Aalto University

OPEN ACCESS

This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.

Puolamäki, Kai; Henelius, Andreas; Ukkonen, Antti

Randomization algorithms for large sparse networks

Published in:
Physical Review E

DOI:
10.1103/PhysRevE.99.053311

Published: 30/05/2019

Document Version
Publisher's PDF, also known as Version of record

Published under the following license:
CC BY

Please cite the original version:
Puolamäki, K., Henelius, A., & Ukkonen, A. (2019). Randomization algorithms for large sparse networks. *Physical Review E*, *99*(5), 1-15. Article 053311. https://doi.org/10.1103/PhysRevE.99.053311

This material is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of the repository collections is not permitted, except that material may be duplicated by you for your research use or educational purposes in electronic or print form. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone who is not an authorised user.

Randomization algorithms for large sparse networks

Kai Puolamäki,[1,2,*] Andreas Henelius,[1,2,3,†] and Antti Ukkonen[1,‡]
[1]*Department of Computer Science, University of Helsinki, Finland*
[2]*Aalto University, Helsinki, Finland*
[3]*Finnish Institute of Occupational Health, Helsinki, Finland*

(Received 16 November 2018; revised manuscript received 8 March 2019; published 30 May 2019)

In many domains it is necessary to generate surrogate networks, e.g., for hypothesis testing of different properties of a network. Generating surrogate networks typically requires that different properties of the network are preserved, e.g., edges may not be added or deleted and edge weights may be restricted to certain intervals. In this paper we present an efficient property-preserving Markov chain Monte Carlo method termed CycleSampler for generating surrogate networks in which (1) edge weights are constrained to intervals and vertex strengths are preserved exactly, and (2) edge and vertex strengths are both constrained to intervals. These two types of constraints cover a wide variety of practical use cases. The method is applicable to both undirected and directed graphs. We empirically demonstrate the efficiency of the CycleSampler method on real-world data sets. We provide an implementation of CycleSampler in R, with parts implemented in C.

DOI: 10.1103/PhysRevE.99.053311

I. INTRODUCTION

In many applications it is useful to represent relationships between objects with a network in which vertices correspond to objects of interest and associations between objects are expressed with directed or undirected edges. The edges can also be weighted. Given such a network, one might be interested in questions such as community detection [1], clustering coefficients [2,3], centrality measures [4], shortest path distributions [5], or different measures of information propagation [6]. However, it is often useful to study whether a possibly interesting finding from a given network reflects a real phenomenon, or if it is merely caused by, e.g., noise or systematic errors. A simple approach to this is to compare the original finding to findings from *surrogate networks* that share some *relevant properties* with the original network but are otherwise inherently "random." For example, communities found in the original network should probably exhibit greater structure than communities in appropriately randomized networks. Usual solutions thus involve generating a number of surrogate networks by fixing some network properties of interest and drawing a uniform sample of surrogate networks from the set of all networks satisfying the given properties.

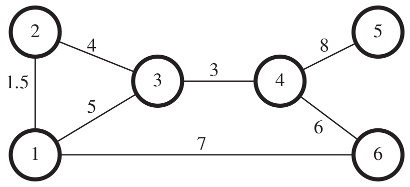
Existing methods for generating surrogate networks can be assigned into two categories: *property-preserving* and *structure-preserving* methods. Property-preserving methods [2,7–9] do not preserve network topology, i.e., they can introduce new edges and remove existing ones. These approaches can be viewed as always considering a fully connected clique, inside which the edge weights are rearranged. Their aim is to preserve some network property of interest, such as vertex degrees or vertex strengths. Property-preserving methods can be further divided into those preserving the property *exactly* or *in expectation*. For example, preserving vertex degrees exactly is relatively straightforward using, e.g., edge swaps [2,10]. Preserving higher-order statistics is often possible only in expectation [8]. That is, the expected value of the property remains equal to some given constraint, but its observed value in an individual surrogate network may deviate from this constraint.

Structure-preserving methods [11], on the other hand, keep the network topology fixed (new edges are not inserted and existing ones are not removed) but usually maintain the desired property, e.g., vertex strengths, only in expectation. Such approaches are usually based on *maximum-entropy* models [12–14] where surrogate networks are generated simply by drawing edge weights from a parametrized i.i.d. distribution. While these methods are often computationally quite efficient, maintaining the desired property only in expectation may not be enough. It is, e.g., conceivable that without additional constraints the network property of interest may occasionally take values that cannot be observed in real networks, and in these cases the sampled vertex strengths may hence be unsuitable for the task of comparing an original finding to "random findings."

As a toy example, consider the network shown in Fig. 1, with six vertices and seven edges. This artificially generated toy network describes telephone calls between six individuals (the vertices) over an observation period of 24 h. An edge between two vertices represents the total cumulative call duration (in hours) between two individuals. Our goal is now to generate surrogate networks having exactly the same edges

*kai.puolamaki@helsinki.fi
†andreas.henelius@helsinki.fi
‡antti.ukkonen@helsinki.fi

Published by the American Physical Society under the terms of the Creative Commons Attribution 4.0 International license. Further distribution of this work must maintain attribution to the author(s) and the published article's title, journal citation, and DOI.



Vertex strengths

Vertex	1	2	3	4	5	6
Strength	13.5	5.5	12.0	17.0	8.0	13.0

FIG. 1. Network where persons are represented as vertices and phone calls between persons as edges, with the edge weight denoting the total call duration in hours between two persons. The table on the right shows the vertex strengths (sum of adjacent edge weights) for each vertex of the network.

as in the original network, i.e., we assume that people cannot communicate outside their friendship network and hence no new edges are created. We consider two different cases. First, we consider the case where the call duration between two individuals (the edge weights) can vary within an interval, but the vertex strengths (sum of edge weights adjacent to a vertex) must stay fixed at their original values, i.e., the total duration spent on the phone by every individual must remain the same. Second, we consider the case where both edge and vertex strengths are allowed to vary within a given interval. In both cases the interval width is constrained by the simple fact that during a 24-h period a person cannot spend more than 24 h on the phone in total.

Table I shows the range (min and max) of the edge and vertex strengths for 10 000 surrogate networks for the two cases described above, obtained using (1) the method presented in this paper, termed *CycleSampler*, and (2) a maximum-entropy model (described in Appendix A). In Case 1, the CycleSampler method preserves the vertex strengths exactly. (Notice that the range of vertex strengths matches the vertex strengths given in Fig. 1.) In Case 2 the CycleSampler method simultaneously preserves both edge and vertex strengths between 0 and 24 h, while the maximum-entropy method preserves these only in expectation. It is clear that the maximum-entropy model easily satisfies constraints on edge weights but violates the 24-h vertex strength constraint. This is because the edge weights are sampled i.i.d. and thus for vertices having a large degree the sum of sampled weights on adjacent edges can easily increase beyond the maximum value allowed.

This limitation of the maximum-entropy model is further highlighted in the numerical example presented in Fig. 2. This example shows that even in a very simple case the majority of surrogate networks from a maximum-entropy model will *not* satisfy hard constraints on vertex strengths, even if the expected value of vertex strengths is preserved. Furthermore, the resulting distribution of edge weights is not uniform. The CycleSampler method proposed in this paper, on the other hand, produces surrogate networks satisfying all constraints. We here examine the uniformity of the distributions in the given geometry. If some transformation was applied to the distributions shown in Fig. 2 to make one of these distributions uniform, then the other distribution would not be uniform using the same transformation.

Summary of contributions

In this paper we present the *CycleSampler* method, which is a *structure-preserving sampling method for edge weights that explicitly maintains vertex strengths within a given interval*. This interval can be set to have zero width, in which case the vertex strengths are maintained exactly in the generated surrogate networks. The approach can be viewed as a generalization of the property-preserving Markov chain Monte Carlo (MCMC) algorithm described in Ref. [2] to the structure-preserving case. However, the requirement to not introduce new edges or remove existing ones presents some nontrivial algorithmic challenges.

In short, our approach samples uniformly from the *null space* of the given network's *incidence matrix* (a binary matrix where vertices are rows and edges are columns), which requires constructing a basis for this null space. It is crucial that the basis is *sparse*, since the null space may be very high-dimensional (in the millions). Methods that require keeping a dense basis in memory, as found by textbook methods, may even be infeasible for larger networks. The problem of finding a sparse basis for general matrices has been studied previously [15,16], and some variants of it are NP-hard [17]. However, a sparse basis for the null space of an incidence matrix can be constructed very efficiently using a spanning tree of the original network [18]. We leverage this idea to devise an MCMC algorithm which is efficient and scalable and can be used to generate surrogate networks having millions of

TABLE I. Range of edge and vertex strengths of 10 000 surrogate networks generated using the CycleSampler method presented in this paper and the maximum-entropy method. In Case 1 vertex strengths are fixed, while in Case 2 edge and vertex strengths can vary. Note that edge and vertex strengths stay within the physically feasible interval [0, 24] hours for the CycleSampler method, whereas the maximum-entropy solution can lead to unfeasible vertex strengths.

	Case 1		Case 2	
Method	Edge weights	Vertex strengths	Edge weights	Vertex strengths
CycleSampler	[0.00, 12.00]	[5.50, 17.00]	[0.00, 23.40]	[0.00, 24.00]
Maximum entropy			[0.00, 24.00]	[0.00, 56.66]

