation is reasonably accurate for pairs of nodes separated by $\Delta\theta_{ij} > \sqrt{\mathrm{e}^{-2r_i} + \mathrm{e}^{-2r_j}}$, the fraction of which converges to one in the thermodynamic limit.

In the case of network embedding, our aim is to find coordinates of nodes in $\mathbb{R}^2$ of a network whose structure can be modeled by the $\mathbb{S}^1$ model. To do so, the weight function $\omega(\cdot)$ is taken to be proportional to the adjacency matrix so that it is only different from zero if nodes i and j are connected. Yet, the weight associated to connected pairs of nodes is still assumed to be a decreasing function of their original Euclidean distance, which must somehow be estimated from the network structure. To do so, we leverage the $\mathbb{S}^1$ model and estimate the expected distance in $\mathbb{R}^2$ (the chord length) of a pair of nodes based on their degrees. The set of coordinates that minimize the loss function ϵ is the solution of a generalized eigenvalue problem with the Laplacian matrix, for which very fast algorithms exist if the network is sparse [28].

2.2.2. ML estimation

Given a real network with adjacency matrix $\{a_{ij}\}$, ML estimation finds the values of $\{\kappa_i, \theta_i\}$, $i = 1, \cdots, N$, that provide a good match between the $\mathbb{S}^1$ model and the observed network. The posterior probability, or likelihood, that a network specified by its adjacency matrix $\{a_{ij}\}$ is generated by the $\mathbb{S}^1$ model is

$$\mathcal{L}(\{a_{ij}\}|\mathbb{S}^1) = \int \cdots \int \mathcal{L}(\{a_{ij}\}, \{\kappa_i, \theta_i\}|\mathbb{S}^1) \prod_{i=1}^{N} \mathrm{d}\theta_i \mathrm{d}\kappa_i, \tag{7}$$

where the function $\mathcal{L}(\{a_{ij}\}, \{\kappa_i, \theta_i\}|\mathbb{S}^1)$ is the joint probability that the $\mathbb{S}^1$ model generates simultaneously the set of hidden variables $\{\kappa_i, \theta_i\}$ and the adjacency matrix $\{a_{ij}\}$. Using Bayes rule, we then compute the likelihood that the hidden variables $\{\kappa_i, \theta_i\}$ take particular values conditioned on the observed adjacency matrix $\{a_{ij}\}$

$$\mathcal{L}(\{\kappa_i, \theta_i\}|\{a_{ij}\}, \mathbb{S}^1) = \frac{\mathcal{L}(\{a_{ij}\}, \{\kappa_i, \theta_i\}|\mathbb{S}^1)}{\mathcal{L}(\{a_{ij}\}|\mathbb{S}^1)} = \frac{\mathrm{Prob}(\{\kappa_i, \theta_i\})\mathcal{L}(\{a_{ij}\}|\{\kappa_i, \theta_i\}, \mathbb{S}^1)}{\mathcal{L}(\{a_{ij}\}|\mathbb{S}^1)}, \tag{8}$$

where

$$\mathrm{Prob}(\{\kappa_i, \theta_i\}) = \prod_{i=1}^{N} \rho(\theta_i, \kappa_i) \tag{9}$$

is the prior probability density function of the hidden variables,

$$\mathcal{L}(\{a_{ij}\}|\{\kappa_i, \theta_i\}, \mathbb{S}^1) = \prod_{i<j} p_{ij}^{a_{ij}}(1 - p_{ij})^{1-a_{ij}}, \tag{10}$$

is the probability that the $\mathbb{S}^1$ model generates the adjacency matrix $\{a_{ij}\}$ conditioned on the hidden variables $\{\kappa_i, \theta_i\}$, and p_{ij} is the connection probability given by equation (1).

If we have information about the prior distribution of hidden variables, $\mathrm{Prob}(\{\kappa_i, \theta_i\})$, Bayesian estimators can be obtained by maximizing the likelihood in equation (8). However, in most cases, prior information is not available. Besides, by using improper priors we do not need to specify the form of the distribution of expected degrees $\rho(\kappa)$. This gives Mercator the flexibility to embed networks with arbitrary degree distributions. We then assume that $\mathrm{Prob}(\{\kappa_i, \theta_i\}) = cte$ and obtain the ML estimator as the set of values $\{\kappa_i^*, \theta_i^*\}$ that maximize equation (10) or, equivalently, its logarithm

$$\ln \mathcal{L}(\{a_{ij}\}|\{\kappa_i, \theta_i\}, \mathbb{S}^1) = \sum_{i<j} [a_{ij} \ln p_{ij} + (1 - a_{ij}) \ln (1 - p_{ij})]. \tag{11}$$

One of the advantages of using the $\mathbb{S}^1$ model versus the $\mathbb{H}^2$ is that the maximization with respect to the expected degrees κ can be performed semi-analytically. Indeed, the derivative of equation (11) with respect to the expected degree κ_l of node l is

$$\frac{\partial}{\partial \kappa_l} \ln \mathcal{L}(\{a_{ij}\}|\{\kappa_i, \theta_i\}, \mathbb{S}^1) = \frac{\beta}{\kappa_l} \sum_{i \neq l} (a_{il} - p_{il}), \tag{12}$$

where the second term on the right-hand side is the expected degree of node l, and the first term is its actual degree k_l. The value κ_l^* that maximizes the likelihood is therefore the solution of

$$k_l = \sum_{i \neq l} p_{il}. \tag{13}$$

The term on the right-hand side can be evaluated in the model assuming a homogeneous angular distribution of nodes on the circle. Notice that the numerical solution of this equation automatically takes into account finite size effects. We use this method to provide estimates of the expected degrees that are then used to maximize the likelihood function with respect to the angular coordinates, as explained in section A.6.

3. Mercator at a glance

We have now all the theoretical background to briefly describe Mercator; the full details are given at sections A.1–A.7. Given a network with adjacency matrix $\{a_{ij}\}$, we first measure its average degree $\langle k \rangle$, average

clustering coefficient $\bar{c}$, and all individual nodes' degrees $\{k_i, i = 1, \cdots, N\}$. Second, we estimate hidden degrees and parameters β and μ. Third, as in [24] we estimate the angular ordering of nodes using the model-corrected LE, and adjust the angles according to the expected angular separation between consecutive nodes given by the $\mathbb{S}^1$ model. This yields Mercator's fast mode version, which produces a first embedding. Fourth, the angular coordinates are refined using ML. Finally, hidden degrees are readjusted given the newly inferred angular positions. All the steps together conform Mercator refined mode. More precisely, Mercator executes the following steps.

3.1. Fast mode

1. Propose an approximate value for β and compute $\mu = \frac{\beta}{2\pi\langle k\rangle}\sin\frac{\pi}{\beta}$.
2. Using equations (1) and (13), adjust the hidden variables $\{\kappa_i\}$ such that the expected degree of each node in the $\mathbb{S}^1$ model matches the observed degree in the original network. This step assumes that nodes are homogeneously distributed and uses the values of β and μ from step 1. The initial guess is $\kappa_i = k_i$ (the degree of node i in the original network).
3. Using results from steps 1 and 2, evaluate the theoretical value of the average clustering coefficient of the network in the $\mathbb{S}^1$ model. If this value differs from the one measured for the original network, adjust the value of β and return to step 1. Otherwise, proceed to step 4.
4. For every connected pair of nodes with hidden variables κ_i and κ_j and original degrees k_i, $k_j > 1$,estimate their expected chord length in $\mathbb{R}^2$ as $d_{ij} = 2\sin\frac{\langle\Delta\theta_{ij}\rangle}{2}$, where $\langle\Delta\theta_{ij}\rangle$ is the expected angular separation between connected nodes i and j in the $\mathbb{S}^1$ model.
5. Construct a weighted Laplacian matrix $L_{ij} = D_{ij} - \omega_{ij}$, where D is the diagonal matrix with entries $D_{ii} = \sum_j \omega_{ij}$ and weights are given by $\omega_{ij} = a_{ij}\mathrm{e}^{-d_{ij}^2/t}$ with t being the variance of d_{ij}. Then solve the generalized eigenvalue problem

$$L\mathbf{v} = \lambda D\mathbf{v}.$$

   We note $\mathbf{v_1} = (v_{1,1}, v_{1,2},\cdots,v_{1,N})$ and $\mathbf{v_2} = (v_{2,1}, v_{2,2},\cdots,v_{2,N})$ the first two eigenvectors with the smallest nonzero eigenvalues.
6. Assign an angular position to each node i as $\theta_i = \mathrm{atan2}(v_{2,i}, v_{1,i})$.
7. Make a sorted list of the nodes based on their angular position $\{\theta_i\}$. Nodes of degree 1 that were excluded at step 4 are now reinserted in the sorted list randomly before or after their unique neighbor. Note that the angular coordinates computed at step 6 are only used to determine the order in which nodes are located angularly. Their precise angular coordinates are evaluated at the next step.
8. For each pair of consecutive nodes in the list, evaluate its angular separation as given by LE from step 7 and, similarly to [25], its expected angular separation using the $\mathbb{S}^1$ model conditioned on whether the two nodes are connected or not, their hidden variables κ_i and κ_j, and the fact that they are consecutive. Choose as the final gap the larger of the two. Finally, all gaps are normalized to sum up to 2π. This produces a set of angular coordinates for each node $\{\theta_i, i = 1, \cdots, N\}$.

