2K+ Graph Construction Framework: Targeting Joint Degree Matrix and Beyond

Bálint Tillman, Athina Markopoulou, *Senior Member, IEEE,* Minas Gjoka, and Carter T. Butts

***Abstract*—In this paper, we study the problem of generating synthetic graphs that resemble real-world graphs in terms of their degree correlations and potentially additional properties. We present an algorithmic framework that generates simple undirected graphs with the exact target joint degree matrix JDM, which we refer to as 2K graphs, in linear time in the number of edges. Our framework imposes minimal constraints on the graph structure, which allows us to target additional graph properties during construction, namely node attributes (2K+A), clustering (both average clustering, 2.25K, and degree-dependent clustering, 2.5K) and number of connected components (2K+CC). We also define, for the first time, the problem of directed 2K graph construction (D2K), we provide necessary and sufficient conditions for realizability, and we develop efficient construction algorithms. We evaluate our approach by creating synthetic graphs that target real-world graphs both undirected (such as Facebook) and directed (such as Twitter) and we show that it brings significant benefits, in terms of accuracy and running time, compared to state-of-the-art approaches.**

***Index Terms*—network topology, complex networks, random graphs, degree correlations, generative network models.**

I. Introduction

IT is often desirable to generate synthetic graphs that resemble real-world networks with regards to certain properties of interest. For example, researchers often want to simulate a process on a realistic network topology but they may not have access to a real-world network; or they may want to generate several different realizations of graphs of interest. In this paper, we target both directed and undirected graphs including, but not limited to, online social networks.

There is a large body of work, in classic literature [1], [2], [3], [4], as well as more recently [5], [6], [7], on generating realizations of undirected graphs that exhibit (exactly) target structural properties such as a degree sequence or a joint degree matrix. In this paper, we adopt the dK-series framework [5], [6], which describes graphs in terms of a series of frequencies of induced subgraphs of increasing size, thus providing an elegant way to trade off accuracy (in terms of graph properties) vs. complexity (of the algorithms generating graph realizations with those properties). Construction of 1K-graphs (*i.e.,* graphs with a target degree sequence) is well understood: efficient algorithms and realizability conditions are known since Havel-Hakimi [2], [3]. Construction of 2K-graphs (graphs with a target joint degree matrix) has been studied in parallel by several researchers, namely Amanatidis et al. [8], Czabarka et al. [9], and our group [10]. For $d > 2$, which is necessary for capturing the clustering exhibited in social networks, we recently proved that the problem is NP-hard [11] and we developed efficient heuristics [10]. In contrast, construction of directed graphs was not as well developed: results were known for construction of graphs with a target directed degree sequence [12], [13], but the notion of directed degree correlation, or directed dK-series for $d \geq 2$, has not been previously defined.

In this paper, we present a general algorithmic framework that allows us to construct synthetic graphs with an exact target JDM (which we refer to as "2K" graphs) and potentially additional properties (which we refer to as "2K+" graphs). The core of our 2K+ framework is an algorithm that can provably create synthetic undirected graphs with the exact target JDM, and it does so efficiently (*i.e.,* in linear time in the number of edges), while imposing minimal constraints on the graph structure (*i.e.,* it can potentially create any 2K realization). We exploit the latter feature to impose additional properties during construction, namely node attributes (2K+A), clustering (both average clustering, 2.25K, and degree-dependent clustering, 2.5K) and number of connected components (2K+CC). We also extend the 2K framework, for the first time, to directed graphs. We define two notions of degree correlation in directed 2K graphs: directed 2K (D2K) and its special case D2.1K. D2K includes the notion of directed degree sequence (DDS) and maps directed graphs to bipartite undirected graphs to also express degree-correlation via a joint degree-attribute matrix (JDAM) for the bipartite graph. This problem definition lends itself naturally to techniques we previously developed for undirected 2K [10], which we exploit to develop (i) necessary and sufficient realizability conditions and (ii) an efficient algorithm that constructs graph realizations with the exact target D2K.

The outline of the paper is as follows. Section II summarizes related work and defines the problem of 2K+ graph construction; Section II-D provides an overview of our contributions. Section III defines and solves the basic 2K-construction problem, extends it to the 2K+ framework for undirected graphs, and provides a comparison to state-of-the-art methods when targeting several real-world undirected graphs. Section IV defines the Directed 2K problem (D2K and its special case D2.1K); it provides realizability conditions for D2K, an efficient algorithm for constructing such realizations, and evaluation targeting real-world directed graphs. Section V

B. Tillman, A. Markopoulou, and C. T. Butts are with the University of California, Irvine, CA, USA. (E-mail: tillmanb@uci.edu, athina@uci.edu, buttsc@uci.edu).