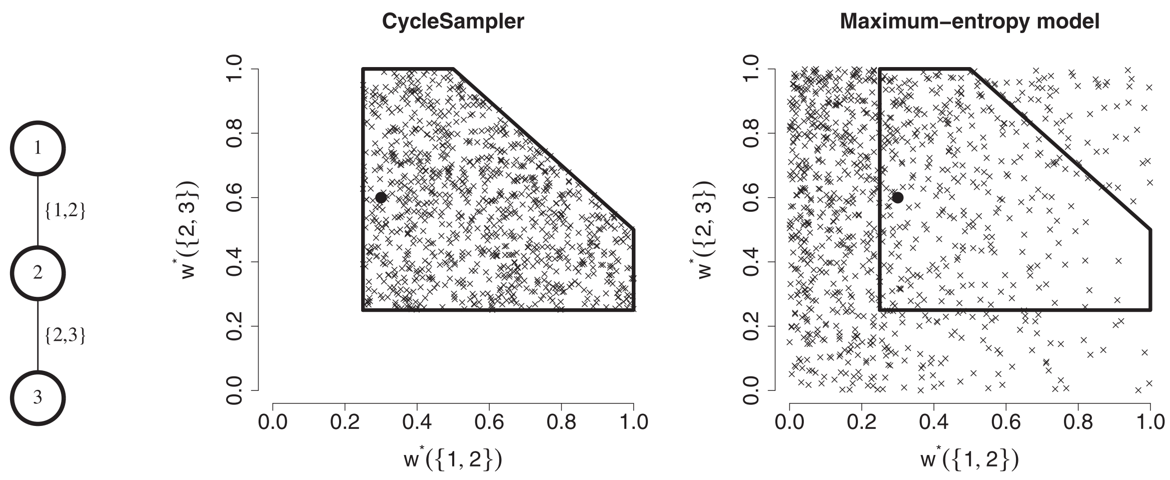


FIG. 2. An illustration of the difference between the CycleSampler method and the maximum-entropy model. Left: a graph with three vertices {1, 2, 3} and two edges with the observed weights $w(\{1, 2\}) = 0.3$ and $w(\{2, 3\}) = 0.6$, respectively. The vertex strengths are $W(1) = w(\{1, 2\}) = 0.3$, $W(2) = w(\{1, 2\}) + w(\{2, 3\}) = 0.9$, and $W(3) = w(\{2, 3\}) = 0.6$. We further assume that the edge weights are constrained to $w^*(e) \in [0, 1]$ and the vertex strengths to $W^*(v) \in [0.25, 1.5]$. Middle: edge weights w^* sampled using the CycleSampler method introduced in this paper. The black polygon encloses the set of feasible edge weights subject to the vertex strength constraints. The distribution of sampled edge weights is uniform, and both edge and vertex strengths satisfy the constraints. The black dot shows the observed (original) weights. Note that although we use the observed weights as the initial point of our Markov chain, the resulting distribution is asymptotically identical for all choices of the initial point as long as it satisfies the constraints. Right: the edge weights w^* from the maximum-entropy distribution over edge weights $p(w(\{1, 2\}), w(\{2, 3\}))$ such that the expected vertex strengths match the observed ones: $E_p[W^*(1)] = 0.3$, $E_p[W^*(2)] = 0.9$, and $E_p[W^*(3)] = 0.6$. In the maximum-entropy model there is no obvious way to simultaneously restrict both edge weights and vertex strengths to strict intervals, and most of the sampled edge weights are indeed outside the black polygon. Furthermore, the distribution of edge weights is generally not uniform in this geometry.

edges, while preserving vertex strengths as described above. We also present an empirical evaluation of the scalability of our algorithm in a number of real-world cases. We provide an open-source implementation of the CycleSampler method. The CycleSampler is implemented in C, as an extension to R [19], and is freely available for download [20].

II. THE CYCLESAMPLER ALGORITHM

In this section we describe the CycleSampler algorithm. We first formalize the problem considered here, after which we provide a high-level description of the algorithm. We then consider how this procedure works for undirected graphs. This discussion is followed by an example illustrating the procedure, after which we give a detailed description of the algorithms with proofs. We then describe how to the algorithm can also be extended to directed graphs. Finally, we provide a few notes concerning the practical implementation of the algorithm.

A. Problem definition

Let $G = (V, E)$ be a graph, where the m vertices are given by $V = [m]$, where we denote $[m] = \{1, \dots, m\}$, and the edges by $E \subseteq \cup_{v\in V} \cup_{v'\in V} \{\{v, v'\}\}$. The weight of an edge $e \in E$ is denoted $w(e) \in \mathbb{R}$. We assume that there are no self-loops in the observed graph, i.e., $|e| = 2$ for all $e \in E$.

In the following we assume that the graph G is connected. If the graph is not connected, the CycleSampler algorithm can be applied separately for each connected component, since these essentially represent separate graphs.

We use the neighborhood function $n(v)$, where $v \in V$, to represent the set of edges connected to a vertex v:

$$n(v) = \{e \in E \mid v \in e\}. \quad (1)$$

We define the *strength of a vertex* $v \in V$ as the sum of the weights of the edges connected to it:

$$W(v) = \sum_{e\in n(v)} w(e). \quad (2)$$

In colloquial terms, our task is to obtain a uniform sample of edge weights $w(e)$, such that the weight of every edge and the strength of every vertex remain within given intervals. This problem can be formally defined as follows.

Problem 1. Given a connected graph $G = (V, E)$ and a set of intervals $[a(e), b(e)]$ for each edge $e \in E$ and $[A(v), B(v)]$ for each vertex $v \in V$, respectively, such that $a(e) \leqslant w(e) \leqslant b(e)$ and $A(v) \leqslant W(v) \leqslant B(v)$, obtain a sample uniformly at random from the set of allowed edge weights $\mathcal{W}^*$, given by

$$\mathcal{W}^* = \left\{ w^* : E \mapsto \mathbb{R} \mid \forall e \in E \text{ it holds that } w^*(e) \in [a(e), b(e)] \text{ and } \forall v \in V \text{ it holds that } \sum_{e\in n(v)} w^*(e) \in [A(v), B(v)] \right\}. \quad (3)$$

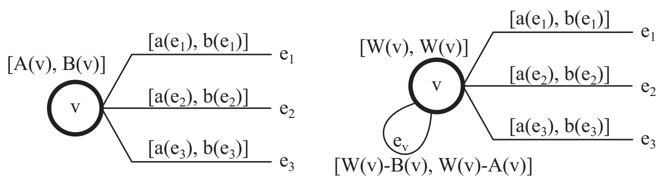


FIG. 3. In the graph on the left there are no self-loops and the vertex $v \in V$ has a range of allowed strengths $[A(v), B(v)]$. In the graph on the right the strength of vertex v is fixed to $W(v)$ and there is a self-loop $e_v = \{v\}$ with a range of allowed weights given by $[W(v) - B(v), W(v) - A(v)]$, where $W(v) = w(e_1) + w(e_2) + w(e_3)$. These two graphs are equivalent in the sense that the allowed ranges of the weights of the edges e_1, e_2, and e_3 as the vertex strength (without the self-loop) $w(e_1) + w(e_2) + w(e_3)$ are the same for both graphs. It follows that a set of allowed weights $\mathcal{W}^*$ given by Eq. (3) defined without self-loops and ranges for vertex strengths can be equivalently defined by graphs with self-loops and fixed vertex strengths.

Existing algorithms for generating surrogate networks cannot be directly applied to solve Problem 1 for two reasons. First, we aim to preserve network structure, i.e., we can only modify weights on existing edges, not introduce new edges. Second, we allow vertex strengths to vary within given intervals, while other approaches aim to maintain them either exactly or only in expectation.

B. Interval constraints on vertices

In Problem 1 we allow the strength of each vertex $v \in V$ to vary on the interval $[A(v), B(v)]$. This sampling problem becomes easier if these interval constraints are replaced with equality constraints, i.e., a variant of the problem where the vertex strengths are preserved *exactly*.

To do this, we define an equivalent graph containing self-loops (edges of type $e_v = \{v\}$) where the vertex strengths are fixed, i.e., $W(v) = A(v) = B(v)$. The variability in the vertex strengths is absorbed in the self-loops. This graph with self-loops can be constructed using the transformation shown in Fig. 3. Using this scheme, any graph with interval constraints on vertex strengths and no self-loops can be transformed to a graph with equality constraints on vertex strengths and self-loops and vice versa.

We can now rewrite Eq. (3) as follows:

$$\mathcal{W}^* = \left\{ w^* : E \mapsto \mathbb{R} \mid \forall e \in E \text{ it holds that } w^*(e) \in [a(e), b(e)] \text{ and } \forall v \in V \text{ it holds that } \sum_{e \in n(v)} w^*(e) = W(v) \right\}. \quad (4)$$

The problem hence becomes to obtain a uniform sample from the set $\mathcal{W}^*$ of Eq. (4). Note that in Eq. (4) the set of edges E now contains self-loops on those vertices that originally had interval constraints. In the following we therefore assume—without loss of generality— that there are only equality constraints on vertex strengths, i.e., $A(v) = B(v)$ for all $v \in V$.

C. High-level approach

We continue by outlining a sketch of our solution to Problem 1. At a high level our algorithm is a Markov chain Monte Carlo (MCMC) method similar to the algorithm proposed in Ref. [2]. It starts from the observed set of edge weights, introduces a small perturbation to a few of these at every step and runs until convergence. Let C denote a *collection of subsets of* E, i.e., every $E' \in C$ is some (small) set of edges. The algorithm in Ref. [2] as well as ours can be sketched within a common framework as follows:

(1) Initially, let the current state be the observed set of edge weights.

(2) Select some $E' \in C$ uniformly at random.

(3) Perturb the weights of every edge in E' so that all constraints remain satisfied. (Exactly how this is done is described in detail below.)

(4) Repeat steps 2–3 until convergence.

The main difference between the method in Ref. [2] and the method presented here concerns what the collection C contains. It is crucial to make sure that C is constructed such that the resulting Markov chain indeed converges to a uniform distribution over $\mathcal{W}^*$. This is shown below in Theorem 1 in Sec. II F.

In the algorithm of Ref. [2], C contains all possible cycles of length four. Since in Ref. [2] the underlying graph is assumed to be a clique, there are plenty of such cycles, and it can be shown that these are enough for the Markov chain to reach a uniform distribution. Also, it is fairly easy to see that in cycles of length four the edge weights can always be adjusted in a simple manner so that all constraints remain satisfied. However, in our case finding a suitable C is *complicated by the requirement of not introducing new edges*. Simply choosing all cycles of length four from G is not enough. In the remainder of the paper we discuss our main technical contribution: *an approach for constructing C in general undirected graphs (not only cliques) so that the resulting Markov chain converges to the uniform distribution over* $\mathcal{W}^*$. In addition, we show how a simple transformation of the input graph allows us to *extend the approach also to directed graphs*.

D. Solution for undirected graphs

In this section we first describe the solution for general undirected graphs. In Sec. II G below we describe the solution for directed graphs. We assume for simplicity of discussion and without loss of generality that the input graph G consists of a single connected component. (In general, we can independently sample each of the connected components of the graph in the sampling process introduced later). The problem of constructing a suitable collection C of edges becomes easier if we view our sampling problem in terms of systems of linear equations. In this way we can express our problem using known concepts from linear algebra.

As a first step, define the *incidence matrix* $\mathbf{A} \in \{0,1\}^{|V|\times|E|}$ of the graph G in the usual manner as $\mathbf{A}_{ve} = 2I(v \in e)/|e|$. Here $v \in V$ and $e \in E$ and $I(\square)$ is an indicator function which equals unity if $\square$ is true and is zero otherwise. Also, let $\mathbf{W} \in \mathbb{R}^{|V|}$ denote the vector of observed vertex strengths defined by $\mathbf{W}_v = W(v)$ for all $v \in V$, and denote by $\mathbf{w}^* \in \mathbb{R}^{|E|}$ the vector of edge weights. Given these, sampling uniformly from $\mathcal{W}^*$ of Eq. (4) is equivalent to the problem of sampling uniformly from the set

$$\mathcal{S} = \{\mathbf{w}^* \in \mathbb{R}^{|E|} \mid \mathbf{A}\mathbf{w}^* = \mathbf{W} \ \text{ and } \ \forall e \in E \text{ it holds that } \mathbf{w}^*_e \in [a(e), b(e)]\}. \tag{5}$$

By our assumption the original observed weight vector $\mathbf{w}$ is in $\mathcal{W}^*$. It follows that $\mathbf{A}\mathbf{w} = \mathbf{W}$ and therefore $\mathbf{w} \in \mathcal{S}$.