These 8 steps summarize a fast and accurate embedding procedure that, as discussed in section 4, already outperforms current state-of-the-art methods. The next section explains how its accuracy can be further increased.

3.2. Refined mode

The embedding can be significantly improved with ML techniques using the embedding obtained with the fast mode of Mercator as initial conditions. This is due to the fact that ML uses the information contained both in the presence *and* absence of links in the network, whereas LE only relies on the information conveyed by the presence of links. The major drawback of ML is the complex configuration space that needs to be explored to find the optimal embedding. However, if the starting point of the exploration is good enough, the maximization of the likelihood function is easy and efficient. Thus, starting from the embedding obtained with the fast mode, we proceed as follows.

9. Extract the onion decomposition of the network [29] and sort nodes accordingly, starting with the node in the deepest layer. The onion decomposition is a generalization of the k-core decomposition that provides the *internal organization* of each k-shell. It therefore offers a more precise way to order nodes than based on

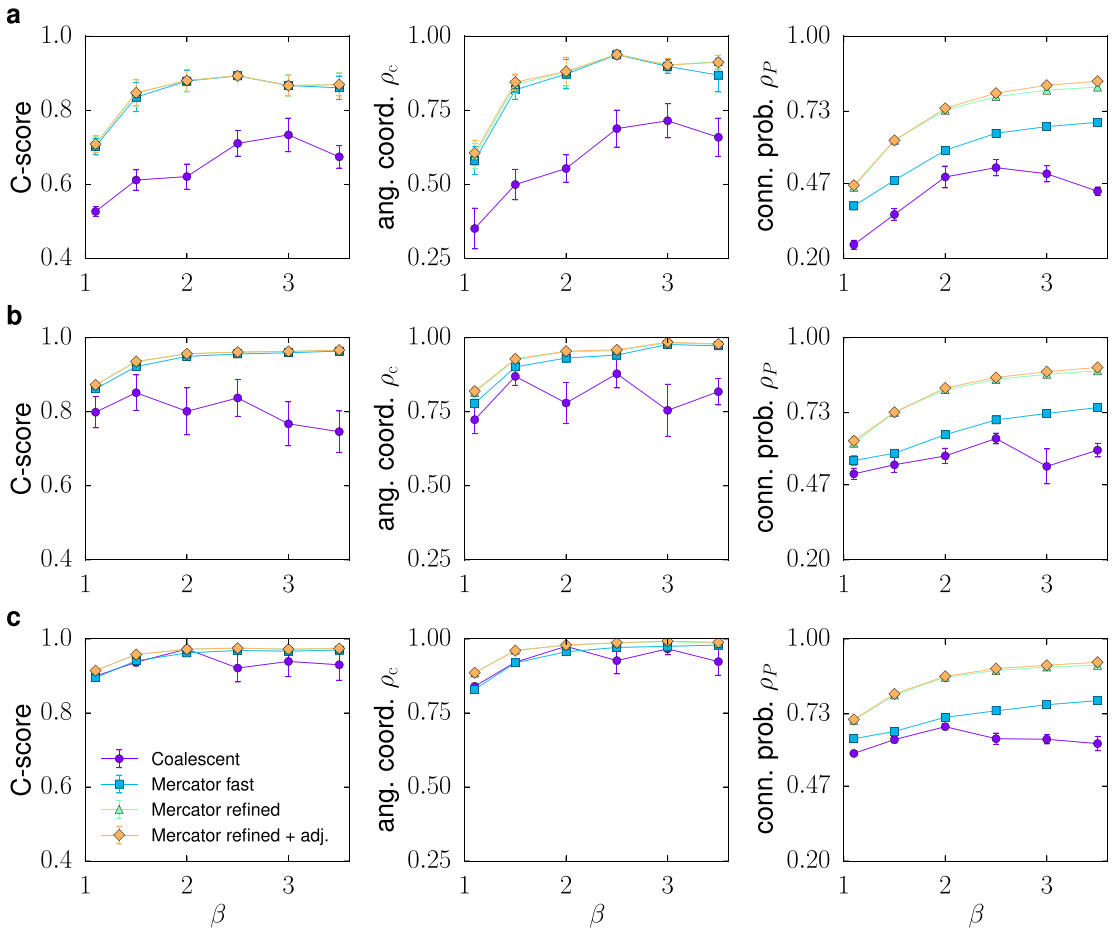

Figure 1. Comparison between the angular coordinates inferred by different embedding methods in terms of the C-score (left) and the Pearson correlation coefficient between original and inferred angular coordinates (middle), as a function of the clustering parameter β. The right column shows the Pearson correlation coefficient between the original and the inferred connection probabilities of all pairs of nodes. Every row corresponds to a different value of $\langle k \rangle$: (a) 4, (b) 8, and (c) 12. For every value of the parameters, we generated and embedded 10 synthetic networks with the $\mathbb{S}^1$ model of size $N = 1000$ and with power-law degree distribution exponent $\gamma = 2.5$. The plots show the resulting averages and standard deviations.

their position in the k-core decomposition alone. Doing so allows the likelihood optimization phase to begin with the most central nodes (based on mesoscale topological information) thereby greatly facilitating the finding of an acceptable local maximum in the configuration space.

10. For each node in the sorted list, find the average angular coordinate of its neighbors.

11. Sample $\mathcal{O}(\ln N)$ angular positions around this average value using a normal distribution whose standard deviation is set to half of the angular distance between this average value and the farthest neighbor.

12. Compute the local log-likelihood of the sampled angular positions

$$\ln \mathcal{L}_i = \sum_{j \neq i} a_{ij} \ln p_{ij} + (1 - a_{ij}) \ln(1 - p_{ij}), \tag{14}$$

where p_{ij} is computed with equation (1), and set the new angular position of the node to the sampled angle with highest log-likelihood.

13. Once the position of every node has been optimized once, repeat step 2 to find a better estimate of the hidden variables κ_i using the newly inferred angular positions. This last step is optional, although it generally leads to substantial improvements of the final embedding.

4. Validation in synthetic networks

4.1. Testing the $\mathbb{S}^1$ model

We test Mercator using synthetic networks of different average degrees and clustering coefficients generated with the $\mathbb{S}^1$ model, and consider several quality measures to check the accuracy of the embeddings. We first focus

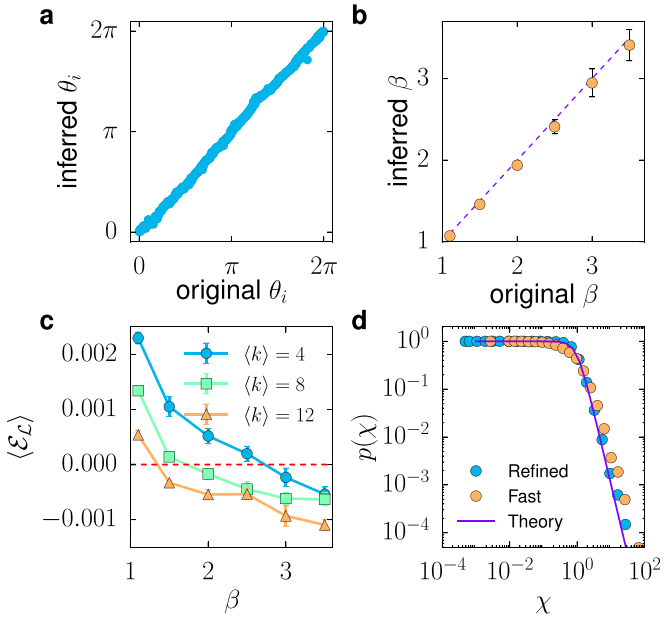

Figure 2. (a) Example of the inferred angular coordinates versus original angular coordinates for a network with $\langle k \rangle = 12$ and $\beta = 3$. (b) Inferred versus original values of the global parameter β obtained from the embeddings of the networks in figure 1. (c) Relative likelihood difference $\langle \mathcal{E}_\mathcal{L} \rangle = \langle \bar{\mathcal{L}}_{\text{infer}} / \bar{\mathcal{L}}_{\text{real}} \rangle - 1$ averaged over all the networks of a given $\langle k \rangle$ and β embedded with Mercator (refined mode with final adjustment of the $\{\kappa_i\}$ at step #13). In many cases, Mercator is able to find embeddings with likelihoods above those that generated the networks. In both plots, the errorbars represent the corresponding standard deviations. (d) Empirical connection probabilities, obtained using the fast and refined modes of Mercator, compared to the theoretical curve for the network in (a).