M. Gjoka was with the University of California, Irvine, when the work was conducted. He is currently with Google, Santa Monica, CA, USA. (E-mail: mgjoka@uci.edu.)

This work was supported by NSF Award 1526736 and a Networked Systems Fellowship at UCI. A. Markopoulou is a member of the Center for Pervasive Communications and Computing (CPCC) at UCI.

concludes the paper. Additional results (proofs and simulation results) are deferred to the Appendices, available as Supplemental Materials.

II. Our Work in Perspective

A. Problem Statement

When constructing a synthetic graph that resembles a real graph G, we have to specify several aspects of the problem.

First, we have to choose the properties of G that should be preserved: this is in itself a challenging research question. We adopt the systematic framework of *dK-series* from Mahadevan *et al.* [5], which was introduced to characterize the properties of a graph using a series of probability distributions specifying all degree correlations within d-sized, simple, and connected subgraphs of a given graph G. In this framework, higher values of d capture progressively more properties of G at the cost of more complex representation of the probability distribution. The dK-series exhibit two desired properties: inclusion (a dK distribution includes all properties defined by any d'K distribution, $\forall d' < d$ and convergence (nK, where $n = |V|$ specifies the entire graph, within isomorphism). [1]

Second, we have to define in what sense the synthetic graph should resemble the original one. In this paper, we produce *simple* graphs that exhibit the target properties *exactly*. This is different from the stochastic approach presented by [14] (target properties are achieved in expectation) or the configuration model in [15] (graphs could be multigraphs as well).

Depending on how probabilistic construction is performed, its realizations may be associated with JDMs that are far from the target, which may or may not be desirable in practice. While our focus in this paper is on exact construction, we note that probabilistic and deterministic construction are complementary approaches in a broader graph construction toolkit and can be used together. Exact construction can facilitate probabilistic construction by, for example, first simulating JDMs from a target distribution and then construction graphs satisfying those simulated JDMs (if necessary, filtering out unfeasible JDMs as a form of rejection sampling). Many approaches to probabilistic simulation of complex network distributions (*e.g.,* those based on Markov chain Monte Carlo or related methods) are fairly expensive, and exact construction may be faster (provably on the order of the number of edges) for large graphs.

Third, we have to specify what realizations with the target properties can be achieved and how: at least one (if such exists), all possible realizations (and the corresponding sampling method), a subset of realizations, etc.

A **dK-construction problem** takes as input the target properties[2] (*i.e.,* the dK-series and potentially additional properties), and addresses the three following subproblems.

- **Realizability:** Provide necessary and sufficient conditions such that there exist simple graphs with these target properties.
- **Construction:** Design an algorithm that generates at least one such graph realization.
- **Space of realizations:** Characterize the space of all graph realizations with these target properties and provide ways to sample from them.

In the next subsections, we discuss in detail the dK-series framework [5] and summarize prior work.

B. Prior Work on Undirected Graph Construction

Consider an undirected graph $G = (V, E)$, with $n = |V|$ nodes and $m = |E|$ edges. Let $\deg(v)$ be the degree of node v. Let V_k be the set of nodes that have degree k, also referred to as degree group k.

0K Construction. 0K describes graphs with a prescribed number of nodes and edges. This notion corresponds to simple Erdős-Rényi (ER) graphs with fixed number of edges.

1K Construction. In an undirected graph G, a node v has degree $deg(v)$, $D_k = |V_k|$ is the number of nodes of degree $k, k = 1, ..., d_{max}$, where d_{max} is the maximum degree in the graph. The degree sequence is simply:

$$DS = \{deg(v_1), deg(v_2), ..., deg(v_{|V|})\} \tag{1}$$

In the dK-series terminology, the degree sequence specifies 1K. Degree sequences have been studied since the 1950s, thus we only focus on the most relevant results. The realizability conditions for degree sequences were given by the Erdős-Gallai theorem [1], and first algorithm to produce a single realization by Havel-Hakimi [2], [3]. The space of simple graph realizations of 1K distributions is connected over double edge swaps preserving degrees [4]. More recently, importance sampling algorithms were proposed in Blitzstein *et al.* [16] and Genio *et al.* [17].

2K Construction. A Joint Degree Matrix (JDM) is given by the number of edges between nodes of degree k and l [3]:

$$JDM(k, l) = \sum_{v \in V_k} \sum_{w \in V_l} 1_{\{(v,w) \in E\}} \tag{2}$$

Degree assortativity is a scalar that is often used to summarize JDM. A given 2K (JDM) also fixes 1K (the degree vector D_k):

$$D_k = |V_k| = \frac{1}{k} \sum_{l=1}^{d_{max}} \text{JDM}(k, l) \tag{3}$$

as well as the number of edges $m = |E|$, and the number of nodes $n = |V|$ in the graph, thus 0K.