For the moment, let us focus only on the underdetermined linear system $\mathbf{A}\mathbf{w}^* = \mathbf{W}$. A simple known property of such systems is that their solution space can be expressed as a sum of a single known solution such as $\mathbf{w} \in \mathcal{S}$ and *any* vector $\mathbf{x}$ from the *null space* of $\mathbf{A}$. The null space of $\mathbf{A}$, denoted by $\mathrm{Null}(\mathbf{A})$, is defined by the set $\mathrm{Null}(\mathbf{A}) = \{\mathbf{x} \in \mathbb{R}^{|E|} \mid \mathbf{A}\mathbf{x} = \mathbf{0}\}$. It is easy to see that $\mathbf{A}\mathbf{w}^* = \mathbf{W}$, where $\mathbf{w}^* = \mathbf{w} + \mathbf{x}$, for any $\mathbf{x} \in \mathrm{Null}(\mathbf{A})$. Because of the constraints on edge weights, we cannot simply use *any* $\mathbf{x} \in \mathrm{Null}(\mathbf{A})$. Instead, $\mathbf{x}$ must come from a *convex subset* of $\mathrm{Null}(\mathbf{A})$. Therefore, the problem of sampling uniformly from $\mathcal{S}$ is equivalent to the problem of sampling uniformly from said convex subset of $\mathrm{Null}(\mathbf{A})$.

This is a known problem and could be solved using textbook methods, such as those described, e.g., in Ref. [21]. Those approaches, however, must usually compute a *basis* for $\mathrm{Null}(\mathbf{A})$, which in general is a dense matrix of size $|E| \times \dim[\mathrm{Null}(\mathbf{A})]$, where the cardinality of the null space of $\mathbf{A}$ is in the same order of magnitude as $|E|$. While this is not a problem as long as the incidence matrix $\mathbf{A}$ is fairly small, methods that require storing such a matrix may become infeasible for very large networks. However, since $\mathbf{A}$ is the incidence matrix of a network, a *sparse basis* is easily constructed by combining *cycles* of G, as shown, e.g., in Ref. [18].

In short, this works as follows. We first find a *spanning tree* T of G. Every edge that does not belong to T clearly induces a cycle when combined with edges in T. Given T, every such even-length cycle is directly an element of the basis, while odd-length cycles are paired together to form elements of the basis until it is complete. Details of the algorithm are given in Sec. II F below. Note that such a sparse basis is easily represented by a collection C of subsets of edges, together with appropriate weights for every edge.

E. Illustrating example

In this section we provide an example illustrating the above discussed concepts. Consider again the network introduced above in Fig. 1, with six vertices $V = \{1,2,3,4,5,6\}$ and seven edges. We now add two self-loops to vertices 1 and 6 in this network, giving the nine edges

$$E = \{\{1\},\ \{1,2\},\ \{1,3\},\ \{1,6\},\ \{2,3\},\ \{3,4\},\ \{4,5\},\ \{4,6\},\ \{6\}\}.$$

Further assume that the edges in the spanning tree of this graph are given by

$$E_s = \{\{1,3\},\ \{2,3\},\ \{3,4\},\ \{4,5\},\ \{4,6\}\}$$

and the edges not in the spanning tree by

$$F = \{\{1\},\ \{1,2\},\ \{1,6\},\ \{6\}\}.$$

The root vertex is given by $v_{\mathrm{root}} = 3$. This graph is shown in Fig. 4 (the root of the spanning tree is marked with gray), and the corresponding matrix $\mathbf{A}$ is shown in Table II. The cycle vectors are shown in Fig. 5 and the basis of the null space in Fig. 6. A different root vertex could be used when constructing the spanning tree, which would lead to a different set of vectors that span the null space; however, any choice of spanning tree or root vertex will span the same null space.

In Sec. II F we introduce the terms *clean edge* and *dirty edge* when discussing cycles, and these are also used in

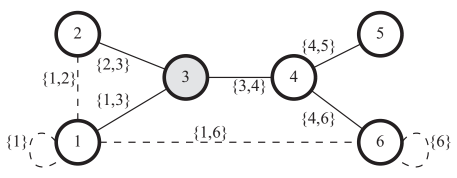


FIG. 4. Example graph with six vertices and nine edges. The five edges in the spanning tree are shown with solid lines and the four edges not in the spanning tree with dashed lines. Vertex 3 is the root vertex of the spanning tree (marked with gray).

TABLE II. The six uppermost rows show the incidence matrix matrix $\mathbf{A}$ for the graph in Fig. 4. The next four rows show the graph cycles $\mathbf{c}_i$, which are also shown graphically in Fig. 5; see Eq. (6) for the definition. The four lowermost rows show the basis vectors $\mathbf{y}_i$, also shown graphically in Fig. 6; see Eqs. (9) and (10) for the definition. The basis vectors have been constructed from the cycles as follows: the basis vector $\mathbf{y}_a$ by an even cycle induced by the clean edge, $\mathbf{y}_a = \mathbf{c}_a$, and the remaining two basis vectors $\mathbf{y}_{bc}$ and $\mathbf{y}_{cd}$ by linear combinations of two odd cycles induced by the dirty edges, $\mathbf{y}_{bc} = \mathbf{c}_b - \mathbf{c}_c$, and $\mathbf{y}_{cd} = \mathbf{c}_c + \mathbf{c}_d$. All of the basis vectors satisfy $\mathbf{A}\mathbf{y}_i = \mathbf{0}$, and span a three-dimensional null space $\mathrm{Null}(\mathbf{A})$, as required by Theorem 2.

$\mathbf{A}$	{1}	{1, 2}	{1, 3}	{1, 6}	{2, 3}	{3, 4}	{4, 5}	{4, 6}	{6}
1	2	1	1	1	0	0	0	0	0
2	0	1	0	0	1	0	0	0	0
3	0	0	1	0	1	1	0	0	0
4	0	0	0	0	0	1	1	1	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	1	0	0	0	1	2
$\mathbf{c}_a^T$	0	0	−1	1	0	1	0	−1	0
$\mathbf{c}_b^T$	1	0	−2	0	0	0	0	0	0
$\mathbf{c}_c^T$	0	1	−1	0	−1	0	0	0	0
$\mathbf{c}_d^T$	0	0	0	0	0	2	0	−2	1
$\mathbf{y}_a^T$	0	0	−1	1	0	1	0	−1	0
$\mathbf{y}_{bc}^T$	1	−1	−1	0	1	0	0	0	0
$\mathbf{y}_{cd}^T$	0	1	−1	0	−1	2	0	−2	1

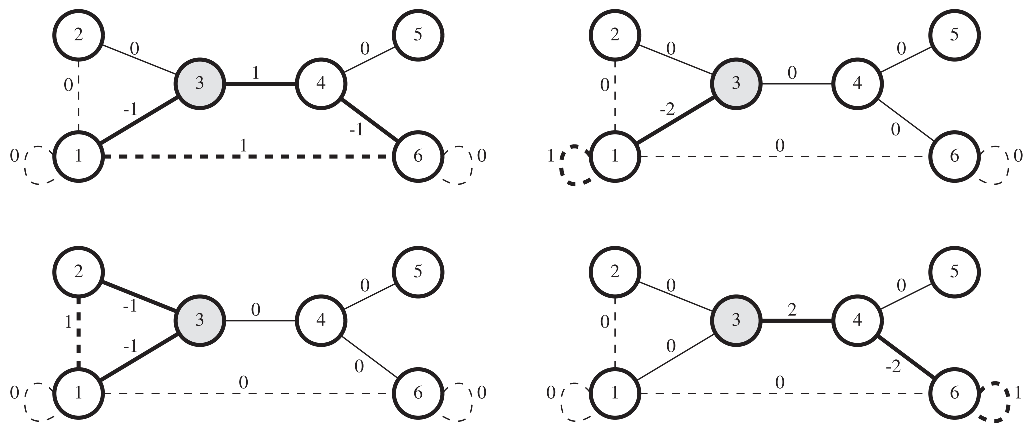


FIG. 5. The cycles of the graph in Fig. 4 and the corresponding values of the respective vector $\mathbf{c} \in \mathbb{R}^{|E|}$. There is one even cycle related to a clean edge ($\mathbf{c}(\{1,6\})$ in the top left graph, and three odd cycles related to dirty edges: $\mathbf{c}(\{1\})$ in the top right graph, $\mathbf{c}(\{1,2\})$ in the bottom left graph, and $\mathbf{c}(\{6\})$ in the bottom right graph.

Table II, Fig. 5, and Fig. 6. In brief, a clean edge in the graph induces a cycle of even length (containing an even number of edges), while a dirty edge induces a cycle of odd length. Combining two odd cycles, each induced by a dirty edge, yields a cycle of even length.

F. Details of the algorithm

In this section we give a detailed proof of the proposed algorithm, an overview of which is given above.

The CycleSampler Algorithm

Assume we have vectors $\mathbf{y}_1, \ldots, \mathbf{y}_l$ that span the null space Null($\mathbf{A}$), i.e., any null space vector $\mathbf{x} \in$ Null($\mathbf{A}$) can be formed as a linear combination of these vectors. The vectors $\mathbf{y}_1, \ldots, \mathbf{y}_l$ thus form a *basis* of Null($\mathbf{A}$). Given this basis, we can obtain surrogate networks as follows:

(1) First, transform the graph to a form where the vertex strengths are fixed and the variability in them is described by self-loops, as in Fig. 3.

(2) Initially, let the current state be the observed set of edge weights, $\mathbf{w}^* \leftarrow \mathbf{w}$, with the weight of self-loops initially set to zero.

(3) Pick a basis vector $\mathbf{y}_i$ at random and let $[a, b]$ be the largest range of allowed values such that $\mathbf{w}^* + \alpha\mathbf{y}_i$, where $\alpha \in [a, b]$, stays within $\mathcal{W}^*$. Sample α uniformly at random from $[a, b]$.

(4) Update $\mathbf{w}^* \leftarrow \mathbf{w}^* + \alpha\mathbf{y}_i$ and repeat from step 3 above.

Note that because $\mathcal{W}^*$ is a simple convex space—an $|E|$-dimensional rectangle—we can find $[a, b]$ for a given $\mathbf{y}_i$ efficiently by a simple loop over the nonzero dimensions of $\mathbf{y}_i$. The updates at step 4 form a Markov chain of edge weight vectors $\mathbf{w}^*$.

Theorem 1. The CycleSampler algorithm asymptotically provides (after a sufficient number of iterations) surrogate networks uniformly from the set $\mathcal{W}^*$.