on its capability to recover the angular coordinates. Figure 1 shows the C-score [24], defined as the fraction of pairs of nodes that are correctly ordered in the circle, as well as the Pearson correlation coefficient between the real and inferred angular coordinates[12] and connection probabilities. The results of the two versions of Mercator are compared with those obtained using the Coalescent embedding[13] presented in [24], which was reported to give the best node orderings with respect to other embedding algorithms in the literature. Notice that Mercator is able to outperform the results even in its fast mode, especially for networks with a low average degree. Strikingly, figure 1 reveals that the refined version of Mercator results in a significant improvement of the connection probabilities, despite the change in the coordinates being seemingly small. Figure 2(a) depicts, as an example, the inferred angular coordinates versus the real ones for one of the networks considered.

Mercator also has the clear advantage of systematically inferring the hidden degrees and global model parameters. In figure 2(b) we show that the inference of β is very precise for all the synthetic networks considered in this section. This has important implications for applications that require finding a good congruency between the network and the model. Indeed, Mercator is able to find embeddings with very high likelihoods. To quantify this, we consider the relative likelihood difference $\mathcal{E}_\mathcal{L} = \bar{\mathcal{L}}_{\text{infer}} / \bar{\mathcal{L}}_{\text{real}} - 1$, where $\bar{\mathcal{L}} \equiv \mathcal{L}^{2/N(N-1)}$ is the geometric average of the likelihood over all pairs of nodes. Hence, a positive (negative) $\mathcal{E}_\mathcal{L}$ indicates that the inferred embedding has a higher (lower) likelihood than the real coordinates and model parameters. Strikingly, for low values of β, the embeddings found by Mercator have $\mathcal{E}_\mathcal{L} > 0$ (figure 2(c)). Finally, figure 2(d) presents an example of the empirical connection probability (fraction of connected pairs as a function of the rescaled distance $\chi = d/(\mu\kappa\kappa')$), which is extremely congruent with equation (1), especially in its refined mode. Put together, these results reveal that Mercator is not just the most accurate algorithm in terms of the reconstruction of the angular coordinates of synthetic networks—arguably the most difficult aspect of the embedding problem —, but it also determines correctly all other model parameters, including hidden degrees.

4.2. Testing the PSO model

We also compare Mercator with the Coalescent embedding algorithm proposed in [30] for synthetic networks generated with the PSO model [3] and with the non-uniform PSO model [7]—growing geometric models that generate complex networks in hyperbolic space, the latter of the two with inhomogeneous angular distributions.

[12] Notice that the model is invariant with respect to global rotations and inversions of the angular coordinates. Therefore, we consider the maximal Pearson correlation coefficient over all such possible transformations.

[13] We used the repulsion–attraction pre-weighting rule RA1, LE for dimensionality reduction, and equidistant adjustment (EA). This choice is motivated by the fact that LE was reported to yield the best results on synthetic networks [24].

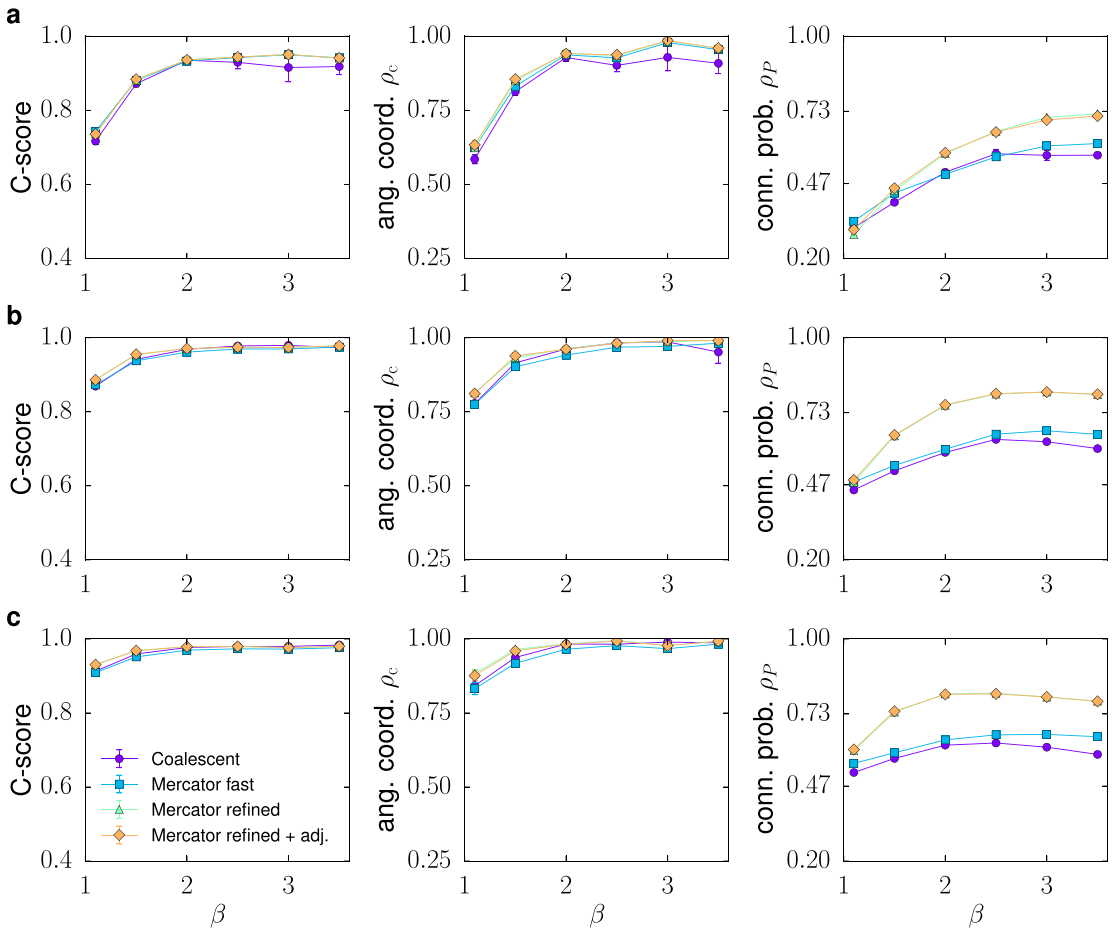

Figure 3. Comparison between the angular coordinates inferred by different embedding methods in terms of the C-score (left) and the Pearson correlation coefficient between original and inferred angular coordinates (middle), as a function of the clustering parameter β, for networks generated with the PSO model. The right column shows the Pearson correlation coefficient between the original and the inferred connection probabilities of all pairs of nodes. Every row corresponds to a different value of $\langle k \rangle$: (a) 4, (b) 8, and (c) 12. For every value of the parameters, we generated and embedded 10 synthetic networks of size $N = 1000$ and with power-law degree distribution exponent $\gamma = 2.5$. The plots show the resulting averages and standard deviations.