Realizability conditions for undirected 2K were provided by Amanatidis *et al.* [8] [18]. Algorithms for generating realizations of a target JDM were provided in Czabarka *et al.* [9], Gjoka *et al.* [10] and Stanton and Pinar [19]. The algorithms presented in [18] and [19] are designed to produce restricted realizations that exhibit the Balanced Degree Invariant (BDI) property, which evenly spreads edges between degree groups.

[1] A study of how well dK-series match real-world graphs was conducted by Orsini *et al.* in [6]. Six real-world undirected graphs were considered and compared to synthetic graphs produced by dK-series in terms of a range of graph properties (from local to global, targeted and non-targeted). The paper [6] demonstrated the convergence of dK-series for these graphs and properties, for $d \leq 2.5$, in the overwhelming majority of the cases.

[2] We use $\odot$ to denote the target properties, that the constructed graph should have; absence of $\odot$ denotes the actual values for the constructed, or partially constructed, graph.

[3] In case of $k = l$, this notation returns twice the number of edges within degree group k, resulting in minor differences in notation from related work.

In [9] and [10], the algorithms have non-zero probability to produce any realization of a 2K distribution. Bassler *et al.* [20] introduced an importance sampling algorithm. We provide an overview of the relation of these algorithms in Section III-A3.

W.r.t. sampling, the space of 2K-graphs with a realizable JDM was shown to be closed under JDM-preserving double edge swaps [9], [18], [4] and can be used to generate approximate probability samples for 2K. However, fast mixing has not been proved for 1K or 2K, apart from special classes of realizations [21] [22]. We further discuss these types of swaps in Section III in relation to graph properties like number of connected components and clustering.

dK, $d > 2$ Construction. While algorithms of known time complexity exist for $d \leq 2$, Monte Carlo Markov Chain (MCMC) approaches are typically used for $d > 2$. Several attempts were made to find polynomial time algorithms to produce 3K graphs [5] or 2K realizations with prescribed (degree-dependent) clustering coefficient [23], [10] and [6], but a member of our team and collaborators recently proved that checking realizability of these inputs is NP-Complete [11].

Annotated graph construction was proposed by Dimitropoulos et al. in [24] that considered degree correlations, however the proposed construction method will generate graphs with self-loops or multi-edges initially. An additional step removes these extra edges to make the graph simple and finally the largest connected component of the graph is returned.

Partition Adjacency Matrix problem (PAM) is another relaxation of JDM construction by Erdős *et al.* [7], where the number of edges are defined over a partition of nodes (not necessarily forming degree groups) and each part of the partition is associated with a degree sequence. In [7], a construction algorithm was shown to solve the problem for any bipartition and other special cases. However, there is no general solution available for PAM and it is believed that even the realizability question is NP-Complete.

C. Prior Work on Directed Graph Construction

We extend the taxonomy of dK-series, from undirected graphs to also describe properties of directed graphs.

Directed 0K. There is a simple extension of ER graphs to generate directed graphs as well, which we use in our evaluation. In addition, we consider the UMAN model [25], which captures the number of mutual, asymmetric, and null dyads in a graph: UMAN can be thought of as 0K with fixed numbers of mutual and unreciprocated edges.

Directed 1K. In a directed graph, a node v has both in and out degree (d_v^{in}, d_v^{out}) and the *directed degree sequence* can be expressed as follows.

$$DDS = \{(d_v^{in}, d_v^{out}), v \in V\} \tag{4}$$

It is well known from Gale's work [12], that any directed graph can be mapped 1-1 to an undirected bipartite graph,

[4]In [18] an induction proof is provided that shows swaps to reduce the symmetric difference between any two realizations until the difference disappears, a simple example is shown in Fig. 4. In [9] a different approach is shown where a swap sequence is given to reach a BDI realization and later to reduce the problem to a union of unipartite (1K) and bipartite degree sequence problems which have known solutions.

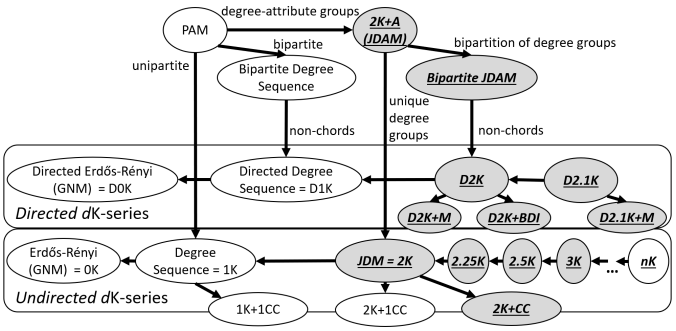


Fig. 1. Overview of problems for both directed and undirected graph construction. Our contributions are the modules highlighted in gray.

where each node v of the directed graph is split in two nodes v_{in} and v_{out}, and the undirected edges across the two (in and out) partitions of the bipartite graph correspond to the directed edges in the directed graph. A self loop (v, v) in the directed graph corresponds to a "non-chord" (v_{in}, v_{out}) in the bipartite graph. We further discuss this transformation and provide examples in Section IV.