Proof. This follows from the facts that (1) because $\mathbf{y}_i$ span the null space, then all points of the null space are reachable

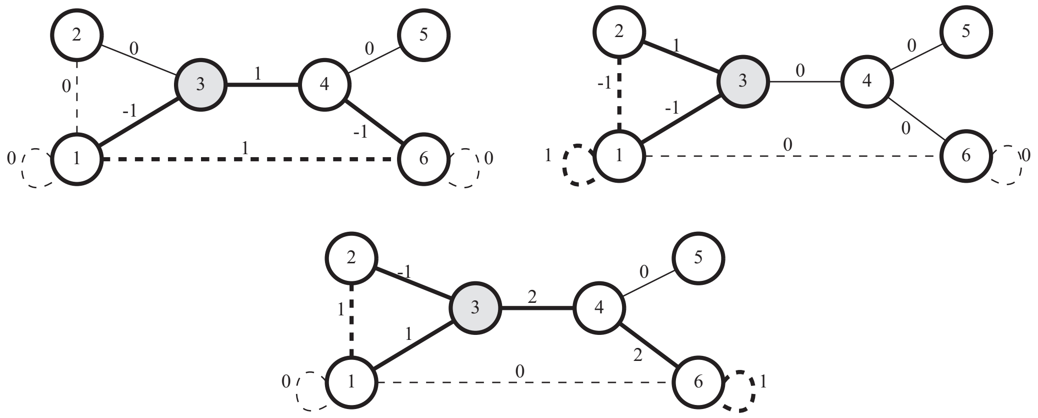


FIG. 6. The basis constructed from the cycles in Fig. 5 and the corresponding values of the respective vector $\mathbf{y}_i \in \mathbb{R}^{|E|}$. There is one basis vector corresponding to the clean edge $\{1, 6\}$ in the top left graph, and two basis vectors corresponding to pairs of dirty edges: $\{1\}$ and $\{1, 2\}$ in the top right graph, and $\{1, 2\}$ and $\{6\}$ in the bottom graph. Each of these basis vectors multiplied by matrix $\mathbf{A}$ of Table II yields zero, and therefore the basis vectors are in the null space Null($\mathbf{A}$). The basis vectors are also given by the bottom rows of Table II. All of the introduced graph cycles defined by edges with nonzero weights are of even length: (top left) $1-3-4-6(-1-\cdots)$, (top right) $1-1-2-3(-1-\cdots)$, and (bottom) $1-2-3-4-6-6-4-3(-1-\cdots)$; notice that edges with the weight of ± 2 are traversed twice in a graph cycle, once in each direction.

with a nonvanishing probability and that (2) the transition probability from state $\mathbf{w}^* \in \mathcal{S}$ to state $\mathbf{w}^{*\prime} \in \mathcal{S}$ is equal to the transition probability from state $\mathbf{w}^{*\prime}$ to state $\mathbf{w}^*$. The last statement follows from the fact that in step 2 of the Cycle-Sampler algorithm the vectors $\mathbf{y}_i$ are chosen with uniform probability, after which in step 3 the number $\alpha \in [a, b]$ is chosen uniformly; i.e., if we transitioned in one step from state $\mathbf{w}_k$ to state $\mathbf{w}_l$ using $\mathbf{y}_i$ and $\alpha = \alpha_k$ chosen from an interval $[a_k, b_k]$, we could in the next step with equal probability as in the first step transition from state $\mathbf{w}_l$ to state $\mathbf{w}_k$ if $\mathbf{y}_i$ was chosen again and if $\alpha = -\alpha_l$ chosen from an interval $[a_l, b_l]$. Here it is important to note that for two consecutive steps k and l it holds that the width of the intervals $[a_k, b_k]$ and $[a_l, b_l]$ are equal, and since α is sampled uniformly at random from these intervals, the probability of α_l in the first step equals that of $-\alpha_l$ in the second step.

In other words, the probability of moving from state k to state l equals that of moving from state l to state k and the Markov chain is therefore *reversible*. ■

It remains to find a complete basis $\mathbf{y}_1, \ldots, \mathbf{y}_l$ of the null space Null($\mathbf{A}$). This can be done using a spanning tree of the connected graph G.

We denote by $v_{\text{root}} \in V$ the root vertex of the spanning tree. Also, we denote by $E_s \subseteq E$ the $|V| - 1$ edges that appear in the spanning tree and by $F = E \setminus E_s$ the remaining $|E| - |V| + 1$ edges that do not appear in the spanning tree.

We further denote by $E(v) \subseteq E_s$ the set of edges in the spanning tree between vertex $v \in V$ and the root vertex. For the root vertex v_{root} we have $E(v_{\text{root}}) = \emptyset$. We call the number of edges between $e \in E_s$ and the root vertex in the spanning tree the *depth* of e, denoted by depth(e). The edges adjacent to the root vertex have a depth of zero. We further define the depth of vertex v to be its distance from the root vertex, i.e., $\text{depth}(v) = |E(v)|$.

Recall that every edge in the set F induces a cycle of G. Next, we represent these cycles as vectors in $\mathbb{R}^{|E|}$. Let $\mathbf{n}(e) \in \{0, 1\}^{|E|}$ denote a 0-1 vector defined by $\mathbf{n}(e)_{e'} = I(e = e')$ that represents the edge $e \in E$. We define the *cycle* induced by the edge $\{v, v'\} \in F$ as the vector $\mathbf{c}(v, v') \in \mathbb{R}^{|E|}$ given by

$$\mathbf{c}(v, v') = \mathbf{n}(\{v, v'\}) + \sum_{e \in E(v)} (-1)^{\text{depth}(v)+\text{depth}(e)}\mathbf{n}(e) + \sum_{e' \in E(v')} (-1)^{\text{depth}(v')+\text{depth}(e')}\mathbf{n}(e'). \tag{6}$$

The above sum essentially traverses the cycle induced by $\{v, v'\} \in F$ and assigns to each edge a weight the sign of which depends on its distance from the root.

We further split the edges not in the spanning tree (the set F) into *clean edges*,

$$F_c = \{\{v, v'\} \in F \mid \text{depth}(v) + \text{depth}(v') \text{ is odd}\}, \tag{7}$$

and *dirty edges*,

$$F_d = \{\{v, v'\} \in F \mid v = v' \text{ or } (\text{depth}(v) + \text{depth}(v') \text{ is even})\}. \tag{8}$$

Notice that the set of clean edges F_c cannot contain self-loops, but the set of dirty edges F_d may contain self-loops (i.e., $v = v'$). The nonzero elements of a cycle introduced by clean edges contain graph loops with an even number of edges, while a cycle introduced by a dirty edge contains a graph loop with an odd number of edges. Indeed, by taking a spanning tree of a graph and adding an edge not in the graph we always get a unique *graph cycle*. Those graph cycles form the *cycle basis* of the graph. In particular, it is well known that the cycle basis of a graph contains only even-length cycles if and only if the graph is bipartite. Therefore $F_d = \emptyset$ is equivalent to the statement that the graph G is bipartite.

We construct a basis for the null space as follows.

(1) For each clean edge $e \in F_c$ we define a basis vector by the respective cycle,

$$\mathbf{y}_i = \mathbf{c}(e). \tag{9}$$

For an example, see Fig. 6(a). The nonzero elements of the basis vector $\mathbf{y}_i$ form a graph cycle of even length, with alternating weights of ± 1.

(2) Dirty edges in F_d are dealt with slightly differently. We always take *two* edges from F_d and combine the respective cycles to form a basis vector. In particular, assume the edges in F_d are arranged in an arbitrary order, numbered by $1, \ldots, |F_d|$. A basis vector is then formed by taking two adjacent edges and combining their respective cycles as follows:

$$\mathbf{y}_i = \mathbf{c}(e_i) - (-1)^{\text{depth}(e_i)+\text{depth}(e_{i+1})}\mathbf{c}(e_{i+1}), \tag{10}$$

where $i \in [|F_d| - 1]$. For an example, see Figs. 6(b) and 6(c). Again, the nonzero elements of the basis vector $\mathbf{y}_i$ form a graph cycle of even length.

The number of distinct basis vectors that could be defined by Eqs. (9) and (10), is therefore $|F_c| + |F_d|(|F_d| - 1)/2$.

We show that the basis vectors of Eqs. (9) and (10) form a complete basis of the null space Null($\mathbf{A}$) by first proving the following three lemmas.

Lemma 1. The basis vectors (cycles) defined by Eq. (9) are in the null space Null($\mathbf{A}$), and they span an $|F_c|$ dimensional space.

Proof. A vector $\mathbf{y}_i$ defined by Eq. (9) is in the null space Null($\mathbf{A}$), because the equation $\mathbf{A}\mathbf{y}_i = \mathbf{0}$ is satisfied for all $\mathbf{y}_i$.

The vectors $\mathbf{y}_i$ are clearly linearly independent, because each of the vectors contains a unique nonzero dimension given by an edge $e \in F_c$ which is zero in all other vectors. Therefore, the $|F_c|$ vectors span an $|F_c|$ dimensional space. ■

Lemma 2. If there are at least two dirty edges, i.e., $|F_d| \geqslant 2$, the basis vectors (cycles) defined by Eq. (10) are in the null space Null($\mathbf{A}$), they span an $|F_d| - 1$ dimensional space, and they cannot be expressed as linear combinations of the vectors defined by Eq. (9).

Proof. First, assume that there are at least two dirty edges, because otherwise there are no pairs and hence no vectors defined by Eq. (10). If there is exactly one dirty edge, then this edge cannot be used to form a vector in the null space. A vector (cycle) defined by Eq. (10) is in the null space, because $\mathbf{A}\mathbf{y}_i = \mathbf{0}$ is satisfied.

The vectors span an $|F_d| - 1$ dimensional subspace, because as described above, the vectors are formed by taking all $|F_d| - 1$ adjacent pairs of edges e_i and e_{i+1} from F_d when they are arranged in an arbitrary order, and combining their respective cycles according to Eq. (10). Also, a combination of such vectors is independent of the previous combinations, because it contains a nonzero value for the edge $e_{i+1} \in F_d$ that has a zero value for all of the previous combinations.

A vector $\mathbf{y}_i$ defined by Eq. (10) cannot be expressed as a linear combination of the vectors defined by Eq. (9), because the vector contains a nonzero element for two edges in F_d that do not occur in any of the vectors defined by Eq. (9). ■

Lemma 3. The dimensionality of the null space of **A**, Null(**A**), is $|F_c|$ if there are no dirty edges and $|F_c| + |F_d| - 1$ if there are dirty edges.

Proof. Consider the rank of the incidence matrix $\mathbf{A} \in \{0, 1\}^{|V|\times|E|}$ of the graph $G = (V, E)$. The rank of this matrix equals at most the number of rows in the matrix, i.e., $\text{rank}(\mathbf{A}) \leqslant |V|$. If and only if there is a nonzero vector $\mathbf{v} \in \mathbb{R}^{|V|}$ such that $\mathbf{v}^T\mathbf{A} = \mathbf{0}$, then $\text{rank}(\mathbf{A}) < |V|$, otherwise $\text{rank}(\mathbf{A}) = |V|$.

The vector **v** must satisfy the following two properties. (1) The element in **v** corresponding to a vertex with a self-loop must be zero, i.e., $\mathbf{v}_i = 0$, because the column (edge) representing a self-loop has only one nonzero value. (2) The elements in **v** corresponding to a pair of vertices connected by an edge $\{i, j\} \in E$ must have opposite signs, i.e., $\mathbf{v}_i = -\mathbf{v}_j$, otherwise the column (edge) in the matrix product $\mathbf{v}^T\mathbf{A}$ is nonzero.