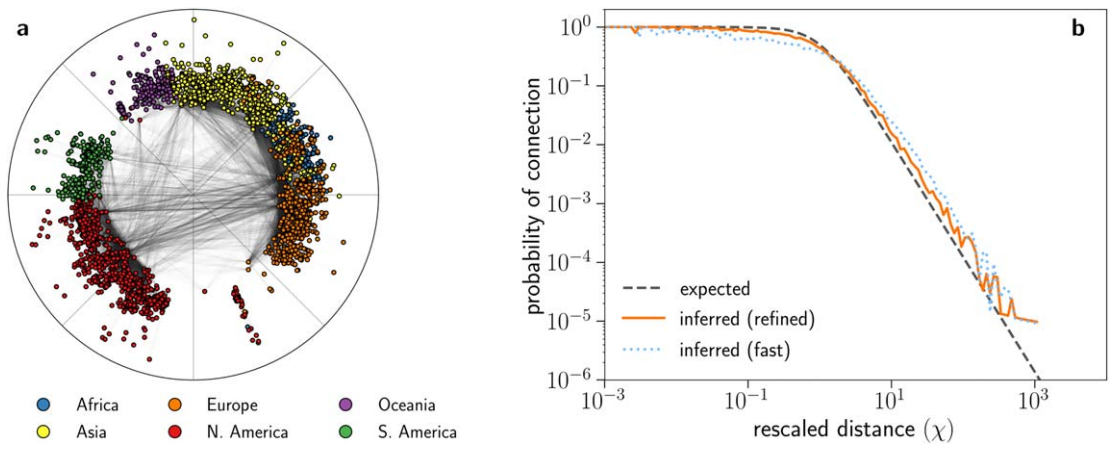

Figure 4. (a) Hyperbolic embedding of the world airports network obtained by Mercator in the refined mode. Nodes are colored according to the continent in which they are located, an information that is not used to obtain the embedding. (b) Comparison of the expected connection probability based on the inferred value of β (expected) and the actual connection probability computed with the inferred hidden variables $\{k_i, \theta_i\}$ (inferred) obtained using the fast and refined modes.

Results are shown in figure 3 and in figure S6 in the supplemental material. Even if Mercator often results in embeddings with better congruency with the network's ground-truth (especially regarding the connection probabilities), it is worth mentioning that the improvement with respect to the Coalescent embedding algorithm

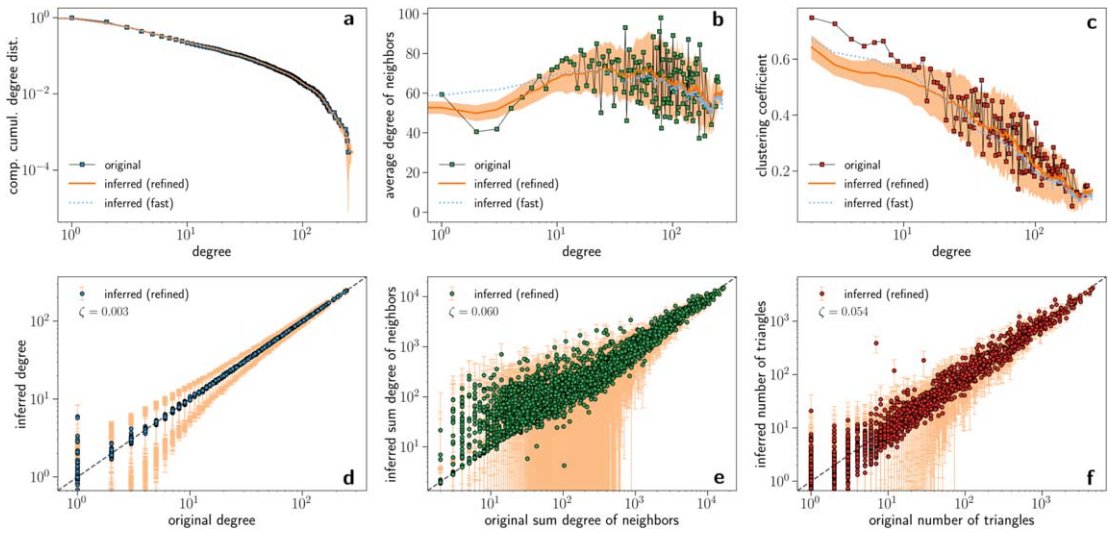

Figure 5. Topological validation of the embedding of the airports network. The first row shows (a) the complementary cumulative degree distribution, (b) the average nearest neighbors degree, $\bar{k}_{nn}(k)$, and (c) the clustering spectrum, $\bar{c}(k)$. Symbols correspond to the value of these quantities in the original network, whereas the red lines indicate an estimate of their expected values in the ensemble of random networks inferred by Mercator in the refined mode (the results obtained using the fast mode of Mercator are shown in blue). This ensemble was sampled by generating 100 synthetic networks with the $\mathbb{S}^1$ model and the inferred parameters and positions by Mercator. The orange regions correspond to an estimate of the 2σ confidence interval around the expected values. The second row shows scattered plots of (d) the degree of every nodes, (e) the sum of the degrees of their neighbors, and (f) the number of triangles to which they participate. The plots show the estimated values of these three measures in the same ensemble of random networks considered above versus the corresponding values in the original network. The error bars show the 2σ confidence interval around the expected values. The quantity ζ corresponds to the fraction of nodes for which the value measured on the original network lies outside the 2σ confidence interval.

is less pronounced for these models than it is for the $\mathbb{S}^1$ model. However, it is important to stress that these two growing models do not reproduce a crucial feature of real-world networks, namely, the increasing average degree of the subgraphs given by the nodes of degree larger than a given threshold k_T, found in [1, 4]; this is a direct consequence of the fact that, in these growing models, nodes are added sequentially and every new node adds exactly m links. Furthermore, since degrees are strongly correlated with nodes' ages, degree-thresholding is basically equivalent to selecting the subgraph given by all the N_{k_T} nodes older than some age, which contains m links per node and, hence, its average degree is $\langle k(k_T)\rangle = 2mN_{k_T}/N_{k_T} = 2m = \langle k\rangle$. The $\mathbb{S}^1$ model, on the other hand, reproduces this crucial topological feature [1] (and it is also able to generate degree distributions which are not clean power-laws), so it is reasonable to assume that a higher fidelity in recovering the $\mathbb{S}^1$-model ground-truth is preferable in order to uncover the geometric information encoded in real networks.

5. Embedding of real networks

Another strength of Mercator is its ability to embed networks with arbitrary degree distributions. As an illustration, we embedded several real-world complex networks from different domains whose degree distributions include clean scale-free, heavy-tailed, and arbitrary distributions. More specifically, the networks under study are: the world airport network[14], the neural network of the visual cortex of the Drosophila Melanogaster at the neuron level [31], the neural network of the *C. Elegans* worm [32], a human connectome [33, 34], the metabolic network of the bacterium *E. Coli* [15, 35], the world trade web [16], a US commute network [36], a cargo ships network [37], a US commodities network [36], and the Internet at the Autonomous Systems level [10].

One particularly telling example is the airports network whose truncated power-law degree distribution with exponent $\gamma < 2$ cannot be easily embedded with methods based on the PSO model. In the case of real networks, we do not have access to the 'real' coordinates to compare with those obtained from our embeddings. Yet, in some cases, metadata related to the similarity between nodes is available and can be used to test whether an embedding is meaningful or, instead, is an artifact of the algorithm. In the case of the airports network, geography is such metadata. Figure 4(a) shows the hyperbolic embedding obtained by Mercator in the refined mode with nodes colored according to the continent they belong to (separating North and South America).

[14] Downloaded from https://openflights.org/data.html.

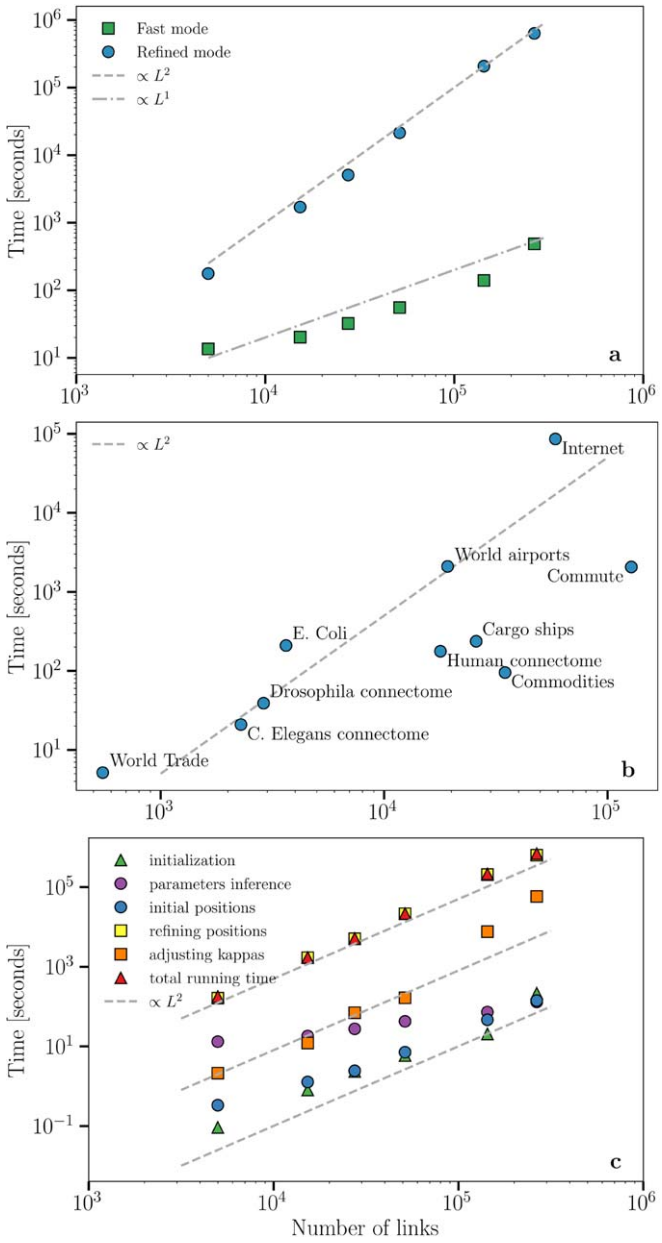

Figure 6. Comparison of the computational complexity of Mercator expressed in terms of the running time versus the number of links in the network (L). (a) Comparison between Mercator's fast and refined modes using synthetic networks generated with the $\mathbb{S}^1$ model with a power law distribution for the expected degrees κ with exponent $\gamma = 2.2$, $\beta = 2$ (clustering) and $\langle k \rangle = 10$. Higher values of γ lead to smaller running times but similar scaling behavior as the number of links increases. (b) Computational complexity of Mercator's refined mode applied to the real network datasets presented in section 5. (c) Mercator's computational complexity broken down into each each step of the algorithm. The same synthetic networks as in (a) were used. Dashed lines proportional to L or L^2 have been added to guide the eye.

Airports belonging to the same continent appear clustered in similar angular positions, thus supporting the relation between the angular space of the embedding and similarity among nodes. Similar analyses were carried out for the Internet at the autonomous systems level [10], metabolic networks [15], and the world trade web [16]. A strong correlation between the angular distribution of points and available metadata was found in all cases. In light of these results, we conclude that our geometric embeddings are meaningful and capture attributes that contribute to the similarity among the elements of complex networks.

Beyond this qualitative agreement, we tested the extent to which the embedding inferred by Mercator is accurate enough to reproduce the topology of the airports network. To do so, we first compare the expected connection probability equation (1) with the inferred value of β against the empirical connection probability, computed using the inferred coordinates of the nodes ($\{\kappa_i, \theta_i\}$). This remarkable agreement confirms that the rescaled distance χ provides a meaningful measure to characterize the interaction between nodes in the network. Second, we used the set of coordinates $\{\kappa_i, \theta_i\}$ as well as the parameters β and μ to generate an ensemble of

synthetic networks using equation (1). We then compared several topological properties of this ensemble with those measured on the original network. Specifically, the first row of figure 5 shows the results for the complementary cumulative degree distribution $P_c(k)$, the average nearest neighbors' degree $\bar{k}_{nn}(k)$, and the clustering spectrum $\bar{c}(k)$. The final adjustment of hidden degrees in step #13 of the Mercator algorithm strongly enhances the reproduction of the degree distribution. Notice that the $\mathbb{S}^1$ model does not include any mechanism to control degree–degree correlations or the shape of the clustering spectrum (recall that β is chosen based on the average clustering coefficient only). Yet, the generated network ensemble reproduces these two quantities with remarkable precision. This is particularly interesting in the case of the average nearest neighbors' degree, which shows a non-trivial assortative behavior for low degrees both in the real network and the ensemble. This suggests that the non-uniform angular distribution of nodes inferred by Mercator (and so the network geometric properties) is partly responsible for the observed degree–degree correlations in real complex networks.

The ensemble of synthetic networks generated from the estimated geometric parameters of the airports network performs also very well at reproducing topological properties of individual nodes. The second row of figure 5 shows scattered plots of the degree of a given node, the sum of degrees of its neighbors, and number of triangles attached to it in the generated network ensemble versus the same quantities measured on the original network. For each node, we also compute the 2σ confidence interval, and ζ shows the fraction of nodes whose original property (degree, sum of neighbors' degree, number of triangles) lies outside this interval. Results show that the fraction of points outside this interval is around 6%, which is consistent with the 2σ (or $\approx$95%) confidence interval. These results, supported by those presented in the supplementary information, clearly illustrate the accuracy of the embeddings provided by Mercator. To the best of our knowledge, such accuracy cannot be obtained with other existing embedding methods.

6. Computational complexity

We now support our claim that the computational complexity of Mercator scales as $\mathcal{O}(N^2)$ for sparse networks composed of N nodes (and $L = \langle k \rangle N/2$ links). Figures 6(a) and (b) show the running time in seconds in function of the number of links (L) for both synthetic and real networks. In both cases, we find that Mercator's refined mode does indeed roughly scale as L^2 although it is clear that other topological properties influence the final total running time. With respect to the fast mode, we find that the computational complexity is roughly linear within the range in the number of links that was considered. Finally, figure 6(c) breaks down the running time into the time spent in each of the Mercator major steps, and doing so shows how most of the running time is spent during the ML step.

7. Conclusions

In this work, we introduce and deliver the full code of Mercator, the most accurate method to embed complex networks into their latent metric spaces. We showed that the quality of the embeddings can be significantly improved by a proper combination of machine learning techniques and powerful statistical methods. Thanks to this combination, Mercator is able to overcome some of the drawbacks of other techniques which, for instance, require perfect power laws in the whole domain of degrees, a condition that is not met by many real networks. Our results also indicate that the obtained embeddings are able to recover ground truth information not contained in the network topology. We expect Mercator to become a standard tool within the toolbox of network scientists and anybody interested in retrieving information from big data systems admitting a network representation.

Acknowledgments

We thank Elisenda Ortiz for useful comments and for an extensive testing of the beta version of Mercator. We acknowledge support from the project *Mapping Big Data Systems: embedding large complex networks in low-dimensional hidden metric spaces*—Ayudas Fundación BBVA a Equipos de Investigación Científica 2017; James S McDonnell Foundation Scholar Award in Complex Systems; the ICREA Academia prize, funded by the Generalitat de Catalunya; Ministerio de Economía y Competitividad of Spain, project no. FIS2016-76830-C2-2-P (AEI/FEDER, UE). GGP acknowledges financial support from the Academy of Finland via the Centre of Excellence program (Project no. 312058 as well as Project no. 287750), and from the emmy.network foundation under the aegis of the Fondation de Luxembourg. AA acknowledges financial support from the project Sentinelle Nord of the Canada First Research Excellence Fund, from the Natural Sciences and Engineering

Research Council of Canada, from 'la Caixa' Foundation and from the Spanish 'Juan de la Cierva-incorporación' program (IJCI-2016-30193).

Appendix. Mercator in details

We now provide the full details of Mercator.

A.1. Sketch of the method

The $\mathbb{S}^1$ model has two global parameters that need to be inferred: μ, controlling the average degree and β, which determines the level of clustering in the network. In addition, every node i is assigned two hidden variables: a hidden degree κ_i and an angular coordinate θ_i. The following method finds the values of μ, β and κ_i for which the expected degrees in the model $\bar{k}(\kappa_i)$, that is, in synthetic networks generated with uniformly distributed angular positions, equal the observed degrees in the real network and, moreover, the expected mean local clustering of the embedding matches the real value. To that end, some values of μ and β are proposed. Next, the corresponding κ_i are calculated. Finally, the expected clustering coefficient is computed and β is adjusted if the predicted value is not within an acceptable range of the original value.

The method relies on the assumption that all nodes with the same degree have the same hidden degree. Therefore, the first preliminary step is reading the network and counting the number of nodes in every degree class k, that we denote by N_k.