Because the graph is connected we can construct a vector **v** simply by starting from one row (vertex), e.g., $i = 1$ and setting $\mathbf{v}_1 \leftarrow x$, where x is some number. We can hence iteratively follow any path in the graph and assign values for the remaining rows in **v** in accordance with the above described properties. The elements in **v** are hence either x or $-x$.

Consider first the case when there are dirty edges, i.e., $F_d \neq \emptyset$, and there is at least one cycle in the graph with an odd number of edges. If we follow this cycle it leads to the situation where $x = -x$, meaning that no nonzero vector **v** exists and $\mathbf{v} = \mathbf{0}$ is the only viable solution and hence $\text{rank}(\mathbf{A}) = |V|$. According to the rank-nullity theorem the rank of the null space is $|E| - \text{rank}(\mathbf{A}) = |E| - |V| = |F| - 1 = |F_c| + |F_d| - 1$, which proves the lemma for the case $F_d \neq \emptyset$.

Consider now the case when there are no dirty edges, i.e., $F_d = \emptyset$ and all graph cycles are of even length. A graph with only even cycles must be bipartite, i.e., the vertices form two disjoint sets. We label all vertices in one set by $+1$ and all vertices in the other set by -1.

We now construct the vector **v** according to this labeling and property (2) above, i.e., the elements in **v** corresponding to two vertices i and j must have opposite signs. We can hence find a vector **v** such that $\mathbf{v}^T\mathbf{A} = \mathbf{0}$ is satisfied and it follows that $\text{rank}(\mathbf{A}) \leqslant |V| - 1$. Next, we consider the rank of a matrix **A** with the first row removed, denoted by $\mathbf{A}'$. This matrix contains (because there are no isolated vertices) at least one column containing only one nonzero entry. The rank of this matrix is therefore at least $|V| - 1$, from which it follows that the $\text{rank}(\mathbf{A}) = |V| - 1$. According to the rank-nullity theorem the rank of the null space is $|E| - \text{rank}(\mathbf{A}) = |E| - |V| + 1 = |F| = |F_c|$, proving the lemma for the case $F_d = \emptyset$. ■

The following theorem follows directly from Lemmas 1, 2, and 3 above.

Theorem 2. The basis vectors defined by Eqs. (9) and (10) span the null space Null(**A**) for a connected graph $G = (V, E)$. The dimensionality of the null space is $|E| - |V| + 1$ if the graph G is bipartite and $|E| - |V|$ otherwise.

G. Solution for directed graphs

Next we show that the above discussed solution to the sampling problem for undirected graphs (Problem 1) can also be applied to directed graphs, by first transforming the directed graph into an equivalent undirected graph. The algorithm proposed in this paper can therefore directly be used to sample edge weights for both directed and undirected graphs.

Let $G_D = (V_D, E_D)$ be a directed graph, where the m_D vertices are given by $V_D = [m_D]$ and the edges by $E_D \subseteq V \times V$. The weight of the directed edge $e \in E_D$ is denoted by $w_D(e) \in \mathbb{R}$. We define the *outgoing* edges of vertex $v \in V_D$ as

$$n_o(v) = \{(v', v'') \in E_D \mid v' = v\}, \tag{11}$$

and the *incoming* edges as

$$n_i(v) = \{(v', v'') \in E_D \mid v'' = v\}. \tag{12}$$

The outgoing strength of a vertex is given by

$$W_o(v) = \sum_{e\in n_o(v)} w_D(e), \tag{13}$$

and the incoming strength by

$$W_i(v) = \sum_{e\in n_i(v)} w_D(e). \tag{14}$$

We are now ready to define the sampling problem for directed graphs.

Problem 2. Given a connected directed graph $G_D = (V_D, E_D)$ and a set of intervals $[a_D(e), b_D(e)]$ for each edge $e \in E_D$ and $[A_o(v), B_o(v)]$ and $[A_i(v), B_i(v)]$ for each vertex $v \in V_D$, respectively, such that $a_D(e) \leqslant w_D(e) \leqslant b_D(e)$, $A_o(v) \leqslant W_o(v) \leqslant B_o(v)$, and $A_i(v) \leqslant W_i(v) \leqslant B_i(v)$, obtain a sample uniformly at random from the set of allowed weights $\mathcal{W}_D^*$, given by

$$\begin{aligned}\mathcal{W}_D^* = \{w_D^* : E_D \mapsto \mathbb{R} \ \ &| \ \forall e \in E_D \text{ it holds that } w_D^*(e) \in [a_D(e), b_D(e)] \quad \text{and} \\ &\forall v \in V_D \text{ it holds that } \sum_{e\in n_o(v)} w_D^*(e) \in [A_o(v), B_o(v)] \quad \text{and} \\ &\forall v \in V_D \text{ it holds that } \sum_{e\in n_i(v)} w_D^*(e) \in [A_i(v), B_i(v)]\end{aligned} \tag{15}$$

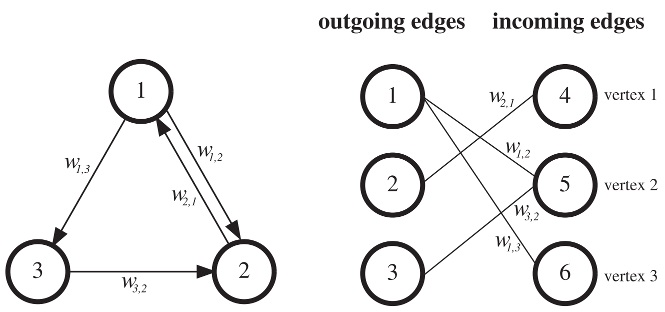


FIG. 7. Transformation of the directed graph on the left to the undirected bipartite graph on the right. The weight of an edge from i to j is denoted by $w_{i,j}$. In the undirected bipartite graph, the vertices on the right are labeled to indicate to which vertex in the original directed graph they correspond.

We can solve Problem 2 by the algorithm used to solve Problem 1 by noticing that a directed graph can be easily transformed to an equivalent undirected graph, as stated by the following theorem and as illustrated in Fig. 7.

Theorem 3. The set of allowed weights $\mathcal{W}_D^*$ of Eq. (15) for a directed graph G_D is equivalent to the set of allowed weights $\mathcal{W}^*$ of Eq. (3) for an undirected bipartite graph G when the graph G is defined as follows. The graph G has $m = 2m_D$ vertices, i.e., $V = [m]$. The set of undirected edges E of G is given by $E = \{\{v', v'' + m_D\} \mid (v', v'') \in E_D\}$. We define a mapping $f{:}E \mapsto E_D$ as follows, $f(\{v', v'' + m_D\}) = (v', v'')$ for all $(v', v'') \in E_D$. The weight of an edge $e \in E$ is given by $w(e) = w_D(f(e))$ and the bounds by $a(e) = a_D(f(e))$ and $b(w) = b_D(f(e))$. The vertices $1, \dots, m_D$ correspond to outgoing strengths and the vertices $m_D + 1, \dots, 2m_D$ to incoming strengths as follows:

$$W(v) = \begin{cases} W_o(v), & v \leqslant m_D \\ W_i(v - m_D), & v > m_D \end{cases}, \tag{16}$$

with the bounds given by

$$A(v) = \begin{cases} A_o(v), & v \leqslant m_D \\ A_i(v - m_D), & v > m_D \end{cases}, \tag{17}$$

and

$$B(v) = \begin{cases} B_o(v), & v \leqslant m_D \\ B_i(v - m_D), & v > m_D \end{cases}. \tag{18}$$

Now, if w^* is a uniform sample from $\mathcal{W}^*$ we can obtain a uniform sample w_D^* from $\mathcal{W}_D^*$ in a straightforward way by setting $w_D^*(f(e)) \leftarrow w^*(e)$ for all $e \in E$.

Proof. The proof follows directly from the definitions. ■

H. Implementation notes

We make some observations that are useful when implementing the method described above. First, it is sufficient to consider a *generating set* rather than a proper basis (a *minimal generating set*). This means that instead of fixing an arbitrary order for the edges in F_d and then applying Eq. (10) as described above, we can always pick *any two* edges from F_d, and take their linear combination. This has the upside that we are not committed to some possibly poor choice of the order for F_d but are free to consider a wider selection of vectors that are all guaranteed to span the desired subspace.

Second, while it is in theory possible to use *any* spanning tree of G, we have empirically made the following observation for different data sets. The mixing of the Markov chains, in terms of the l^2 norm between the starting state of the sampler and the jth surrogate, appears to be faster if the spanning tree is constructed to emphasise high-strength vertices. We have hence chosen to construct the spanning tree as follows. The vertex with the highest vertex strength is chosen as the root vertex $v_{\text{root}} \in V$. We then use a standard breadth-first search over the vertices sorted in descending order of vertex strength (i.e., going from vertices with high strength to low strength).

Note that the generation of surrogate data sets may be performed using, e.g., the method outlined in Ref. [2], which efficiently generates samples with small statistical dependencies.

III. EXPERIMENTAL EVALUATION

To demonstrate the method described in this paper we perform two analyses. First, we consider the scalability of the CycleSampler method. Second, we apply the CycleSampler algorithm to generate surrogates networks for investigating clustering coefficients in a data set describing trade relations between countries.

A. Analysis of scalability

To demonstrate the scalability of the method described in this paper we perform two experiments on seven publicly available sparse real-world networks. These networks are all examples of *recommendation data sets* (note that this does not limit the generality of the discussion). In recommendation data, a *user* provides a *rating* for a given *item*, i.e., the data items are triplets of the form (user, item, rating). We construct networks from these data sets as follows. Each *user* and *item* represents a vertex, and the rating given by a user to an item represents an edge, with a weight equal to the *rating*. Since users rate only items (and not other users), these networks are all bipartite with the vertex partitions given by users and items.

The properties of the networks are presented in Table III. The table shows the dimensions of the networks in terms of the number of rows (users) and columns (items) in the data matrix. The density of all networks is very low, meaning that the networks are sparse. The table also shows the number of edges and vertices in the network and the dimensionality of the null space. The preprocessing of the data sets is described in Appendix B.

When investigating varying properties of recommendation data sets it makes sense to place certain restrictions on (1) the sum of the ratings given by a user to all items and (2) the sum of all ratings received by an item from all users.

In the *first experiment* we generate surrogate networks where the vertex strengths are preserved exactly, while the edge weights $w(e)$ are allowed to vary on an interval corresponding to the range of the edge weights in the original network. In the context of the recommender systems this means that the total ratings given by a user and the total ratings received by an item are both preserved exactly (i.e., the ratings for a given user are just allocated differently).

TABLE III. Properties of the networks. The networks are sorted in order of an increasing number of edges. The columns are as follows: *rows* and *columns* give the full size of the data matrix, and *density* is the number of nonzero entries. The number of edges, vertices, and the dimensionality of the null space (in experiment 1) are given by $|E|$, $|V|$, and $|C| = |E| - |V| + 1$, respectively. In experiment 2 the dimensionality of the null space is $|E| + 1$ due to the addition of one self-loop per vertex. The initialization time for the sampler (e.g., finding the spanning tree and enumerating cycles) and the time needed to take a cycle step (a number of steps equal to the dimensionality of the null space of a given network) are shown in the columns t_{init} and t_{sample}. The times are in seconds, and the subscript *1* refers to experiment 1 whereas the subscript *2* refers to experiment 2.

	Rows	Columns	Density	$\lvert E\rvert$	$\lvert V\rvert$	$\lvert C\rvert$	$t_{\text{init},1}$	$t_{\text{sample},1}$	$t_{\text{init},2}$	$t_{\text{sample},2}$
Last.fm[a]	1.89×10^3	1.74×10^4	2.69×10^{-3}	8.86×10^4	1.93×10^4	6.93×10^4	1.17	0.02	1.40	0.04
MovieLens 100k[b]	9.43×10^2	1.68×10^3	6.30×10^{-2}	1.00×10^5	2.62×10^3	9.74×10^4	0.71	0.02	0.77	0.03
BookCrossing[c]	7.78×10^4	1.86×10^5	3.00×10^{-5}	4.34×10^5	2.64×10^5	1.85×10^5	36.50	0.16	53.86	0.61
FineFoods[d]	2.56×10^5	7.43×10^4	2.95×10^{-5}	5.61×10^5	3.30×10^5	2.53×10^5	62.15	0.23	99.97	0.93
MovieLens 1M[b]	6.04×10^3	3.71×10^3	4.47×10^{-2}	1.00×10^6	9.75×10^3	9.90×10^5	6.88	0.50	7.22	0.50
MovieLens 20M[b]	1.38×10^5	2.67×10^4	5.40×10^{-3}	2.00×10^7	1.65×10^5	1.98×10^7	155.05	15.41	162.47	18.08
TasteProfile[e]	1.02×10^6	3.84×10^5	1.22×10^{-4}	4.77×10^7	1.40×10^6	4.63×10^7	459.65	67.12	498.45	82.21

[a]http://files.grouplens.org/datasets/hetrec2011/hetrec2011-lastfm-readme.txt, http://www.lastfm.com and Ref. [23].
[b]http://grouplens.org/datasets/movielens/ and Ref. [24].
[c]http://www2.informatik.uni-freiburg.de/~cziegler/BX/ and Ref. [25].
[d]https://snap.stanford.edu/data/web-FineFoods.html and Ref. [26].
[e]http://labrosa.ee.columbia.edu/millionsong/tasteprofile and Ref. [27].

In the *second experiment* we generate surrogate networks where both edge and vertex strengths are allowed to vary. The edge weights are again constrained to an interval corresponding to the range of the edge weights in the original network, while the vertex strengths $W(v)$ are constrained to the interval $[0.9W(v), 1.1W(v)]$ for each vertex $v \in V$ in the original network. In the context of recommender systems this means that the total ratings given by a user and the total ratings received by an item cannot vary more than $\pm10\%$ from the value observed in the original data set.

In both experiments we consider the scalability of the CycleSampler method presented in this paper. Essentially, we want to determine typical running times when actually using the CycleSampler on real-world data sets. When using an MCMC sampler it is essential to ensure that the chain has converged before samples are used. However, studying the convergence of Markov chains is a nontrivial problem, and we here consider convergence in terms of a simple graphical technique. More exactly, we produce a trace plot showing the difference between the edge weight vector of the observed network ($\mathbf{w}$) and the jth surrogate network from the sampler ($\mathbf{w}_j^*$) evolves over time. We measure this difference using the l^2 norm

$$\|\mathbf{w} - \mathbf{w}_j^*\|_2 = \left\{\sum_{i=1}^{n} [\mathbf{w}(i) - \mathbf{w}_j^*(i)]^2\right\}^{1/2}. \tag{19}$$

We use this difference as a heuristic and conclude that the chain has converged when the difference no longer increases, i.e., when the curve levels out. It should be noted that this does not prove that the chain has converged, it indicates only if the chain has not converged.

Results

In the experiments we set a target of obtaining 100 000 surrogate networks. We used a cutoff time of 48 h for the experiments. The convergence experiments were run on a high-performance computing cluster [22] (one core from an Intel Xeon E5-2680 2.4 GHz with 30 Gb RAM) using R version 3.5.3. The full number of surrogate networks were obtained for Last.fm, MovieLens 100k, BookCrossing, FineFoods, and MovieLens 1M. For MovieLens 20M we obtained 15 500 surrogates in the first experiment and 16 500 surrogates in the second experiment. For TasteProfile we obtained 4750 surrogates in the first experiment and 4650 surrogates in the second experiment.

The initialization and sampling times (both in seconds) of the sampler are presented in Table III, recorded on a standard laptop equipped with a dual-core 2.6 GHz Intel Core i7-6600U processor and 20 Gb of RAM, running a 64-bit version of R (v. 3.5.2) on Linux. The initialization time (t_{init}) is the time required to set up the sampler, which consists of determining the spanning tree and identifying the cycles.

We here refer to modifying the weights corresponding to one of the cycles in a network as a *step*. We also denote taking a number of steps corresponding to the dimensionality of the null space of a network, $|C|$, as a *cycle step*. A cycle step hence consists of $|E| - |V| + 1$ steps in experiment 1 and $|E| + 1$ steps in experiment 2 (all of our networks are bipartite).

The sampling time (t_{sample}) for a particular network is the time required to take a cycle step. It should be noted that the sampling time does not include the time needed to make the sampler converge (which, as shown below, is on the order of 1000 cycle steps). The reported sampling time is the average of 10 samples. The subscripts *1* and *2* are used to denote the initialization times for experiment 1 and 2, respectively. The initialization and sampling times increase as the dimensionality of the null space of the network increases. This is also reflected in the initialization and sampling times for experiment 2, where the dimensionality of the null space is higher due to the addition of one self-loop per vertex required to preserve vertex strengths on an interval (see Sec. II B).

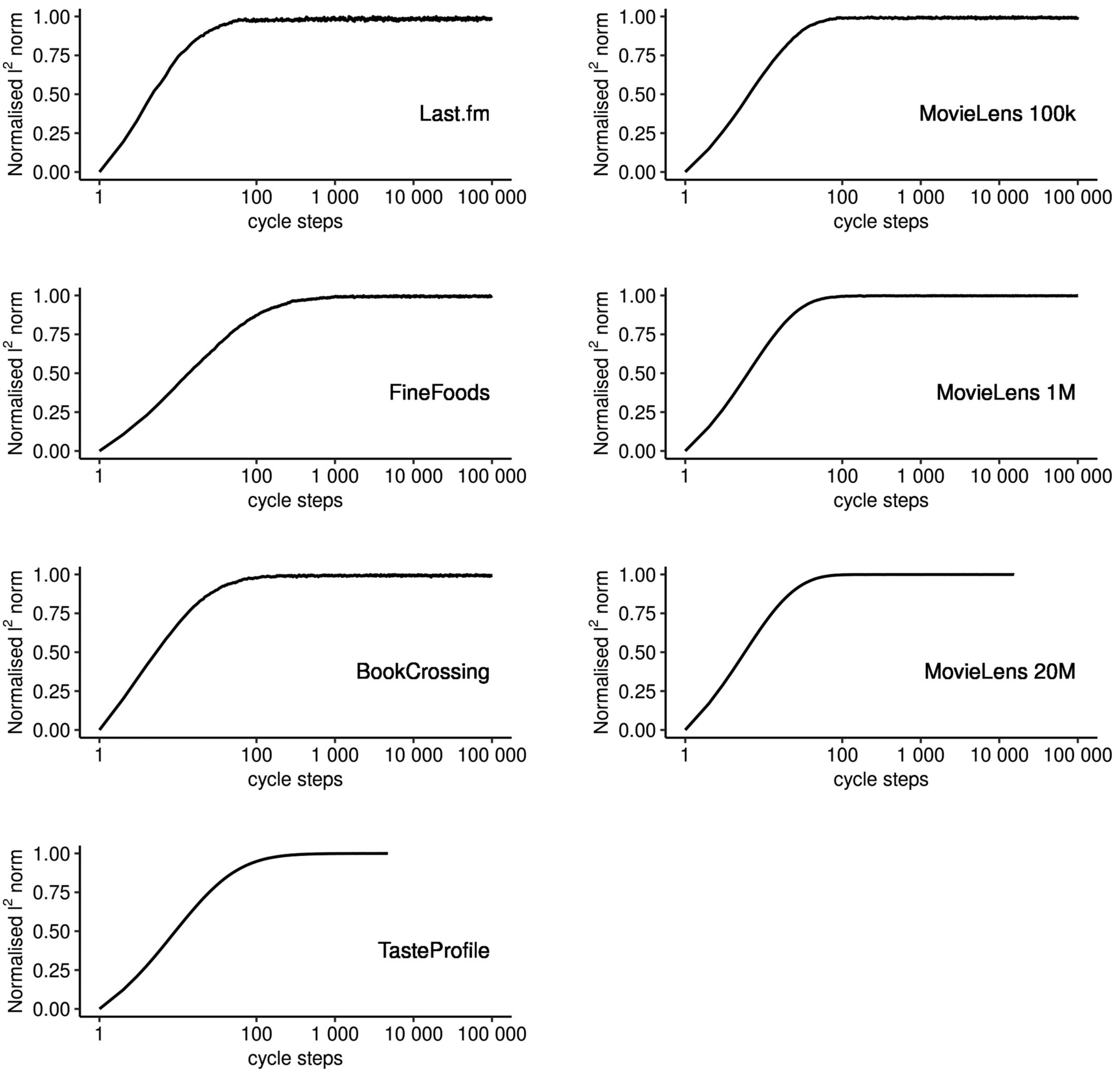


FIG. 8. Evolution of the l^2 norm between the starting state and the current state of the sampler for experiment 1, where vertex strengths are preserved exactly. For visualization purposes, the data points have been downsampled, taking points at logarithmically spaced intervals. Also, the l^2-norm is normalized to the interval [0, 1] so that 0 corresponds to the starting state and 1 to the maximum value of the norm. The x axis shows the number of cycle steps (a number of steps equal to the dimensionality of the null space of a given network) to facilitate comparisons between the different networks.

The initialization time is about a second for small networks (Last.fm, MovieLens 100k) and less than 10 min even for the TasteProfile network with tens of millions of edges. Similarly, the time needed to produce a surrogate network ranges from a fraction of a second for the small networks to about 1.5 min for the largest network.

The results from experiment 1, where the vertex strengths are preserved exactly, are shown in Fig. 8. We notice that on the order of 1000 cycle steps are needed for the sampler to converge for all data sets (i.e., for each data set we must perform 1000 $|C|$ modifications of weights using the cycles in the network).

The results from experiment 2, where the vertex strengths are preserved on an interval, are shown in Fig. 9. The sampler clearly converges more slowly for all data sets than in experiment 1; on the order of 10 000 cycle steps appears to be required for convergence, which is approximately a tenfold increase in number of cycle steps compared to experiment 1. Here the sampler has not yet converged MovieLens 20M and TasteProfile.