A.2. Inferring the hidden degrees

This step assumes some given value of β and the corresponding $\mu = \frac{\beta}{2\pi\langle k\rangle}\sin\frac{\pi}{\beta}$, where $\langle k\rangle$ is the observed average degree. We then assign to every degree class the hidden degree given by $\kappa(k) = k$ as the initial guess. The aim of the following algorithm is to adjust this relation so that $\bar{k}(\kappa(k)) = k \pm \epsilon$, where ϵ can be set, for instance, to $\epsilon = 0.01$. To solve this problem, we need a way to reckon the values of $\bar{k}(\kappa(k))$ from the relation $\kappa(k)$. To that end, it is useful to consider the probability for two nodes with hidden degrees κ and κ' to be connected in the ensemble of networks with global parameters $R = N/(2\pi)$, μ, β and uniformly distributed angular coordinates. This probability is given by

$$p(a_{\kappa\kappa'} = 1) = \int_0^\pi \frac{1}{\pi}\frac{1}{1+\left(\frac{R\Delta\theta}{\mu\kappa\kappa'}\right)^\beta}\mathrm{d}\Delta\theta = {}_2F_1\left(1, \frac{1}{\beta}, 1+\frac{1}{\beta}, -\left(\frac{R\pi}{\mu\kappa\kappa'}\right)^\beta\right). \tag{A1}$$

Starting from the initial guess $\kappa(k) = k$, we perform the following steps to refine the relation $\kappa(k)$:

1. *Initialize expected degrees:* For every degree class k, set $\bar{k}(\kappa(k)) = 0$.
2. *Compute expected degrees:* For every pair of degree classes (k, k'), compute $P \equiv p(a_{\kappa(k)\kappa(k')} = 1)$ using equation (A1). Set $\bar{k}(\kappa(k)) + N_{k'}P \to \bar{k}(\kappa(k))$ and $\bar{k}(\kappa(k')) + N_kP \to \bar{k}(\kappa(k'))$. By doing so, we add the expected number of connections of a node in degree class k with nodes in degree class k' and vice-versa. Notice that, when $k = k'$, we set $\bar{k}(\kappa(k))) + (N_k - 1)P \to \bar{k}(\kappa(k))$ instead.
3. *Compute largest deviation:* Let $\epsilon_{\max} = \max\{|\bar{k}(\kappa(k)) - k|\}_k$ be the maximal deviation between degrees and expected degrees. If $\epsilon_{\max} > \epsilon$, the values of $\kappa(k)$ need to be corrected. Then, for every degree class k, set $|\kappa(k) + [k - \bar{k}(\kappa(k))]u| \to \kappa(k)$, where u is a random variable drawn from $U(0, 1)$. The rationale behind this transformation is that every degree-class hidden degree is corrected according to its expected-degree excess or deficiency; the random variable u prevents the process from getting trapped in a local minimum. Next, go to step 1 to compute the expected degrees corresponding to the new set of $\kappa(k)$. Otherwise, if $\epsilon_{\max} \leqslant \epsilon$, hidden degrees have been inferred for the current global parameters.

A.3. Inferring parameter β

To infer β, we need to compute the expected mean local clustering $\bar{c}$ given the current values of the global parameters as well as of the hidden-degree distribution provided by $\kappa(k)$ and N_k found using the algorithm from the last subsection. The method is based on the following idea. Suppose we want to estimate the expected clustering $\bar{c}(k)$ of some node with degree k. According to the definition of mean local clustering, this quantity is given by the probability for two randomly chosen neighbors of the node to be connected, which can be computed in two steps: first, we randomly choose two of its neighbors and draw their distances to the node from the distribution of distances between connected nodes in the model. Second, we compute the distance between the two neighbors and, with it, the probability for them to be connected. Two important points require some clarification:

a. The model is uncorrelated at the hidden level. Therefore, in the calculation of the clustering, we draw the two neighbors from the uncorrelated distribution $P(k|k') = kP(k)/\langle k\rangle$.

b. The distribution of angular distance $\Delta\theta$ between two connected nodes with hidden degrees κ and κ', $\rho(\Delta\theta|a_{\kappa\kappa'} = 1)$, where $a_{\kappa\kappa'}$ stands for the corresponding adjacency-matrix element, is given by

$$\rho(\Delta\theta|a_{\kappa\kappa'} = 1) = \frac{p(a_{\kappa\kappa'} = 1|\Delta\theta)\rho(\Delta\theta)}{p(a_{\kappa\kappa'} = 1)}. \tag{A2}$$

In the above expression, $p(a_{\kappa\kappa'} = 1|\Delta\theta) = 1/(1 + (R\Delta\theta/(\mu\kappa\kappa')^\beta)$ is the connection probability between the two nodes with hidden degrees κ and κ' separated by a distance $\Delta\theta$. The distribution of distances in the $\mathbb{S}^1$ model is simply $\rho(\Delta\theta) = 1/\pi$, since angular coordinates are uniformly distributed. Finally, $p(a_{\kappa\kappa'} = 1)$ is given in equation (A1). Equation (A2) therefore reads

$$\rho(\Delta\theta|a_{\kappa\kappa'} = 1) = \frac{\frac{1}{\pi}\frac{1}{1+\left(\frac{R\Delta\theta}{\mu\kappa\kappa'}\right)^\beta}}{{}_2F_1\left(1, \frac{1}{\beta}, 1 + \frac{1}{\beta}, -\left(\frac{R\pi}{\mu\kappa\kappa'}\right)^\beta\right)}. \tag{A3}$$

The expected mean local clustering can now be found following three steps:

1. *Initialize mean local clustering:* Let $\bar{c}(k)$ represent the expected mean local clustering of degree class k. Set $\bar{c}(k) = 0$ for all k.

2. *Compute expected mean local clustering spectrum:* For every degree class k, do m times:

   i. Draw two variables k_i from $P(k_i|k)$, $i = 1, 2$.

   ii. Draw the corresponding random variables $\Delta\theta_i$ from the distributions $\rho(\Delta\theta_i|a_{\kappa(k)\kappa(k_i)} = 1)$, $i = 1, 2$ given in equation (A3).

   iii. Consider the two semicircles spanned by the diameter of the circle passing through the degree-k node. It is equally likely for its two neighbors to lay in the same or in different semicircles. Hence, with probability 1/2, set $\Delta\theta_{12} = \min(|\Delta\theta_1 + \Delta\theta_2|, 2\pi - |\Delta\theta_1 + \Delta\theta_2|)$ or $\Delta\theta_{12} = |\Delta\theta_1 - \Delta\theta_2|$.

   iv. Set $\bar{c}(k) + p_{12}/m \to \bar{c}(k)$, where $p_{12} = \frac{1}{1+\left(\frac{R\Delta\theta_{12}}{\mu\kappa(k_1)\kappa(k_2)}\right)^\beta}$ is the probability for nodes 1 and 2 to be connected.

3. *Compute expected mean local clustering:* The expected mean local clustering $\bar{c}$ can be readily computed as $\bar{c} = \sum_k \bar{c}(k)N_k/N$.

If the error in the expected mean local clustering $|\bar{c} - \bar{c}^{\rm emp}| < \epsilon_{\bar{c}}$, where $\epsilon_{\bar{c}}$ is the desired precision and $\bar{c}^{\rm emp}$ is the observed mean local clustering coefficient, we can accept the current value of β and proceed to the inference of the angular coordinates. Otherwise, β needs to be corrected and the hidden degrees must be recalculated accordingly by repeating the process explained in the previous subsection. Notice that, since the expected mean local clustering coefficient is a monotonic function of β, the process can be iterated efficiently using the bisection method. In practice, however, we use a modified version of the bisection method in which the upper bound is let free until we reach a value of β for which the expected clustering is higher than the observed one. More precisely, we start with a value for β picked uniformly between 2 and 3. Then, while the expected clustering is lower than the observed one, we increase β by multiplying it by 1.5. When we reach a value for which the observed clustering is surpassed, we start the regular bisection method. We also note that, for $\epsilon_{\bar{c}} = 0.01$, $m = 600$ is enough. Of course, if more precision is required, m must be increased to guarantee that the fluctuations in the computed $\bar{c}(k)$ are small enough.

A.4. Angular coordinates

Having inferred the values for the parameters β, μ and $\{\kappa_i\}$, we are in a position to infer the angular coordinates, $\{\theta_i\}$, of each node. This is performed by following two steps: a *machine learning* step providing an initial ordering of the nodes as well as realistic positions, and a second step in which nodes are moved in order to maximize the likelihood that the $\mathbb{S}^1$ model generated the original edge list.

A.5. Initial ordering and positions

This step is a modified version of the LE algorithm introduced in [27] and used in [23, 24]. This machine learning algorithm was originally conceived for dimensionality reduction. The main idea is as follows. Given a set of points in $\mathbb{R}^n$, the algorithm first constructs a RGG by, for instance, connecting points located at a distance below some threshold in n-dimensional Euclidean space. Once this graph is known, the points are mapped to $\mathbb{R}^m$ with $m < n$ by diagonalizing the corresponding Laplacian and assigning to every point i the coordinates $\mathbf{y}_i = (v_1^i,\cdots,v_m^i)$, where v_j^i is the ith component of the jth Laplacian eigenvector with non-null eigenvalue (the eigenvectors are ordered according to their eigenvalues). It can be shown that these coordinates minimize the squared distances between connected pairs in the RGG

$$\epsilon = \sum_{i,j} |\mathbf{y}_i - \mathbf{y}_j|^2, \tag{A4}$$

while preventing all nodes from collapsing into a single point. Furthermore, the relevance of every connection (i, j) in the RGG in the above expression can be modulated by assigning a weight ω_{ij} to it according to

$$\omega_{ij} = \mathrm{e}^{-\frac{|\mathbf{x}_i-\mathbf{x}_j|^2}{t}}, \tag{A5}$$

where $|\mathbf{x}_i - \mathbf{x}_j|$ is the distance between the points in $\mathbb{R}^n$ and t is a scaling factor fixed as the mean of the squares of all the distances [24]. The same procedure then leads to the minimization of

$$\epsilon = \sum_{i,j} |\mathbf{y}_i - \mathbf{y}_j|^2\omega_{ij}, \tag{A6}$$

The approach taken by [23, 24] is to consider the network to be embedded as the RGG generated in a higher-dimensional space. Hence, by proceeding to the dimensionality reduction in m (typically $m = 2$) dimensions, we obtain an embedding in R^m, which can be radially normalized so that all points lay in $\mathbb{S}^{m-1}$. The improvement in [24] is to assign weights according to some heuristic and, once the coordinates on the plane are known, these are used to infer the ordering of nodes only. The final coordinates are computed by distributing all nodes on the circle with $\theta_{i+1} - \theta_i = 2\pi/N$, $\forall\, i$. We now propose an improvement based on assigning the weights in the Laplacian matrix as well as the gaps $\theta_{i+1} - \theta_i$ according to the $\mathbb{S}^1$ model.

1. *LE for node ordering:* Since degree-one nodes do not add geometric information, we remove them at this step and work with the subgraph of nodes with $k > 1$. We then apply the LE method to such graph after weighting every link according to equation (A5), where we use

$$|\mathbf{x}_i - \mathbf{x}_j| = 2\sin\frac{\langle\Delta\theta_{ij}\rangle}{2} \tag{A7}$$

as a proxy for the distance $|\mathbf{x}_i - \mathbf{x}_j|$ and $\langle\Delta\theta_{ij}\rangle$ is the expected angular distance between nodes i and j in the $\mathbb{S}^1$ model conditioned to the fact that they are connected. The above expression simply maps such expected angular distance onto the corresponding cord length, since LE is designed to work on Euclidean space and, therefore, it seems natural to consider the $\mathbb{S}^1$ model as embedded in $\mathbb{R}^2$ for the algorithm. The expected distance between the nodes can be readily computed from equation (A3) as

$$\langle\Delta\theta_{ij}\rangle = \int_0^\pi \Delta\theta_{ij}\,\rho(\Delta\theta_{ij}|a_{ij}=1)\,\mathrm{d}\Delta\theta_{ij} = \frac{\pi\ {}_2F_1\left(1,\frac{2}{\beta},1+\frac{2}{\beta},-\left(\frac{R\pi}{\mu\kappa(k_i)\kappa(k_j)}\right)^\beta\right)}{2\ {}_2F_1\left(1,\frac{1}{\beta},1+\frac{1}{\beta},-\left(\frac{R\pi}{\mu\kappa(k_i)\kappa(k_j)}\right)^\beta\right)}. \tag{A8}$$

Since a similar approach developed in [24] has been shown to yield very good results in terms of the angular ordering of the nodes, we use this machine learning step to define a sequence of angular coordinates

$$S' = (\theta_1,\ldots,\theta_{N'}) \tag{A9}$$

for the nodes in the subgraph, where the angles in S' are ordered in increasing order. Each θ_i is computed as

$$\theta_i = \mathrm{atan2}(v_2^i,\ v_1^i), \tag{A10}$$

where v_1^i and v_2^i are the x and y coordinates of node i found by LE. Once we have the ordering of nodes with $k > 1$, we reincorporate the degree-one nodes. This can be easily done by replacing every node i with t degree-one neighbors by the sequence $(n_1^i,\ldots,n_{\lfloor t/2\rfloor}^i,\ i,\ n_{\lfloor t/2\rfloor+1}^i,\ldots,n_t^i)$ in S', where n_j^i is the jth degree-one neighbor of node i (in any arbitrary order) and $\lfloor\cdot\rfloor$ is the floor function. Such operation yields a new sequence of nodes S including all the nodes in the original graph.

2. *Order-preserving adjustment:* This last step of the approximate embedding locates the nodes on the circle preserving the ordering of the nodes in S. To that end, we set every node's coordinate such that the gap

between two consecutive nodes in S is proportional to the expected gap between two consecutive nodes with the same hidden variables and adjacency-matrix element in the $\mathbb{S}^1$ model, except for the gaps in which LE predicts a gap larger than the model. The reason for this heuristic is that, in networks with community structure, large gaps indicate separation between communities, and these are detected by the dimensionality reduction algorithm. We thus apply these steps:

i. *Computing the expected gaps:* Let nodes i and $i+1$ be consecutive in S. The distribution for the length of the gap g_i between them in the $\mathbb{S}^1$ model can be obtained from Bayes' rule, as in equation (A2)

$$\rho(g_i|a_{i+1,i}) = \frac{p(a_{i+1,i}|g_i)\rho(g_i)}{\int_0^\pi p(a_{i+1,i}|g_i)\rho(g_i)\mathrm{d}g_i}, \tag{A11}$$

where now $\rho(g_i)$ is an exponential distribution with mean $2\pi/N$

$$\rho(g_i) = \frac{N}{2\pi}\mathrm{e}^{-\frac{N}{2\pi}g_i}, \tag{A12}$$

and

$$p(a_{i+1,i}|g_i) = \left(\frac{1}{1+\left(\frac{Rg_i}{\mu\kappa(k_{i+1})\kappa(k_i)}\right)^\beta}\right)^{a_{i+1,i}} \times \left(1-\frac{1}{1+\left(\frac{Rg_i}{\mu\kappa(k_{i+1})\kappa(k_i)}\right)^\beta}\right)^{1-a_{i+1,i}}. \tag{A13}$$

The expected gap $\langle g_i\rangle$ can thus be computed as

$$\langle g_i\rangle = \frac{\int_0^\pi g_i p(a_{i+1,i}|g_i)\rho(g_i)\mathrm{d}g_i}{\int_0^\pi p(a_{i+1,i}|g_i)\rho(g_i)\mathrm{d}g_i}. \tag{A14}$$

Both integrals can be carried out numerically. Once the expected gap $\langle g_i\rangle$ has been computed, set

$$g_i = \max(\langle g_i\rangle, g_i^{\mathrm{LE}}), \tag{A15}$$

where g_i^{LE} stands for the angular separation between the nodes as given by the LE algorithm.

ii. *Normalizing the gaps:* By applying the last step to every pair of consecutive nodes (including the pair $(N, 1)$), we obtain a sequence of N expected gaps which, however, needs not sum up to 2π, so we normalize each g_i as

$$\tilde{g}_i = 2\pi\frac{g_i}{\sum_{i=1}^N g_i}. \tag{A16}$$

Finally, we can assign every node's coordinate sequentially, starting with $\theta_1 = 0$, as $\theta_i = \theta_{i-1} + \tilde{g}_{i-1}$, $i = 2,\ldots,N$.

A.6. Likelihood maximization

This stage of the embedding algorithm adjusts the angular coordinates that maximize the likelihood for the observed network to be generated by the model. As opposed to previously proposed likelihood-maximization schemes, we do not need to explore a vast region of configuration space, since the machine learning stage provides a set of coordinates located near an optimal configuration. Hence, we visit every node once and propose several new angular coordinates for it, keeping the one with higher log-likelihood. The steps we follow are:

1. *Define a new ordering of nodes:* We visit the nodes in the order defined by the network's onion decomposition. In the sequence, the ordering among nodes belonging to the same layer in the decomposition is random.