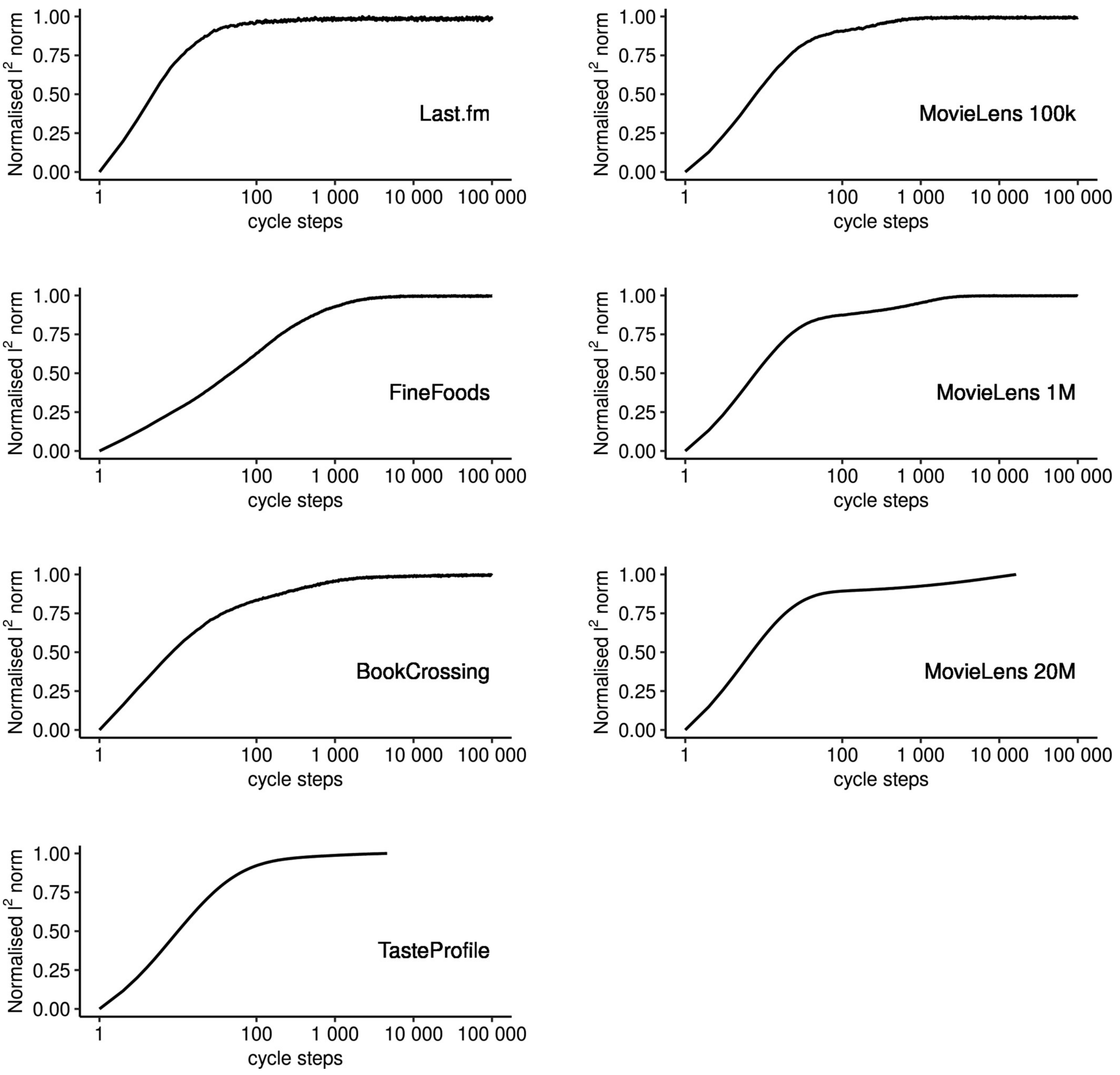


FIG. 9. Evolution of the l^2 norm between the starting state and the current state of the sampler for experiment 2, where vertex strengths are allowed to vary on an interval. Visualization as in Fig. 8.

B. Analysis of international trade networks

To demonstrate the applicability of the proposed CycleSampler method in actual analysis scenarios we here apply the CycleSampler to generate surrogate networks for investigating clustering coefficients in networks describing the trade between different countries.

We here aim to replicate the experiment in Ref. [2], who compared the clustering coefficients of the original data with surrogates where (1) edge weights were preserved exactly and (2) where strengths were preserved. Given a network, weight-preserving surrogates can easily be obtained by randomly shuffling the weights in the network. Strength-preserving surrogates, on the other hand, require more advanced methods such as the CycleSampler method presented in this paper. The authors of Ref. [2] consider the generation of strength-preserving surrogates in the case of *undirected* and *complete* networks; i.e., the direction of trade between countries is not taken into account, and it is assumed that any country can trade with any country. The assumption of complete networks means that the structure of the generated surrogates does not agree with the original networks, i.e., edges may be introduced or removed. In contrast, the surrogates generated here keep the network structure intact.

1. Data

The *International Trade Network* data set we consider here is the *Expanded Trade and GDP* data set [28] obtained from http://ksgleditsch.com/exptradegdp.html. The data set describes the trade relations between different countries from 1948–2000 (one network per year). The dimensions of the trade networks vary between 1948 and 2000 in terms of the number of edges and vertices. For each pair of countries i and j, the data set describes the *export* $E_{i,j}$ and *import* $I_{j,i}$ from i to j (and vice versa). The network is directed, i.e., the amount and direction of trade between two countries is not symmetric. Also, it should be noted that the export from i to j does not exactly equal the import to j from i due to differences in reporting [29].

We form two types of networks from the data set as follows. We form *directed trade networks* $\mathbb{W}^d$ as

$$\mathbb{W}^d_{a,b} = \tfrac{1}{2}(E_{a,b} + I_{b,a}), \tag{20}$$

where $\mathbb{W}^d_{a,b}$ denotes the average of the export from a to b and the import to b from a.

Also, following Ref. [2], we form an *undirected trade network* $\mathbb{W}^u$ as

$$\mathbb{W}^u_{a,b} = \tfrac{1}{2}(E_{a,b} + E_{b,a} + I_{a,b} + I_{b,a}), \tag{21}$$

where $\mathbb{W}^u_{a,b}$ denotes the average of all trade relations between a and b and consequently $\mathbb{W}^u_{a,b} = \mathbb{W}^u_{b,a}$, i.e., the network is undirected and symmetric.

To be able to compare our results to Ref. [2] we follow their procedure and omit zero-weight edges from all networks, and we also normalize the networks by dividing the edge weights in a network by the average edge weight of the network.

2. Clustering coefficient

Let $\mathbb{W} = (V, E)$ be either a directed or undirected trade network as defined above having m vertices. Let $\mathbb{W}_{a,b}$ denote the weight of the edge between vertices (countries) i and j. Also denote by $\max(\mathbb{W})$ the maximum edge weight in $\mathbb{W}$.

We calculate both the *weighted* clustering coefficient C, and the *unweighted* clustering coefficient K of vertex i [2,30] for both *directed* and *undirected* networks.

The weighted clustering coefficient is defined as

$$C_i = \frac{\sum_{j,k} (\mathbb{W}_{i,j}\mathbb{W}_{j,k}\mathbb{W}_{k,i})^{1/3}}{(m-1)(m-2)\max(\mathbb{W})}, \tag{22}$$

and the unweighted clustering coefficient is defined as

$$K_i = C_i \max(\mathbb{W}). \tag{23}$$

For a given network $\mathbb{W}$ we determine the average clustering coefficients $\bar{C}$ and $\bar{K}$ over all m vertices in the network.

3. Generating surrogates

We use the following experimental procedure to investigate the clustering coefficients in the networks. For both undirected and directed networks we generate three types of surrogates:

(1) Surrogates that do not preserve vertex strengths, but which preserve the distribution of edge weights exactly, obtained by randomly shuffling the edge weights in the network.

(2) Surrogates that preserve the vertex strengths exactly but allow the edge weights to vary on a given interval. The edge weights were allowed to vary from $\min w(e)$ to $\max w(e)$, $e \in E$.

(3) Surrogates that preserve the vertex strengths on an interval and allow the edge weights to vary on a given interval. The edge weights were allowed to vary from $\min w(e)$ to $\max w(e)$ for each edge $e \in E$. The vertex strengths $W(v)$ were allowed to vary from $0.75W(v)$ to $1.25W(v)$ for each vertex $v \in V$, i.e., corresponding to a $\pm 25\%$ change in vertex strength.

Following Ref. [2] we refer to the surrogates in (1) as *weight-preserving surrogates* and to the surrogates in (2) and (3) as *strength-preserving surrogates*.

The effect of the strength-preserving surrogates for a country i in a network is to redistribute the trade to and from i to the other countries. For undirected networks the total trade volume remains fixed (or restricted to an interval), but it is not possible to distinguish the trade direction (import or export). For directed networks the directed trade volume, i.e., volume of imports and exports, for a country i remains fixed or restricted to an interval.

We generate 1000 weight-preserving networks by randomly permuting the edge weights. Strength-preserving surrogate networks are sampled using Besag and Clifford's *serial method* [31], which yields exact p values when used in hypothesis testing. In this method the Markov chain is run both forwards and backwards from the observed state, and samples are taken at fixed intervals. Here we acquire 1000 surrogates taking the samples at intervals of 500 steps with one step corresponding to the number of cycles in the network (i.e., a cycle step).

4. Results

The experiments for the International Trade Networks were run on the above described high-performance computing cluster using R version 3.5.3. The experimental results are presented in Fig. 10. The top row shows the average weighted clustering coefficient $\bar{C}$ for directed (top left) and undirected (top right) networks. The bottom row shows the average unweighted clustering coefficient $\bar{K}$ for directed (bottom left) and undirected (bottom right) networks.

The top-right figure showing $\bar{C}$ for undirected networks, and the bottom-right figure showing $\bar{K}$ for undirected networks can be compared to the similar plots in Fig. 5 in Ref. [2] showing $\bar{C}$ and $\bar{K}$ for undirected networks and strength-preserving surrogates from 1948 until 2000.

The confidence bands for $\bar{C}$ and $\bar{K}$ that we calculate for both types of surrogates generated using CycleSampler are generally above the $\bar{C}$ and $\bar{K}$ from the original data. We also notice that $\bar{C}$ and $\bar{K}$ from surrogates preserving vertex strengths on an interval (shown with dotted lines in Fig. 10) are generally higher than $\bar{C}$ and $\bar{K}$ from surrogates preserving vertex strengths exactly (shown with dashed lines in Fig. 10). Also, the confidence bands closely match the time evolution of $\bar{C}$ and $\bar{K}$ from the original data. This applies to both surrogates preserving the vertex strengths exactly and to surrogates preserving the vertex strengths on an interval.

These results can be contrasted with those in Ref. [2] (in particular Fig. 5), who observe that the confidence band for the average weighted clustering coefficient $\bar{C}$ is both above (in the 1950s and 1960s) and below (roughly from 1970 onward) the clustering coefficient calculated from the original data. From this one might conclude that in recent decades trade flows between countries have become more "uniform"

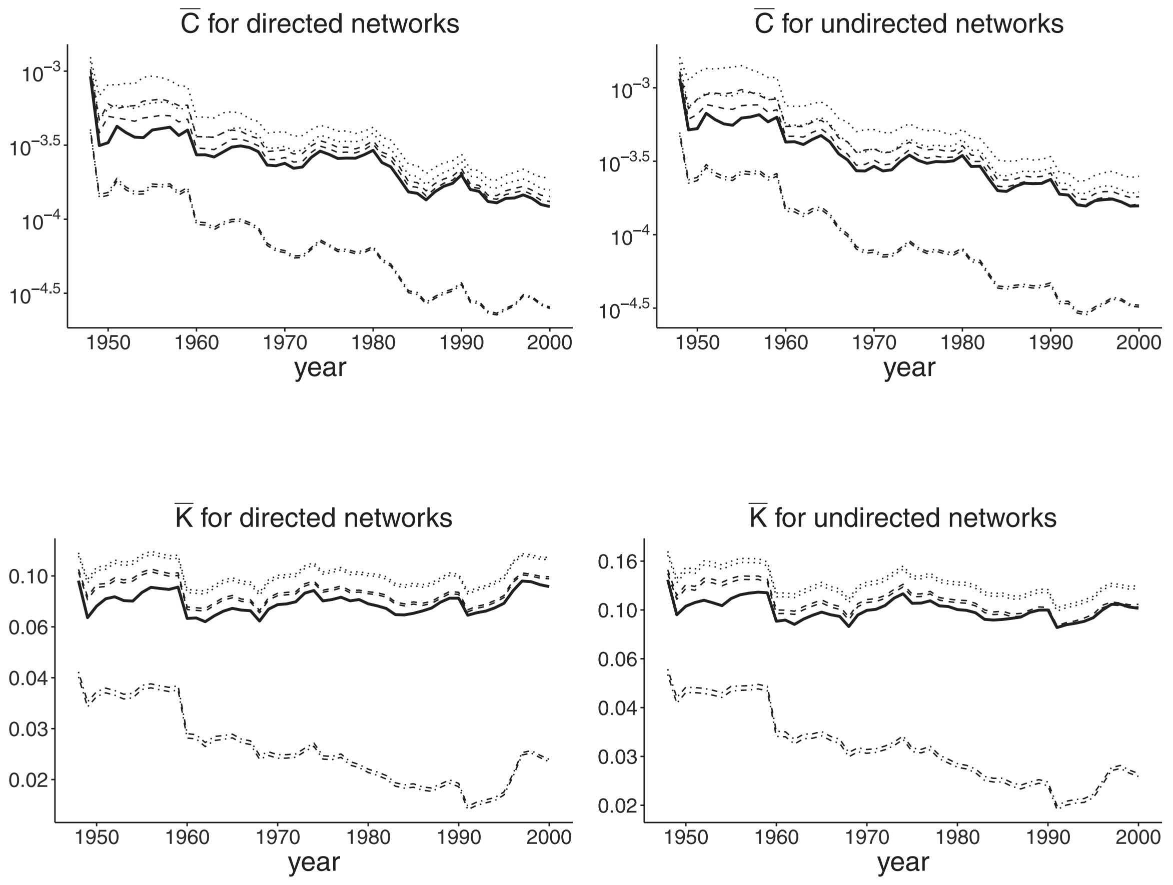


FIG. 10. The average weighted ($\bar{C}$) and unweighted ($\bar{K}$) clustering coefficients for the International Trade Networks between the years 1948–2000. The clustering coefficient for the original data is shown in solid black. For each type of surrogate, the upper and lower one standard deviation confidence bands for the mean are plotted. The confidence intervals for weight-preserving surrogates is shown using dot-dash lines, for surrogates preserving vertex strengths exactly using dashed lines, and for surrogates preserving vertex strengths on an interval using dotted lines.

than observed in random surrogate networks. (The weighted clustering coefficient is small when most of the trade flow takes place between only a few countries.) However, if the structure of the ITNs is taken into account when generating the surrogate networks, our observations suggest that no such change has happened during 1948–2000; we consistently observe that the average weighted clustering coefficient $\bar{C}$ is lower in real networks.

Finally we note that, similar to Fig. 5 in Ref. [2], $\bar{C}$ and $\bar{K}$ from the weight-preserving surrogates (dot-dash lines) are consistently much lower than $\bar{C}$ and $\bar{K}$ calculated from the original data.

IV. CONCLUSIONS

Generating surrogate networks has important applications in multiple domains where it is required to investigate and understand the significance of different phenomena described by the structure of the network. Many such networks are often sparse and large, and consequently generating surrogate networks adhering to specific constraints is a difficult problem. In this paper we presented CycleSampler; a Markov chain Monte Carlo method for structure-preserving sampling of both directed and undirected networks with interval constraints on both edge and vertex strengths.

The presented method provides an efficient means for generating surrogates for large networks, and we provided an empirical evaluation demonstrating that the method scales to large sparse real-life networks. We believe that the CycleSampler method has applications in many domains. We have released an open-source implementation of the method as an R package [20].

ACKNOWLEDGMENTS

This work was funded by the Academy of Finland (decisions 326280 and 326339) and Tekes (Revolution of Knowledge Work). We acknowledge the computational resources provided by the Finnish Grid and Cloud Infrastructure [22].

APPENDIX A: A NOTE ON MAXIMUM-ENTROPY MODELS

We here describe the maximum-entropy model used in the examples in the introduction. The problem formulation in the maximum-entropy model when modeling edge weights is as follows:

Problem 3 (Maximum-entropy model). Find a probability density p over the edge weights w^* from the model such that the entropy

$$S = E_{p(w^*)}[-\log p(w^*)]$$

is maximized, subject to the constraint that the expected vertex strengths match the observed vertex strengths:

$$\text{for all } v \text{ it holds that } E_{p(w^*)}[W^*(v)] = W(v) = \sum_{e\in n(v)} w(e).$$

The solution to Problem 3 is given by the following lemma; for a proof refer to the proof of Theorem 12.1.1 in Ref. [32].

Lemma 4. The probability density p that maximises the entropy in Problem 3 is of the form

$$p(w^*) \propto \exp\left[\sum_v \lambda_v W^*(v)\right] \propto \exp\left[\sum_{e\in n(v)} \lambda_e w^*(e)\right].$$

Hence, in order to solve Problem 3 we must find the edge-specific Lagrange multipliers λ_e, which is a convex optimization problem that can be solved numerically.

APPENDIX B: PREPROCESSING OF DATA SETS

We performed the following preprocessing steps for the networks: (1) duplicated edges were removed; (2) for Last.fm and TasteProfile all ratings above 2500 and 20, respectively, were discarded; (3) for BookCrossing we used only explicit ratings (i.e., nonzero ratings), rows with book id:s containing the symbol "?" were discarded, and the symbols "\," "=" as well as blank spaces were removed from the book id:s.

[1] S. Fortunato, Phys. Rep. **486**, 75 (2010).
[2] G. Ansmann and K. Lehnertz, Phys. Rev. E **84**, 026103 (2011).
[3] D. Hao, C. Ren, and C. Li, BMC Syst. Biol. **6**, 34 (2012).
[4] K. E. Joyce, P. J. Laurienti, J. H. Burdette, and S. Hayasaka, PLoS ONE **5**, e12200 (2010).
[5] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, and S. Vigna, in *Proceedings of the 4th Annual ACM Web Science Conference (WebSci '12)* (ACM Press, New York, 2012), pp. 33–42.
[6] W. Chen, L. V. S. Lakshmanan, and C. Castillo, *Information and Influence Propagation in Social Networks*, Synthesis Lectures on Data Management (Morgan & Claypool, San Rafael, CA, 2013).
[7] A. Coolen, A. De Martino, and A. Annibale, J. Stat. Phys. **136**, 1035 (2009).
[8] S. Hanhijärvi, G. C. Garriga, and K. Puolamäki, in *Proceedings of the 2009 SIAM International Conference on Data Mining (SDM09)*, edited by C. Apte, H. Park, K. Wang, and M. J. Zaki (SIAM, Philadelphia, PA, 2009), pp. 780–791.
[9] E. S. Roberts and A. C. C. Coolen, Phys. Rev. E **85**, 046103 (2012).
[10] A. Gionis, H. Mannila, T. Mielikäinen, and P. Tsaparas, ACM Trans. Knowl. Disc. Data (TKDD) **1**, 14 (2007).
[11] T. Squartini and D. Garlaschelli, New J. Phys. **13**, 083001 (2011).
[12] E. Jaynes, Phys. Rev. **106**, 620 (1957).
[13] E. Jaynes, Phys. Rev. **108**, 171 (1957).
[14] E. Jaynes, Proc. IEEE **70**, 939 (1982).
[15] J. R. Gilbert and M. T. Heath, SIAM J. Alg. Discr. Meth. **8**, 446 (1987).
[16] T. F. Coleman and A. Pothen, SIAM J. Alg. Discr. Meth. **8**, 544 (1987).
[17] T. F. Coleman and A. Pothen, SIAM J. Alg. Discr. Meth. **7**, 527 (1986).
[18] S. Akbari, N. Ghareghani, G. B. Khosrovshahi, and H. Maimani, Linear Alg. Appl. **414**, 617 (2006).
[19] R Core Team, *R: A Language and Environment for Statistical Computing* (R Foundation for Statistical Computing, Vienna, 2018).
[20] K. Puolamäki and A. Henelius, Cyclesampler, https://github.com/edahelsinki/cyclesampler (2018).
[21] K. Van den Meersche, K. Soetaert, and D. Van Oevelen, J. Stat. Softw., Code Snippets **30**, 1 (2009).
[22] Finnish grid and cloud infrastructure, urn:nbn:fi:research-infras-2016072533.
[23] I. Cantador, P. Brusilovsky, and T. Kuflik, in *Proceedings of the 5th ACM Conference on Recommender Systems*, RecSys 2011 (ACM Press, New York, 2011).
[24] F. M. Harper and J. A. Konstan, ACM Trans. Interactive Intel. Syst. (TiiS) **5**, 19 (2016).
[25] C. N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen, in *Proceedings of the 14th International Conference on World Wide Web (WWW '05)* (ACM Press, New York, 2005), pp. 22–32.
[26] J. J. McAuley and J. Leskovec, in *Proceedings of the 22nd International Conference on World Wide Web (WWW '13)* (ACM Press, New York, 2013), pp. 897–908.
[27] T. Bertin-Mahieux, D. P. W. Ellis, B. Whitman, and P. Lamere, in *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, edited by A. Klapuri and C. Leider (University of Miami, Coral Gables, FL, 2011).
[28] K. S. Gleditsch, J. Conflict Resolut. **46**, 712 (2002).
[29] K. Bhattacharya, G. Mukherjee, J. Saramäki, K. Kaski, and S. S. Manna, J. Stat. Mech. (2008) P02002.
[30] J.-P. Onnela, J. Saramäki, J. Kertész, and K. Kaski, Phys. Rev. E **71**, 065103(R) (2005).
[31] J. Besag and P. Clifford, Biometrika **76**, 633 (1989).
[32] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. (John Wiley & Sons, Hoboken, NJ, 2006).