2. *Find new optimal coordinates:* For every node i, we select the optimal coordinate among candidate positions generated in the vicinity of the mean angular coordinate of its neighbors. This is achieved in three steps:

   i. *Compute mean coordinate of node i's neighbors:* Let node i have k_i neighbors, which we now label with index $j = 1, \ldots, k_i$. Since nodes lay on a circle, we must compute their mean angular coordinate $\bar{\theta}_i$ using the vector sum of their positioning vectors in $\mathbb{R}^2$. The polar angle of the resulting vector sum is given by

$$\bar{\theta}_i = \text{atan2}\left(\sum_j \frac{1}{\kappa_j^2}\sin\theta_j, \sum_j \frac{1}{\kappa_j^2}\cos\theta_j\right), \tag{A17}$$

where the hidden degrees in the above expression weight the contribution of every neighbor's positioning vector, as proposed in [21].

ii. *Propose new positions around* $\bar{\theta}_i$: We generate 100 $\max(\ln N, 1)$ candidate angular coordinates from a normal distribution with mean $\bar{\theta}_i$ and standard deviation σ given by

$$\sigma = \max\left(\frac{\pi}{12}, \frac{\Delta\theta_{\max}}{2}\right), \tag{A18}$$

where $\Delta\theta_{\max}$ is the angular distance between $\bar{\theta}_i$ and the most distant neighbor of node i, i.e.

$$\Delta\theta_{\max} = \max\{\min(|\theta_j - \bar{\theta}_i|, 2\pi - |\theta_j - \bar{\theta}_i|)\}_j. \tag{A19}$$

This last step is adapted from [38].

iii. *Select the most likely candidate position:* Compute the local log-likelihood of every candidate position as well as of node i's current angular coordinate according to

$$\ln\mathcal{L}_i = \sum_{j\neq i} a_{ij}\ln p_{ij} + (1 - a_{ij})\ln(1 - p_{ij}). \tag{A20}$$

Locate node i at the angular position maximizing the local log-likelihood.

A.7. Adjusting hidden degrees

The final process adjusts hidden degrees according to the hidden coordinates found so that $\bar{k}(\kappa_i) = k_i$. The algorithm is similar to the initial inference of hidden degrees:

1. *Compute expected degrees:* For every node i, set

$$\bar{k}(\kappa_i) = \sum_{j\neq i} \frac{1}{1 + \left(\frac{R\Delta\theta_{ij}}{\mu\kappa_i\kappa_j}\right)^{\beta}}. \tag{A21}$$

2. *Correct hidden degrees:* Let $\epsilon_{\max} = \max\{|\bar{k}(\kappa_i) - k_i|\}_i$ be the maximal deviation between degrees and expected degrees. If $\epsilon_{\max} > \epsilon$, the set of hidden degrees needs to be corrected. Then, for every node i, set $|\kappa_i + [k_i - \bar{k}(\kappa_i)]u| \to \kappa_i$, where u is a random variable drawn from $U(0, 1)$. As in section A.2, the random variable u prevents the process from getting trapped in a local minimum. Next, go to step 1 to compute the expected degrees corresponding to the new set of hidden degrees. Otherwise, if $\epsilon_{\max} \leqslant \epsilon$, hidden degrees have been inferred for the current global parameters and angular coordinates.

References

[1] Serrano M A, Krioukov D and Boguñá M 2008 *Phys. Rev. Lett.* **100** 078701
[2] Krioukov D, Papadopoulos F, Kitsak M, Vahdat A and Boguña M 2010 *Phys. Rev.* E **82** 036106
[3] Papadopoulos F, Kitsak M, Serrano M A, Boguñá M and Krioukov D 2012 *Nature* **489** 537
[4] Serrano M Á, Krioukov D and Boguñá M 2011 *Phys. Rev. Lett.* **106** 048701
[5] Zuev K, Boguná M, Bianconi G and Krioukov D 2015 *Sci. Rep.* **5** 9421
[6] García-Pérez G, Serrano M Á and Boguñá M 2018 *J. Stat. Phys.* **173** 775
[7] Muscoloni A and Cannistraci C V 2018 *New J. Phys.* **20** 052002
[8] Boguñá M, Krioukov D and Claffy K C 2009 *Nat. Phys.* **5** 74
[9] Gulyas A, Bíró J J, Kőrösi A, Rétvari G and Krioukov D 2015 *Nat. Commun.* **6** 7651
[10] Boguñá M, Papadopoulos F and Krioukov D 2010 *Nat. Commun.* **1** 62
[11] García-Pérez G, Boguñá M and Serrano M Á 2018 *Nat. Phys.* **14** 583
[12] Allard A, Serrano M A, García-Pérez G and Boguñá M 2017 *Nat. Commun.* **8** 14103
[13] Kleineberg K-K, Boguña M, Serrano M A and Papadopoulos F 2016 *Nat. Phys.* **12** 1076
[14] Kleineberg K-K, Buzna L, Papadopoulos F, Boguñá M and Serrano M A 2017 *Phys. Rev. Lett.* **118** 218301
[15] Serrano M A, Boguñá M and Sagués F 2012 *Mol. Biosyst.* **8** 843
[16] García-Pérez G, Boguñá M, Allard A and Serrano M A 2016 *Sci. Rep.* **6** 33441
[17] Wang Z, Wu Y, Li Q, Jin F and Xiong W 2016 *Physica* A **450** 609
[18] Kitsak M, Voitalov I and Krioukov D 2019 (arXiv:1903.08810)
[19] Papadopoulos F, Psomas C and Krioukov D 2015 *IEEE/ACM Trans. Netw.* **23** 198
[20] Papadopoulos F, Aldecoa R and Krioukov D 2015 *Phys. Rev.* E **92** 022807
[21] Bläsius T, Friedrich T, Krohmer A and Laue S 2016 *24th Annual European Symp. on Algorithms (ESA 2016)* vol 57 (*Leibniz International Proc. in Informatics (LIPIcs)*) ed P Sankowski and C Zaroliagis (Dagstuhl: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik) pp 16:1–16:18

[22] Blasius T, Friedrich T, Krohmer A, Laue S, Krohmer A, Laue S, Friedrich T and Blasius T 2018 *IEEE/ACM Trans. Netw.* **26** 920
[23] Alanis-Lobato G, Mier P and Andrade-Navarro M A 2016 *Sci. Rep.* **6** 30108
[24] Muscoloni A, Thomas J M, Ciucci S, Bianconi G and Cannistraci C V 2017 *Nat. Commun.* **8** 1615
[25] Muscoloni A and Cannistraci C V 2018 (arXiv:1802.01183)
[26] Alanis-Lobato G, Mier P and Andrade-Navarro M A 2016 *Appl. Netw. Sci.* **1** 10
[27] Belkin M and Niyogi P 2002 *Proc. 14th Int. Conf. on Neural Information Processing Systems: Natural and Synthetic, NIPS'01* (Cambridge, MA: MIT Press) pp 585–91
[28] Lehoucq R B and Sorensen D C 1996 *SIAM J. Matrix Anal. Appl.* **17** 789
[29] Hébert-Dufresne L, Grochow J A and Allard A 2016 *Sci. Rep.* **6** 31708
[30] Muscoloni A, Thomas J M, Ciucci S, Bianconi G and Cannistraci C V 2017 *Nat. Commun.* **8** 1615
[31] Takemura S-Y *et al* 2013 *Nature* **500** 175
[32] Varshney L R, Chen B L, Paniagua E, Hall D H and Chklovskii D B 2011 *PLoS Comput. Biol.* **7** e1001066
[33] Avena-Koenigsberger A, Misic B and Sporns O 2018 *Nat. Rev. Neurosci.* **19** 17
[34] Hagmann P, Cammoun L, Gigandet X, Meuli R, Honey C J, Wedeen V J and Sporns O 2008 *PLoS Biol.* **6** e159
[35] Orth J D, Conrad T M, Na J, Lerman J A, Nam H, Feist A M and Palsson B O 2011 *Mol. Syst. Biol.* **7** 535
[36] Grady D, Thiemann C and Brockmann D 2012 *Nat. Commun.* **3** 864
[37] Kaluza P, Kolzsch A, Gastner M T and Blasius B 2010 *J. R. Soc. Interface* **7** 1093
[38] Blasius T, Friedrich T, Krohmer A and Laue S 2018 *IEEE/ACM Trans. Netw.* **26** 920