Construction algorithms are known for a bipartite degree sequence with [12], or without non-chords, and therefore for the corresponding directed graphs with or without [13] self-loops, respectively. More recently, an importance sampling algorithm was provided for D1K in [26].

D. Our Contributions: The 2K+ Framework

Our 2K+ framework not only constructs graphs with an exact target JDM but also targets additional properties. This paper unifies and extends a number of our previously disconnected results, which have partially appeared in conferences and workshops, on construction of undirected (2K [23] [27], 2.25K [10], 2.5K [23], 2K+CC [28]) as well as directed graphs (D2K, D2.1K [29]). Figure 1 depicts an overview of the problem space including past work and our contributions.

2K Construction. JDM realizations have been studied independently and in parallel by several groups. The problem was spearheaded by Amanatidis *et al.* first in an unpublished manuscript in 2008 (on the realizability conditions), and in a more recent arxiv in 2015 [18] (on construction algorithms for balanced and single connected component realizations). It was followed by Stanton and Pinar in 2012 [19] also targeting BDI realizations and a proof for connected space that was found to have flaws. Around the same time, construction algorithms that could also produce non-balanced realizations were developed by two other groups: Czabarka *et al.* [9] in 2015, and by Gjoka *et al.* in 2013 [23] and 2015 [10] (ours). The common idea behind all construction algorithms is to add one edge at the time so as to not exceed node degrees or JDM; algorithms vary on if/how they violate any of these two properties and on when/how to correct it using local rewiring; which is always possible for realizable JDMs. Details on the algorithm and comparison can be found in Section III-A and III-A3, respectively. In addition to independently developing one of the 2K algorithms in [10], [23], our contribution in this part lies in the particular order in which we add and rewire

edges, which is necessary for being able to target additional properties and enable a framework beyond just 2K.

A note on use cases: The 2K distribution captures properties such as differential mixing by degree, which can be important for modeling phenomena such as diffusion. In particular, in a degree-conditioned random graph, high-degree nodes are proportionally more likely to be adjacent to each other than to low-degree nodes; this produces a core in the network, and high connectivity among hubs (particularly where the degree distribution is highly skewed), leading to rapid hub-based diffusion. In real networks, however, one may see other mixing patterns involving, for example, higher or lower levels of assortative degree mixing, or entirely different patterns (*e.g.,* a tendency for degree-1 nodes to mix with each other, producing large numbers of isolated dyads). Matching the 2K distribution ensures that these properties are accurately represented. It is interesting to note that 2K can match these properties. while 1K cannot, while having the same linear complexity $O(|E|)$.

Clustering: 2.25K and 2.5K. 2K (JDM) in itself does not capture clustering, which is an essential property of several real-world graphs such as online social networks. 3K captures a very strong notion of clustering, whose construction we recently showed to be NP-hard [11]. Our main motivation behind the 2K+ construction work was to efficiently generate graphs with a target JDM *and* some notion of clustering and we targeted two such notions: average clustering $\bar{c}$ and average degree-dependent clustering $\bar{c}(k)$, defined as follows.

The clustering coefficient c_v of a node v is defined as the ratio of the number of triangles T_v attached to node v divided by the maximum possible number of such triangles. The average clustering coefficient ($\bar{c}$) averages c_v over all nodes V. The average degree-dependent clustering coefficient $\bar{c}(k)$ averages over degree groups. In summary:

$$c_v = \frac{T_v}{\binom{\deg(v)}{2}} \quad \bar{c} = \frac{1}{n}\sum_v c_v \quad \bar{c}(k) = \frac{1}{|V_k|}\sum_{v \in V_k} c_v. \qquad (5)$$

In the **2.25K** problem, we develop a construction approach to target JDM and $\bar{c}$ [10]. In the **2.5K** problem, we develop a hybrid construction and MCMC approach to target JDM and $\bar{c}(k)$ [10], [23], efficiently. The heuristic nature of our approaches is justified by the hardness of 2K construction with *any* notion of clustering [11].

A note on use cases: 2.25K and 2.5K distributions capture clustering, which is also important for diffusion: it is well known that clusters are the obstacle to information cascades over networks. Online social networks exhibit high clustering (e.g., compared to random graphs) and this is one of the main motivations for this work: producing synthetic graphs that resemble real, large online social network graphs in reasonable time. As we will see in the evaluation results, prior MCMC-based approaches targeting clustering on large online social networks do not converge in weeks, while our 2.25K and 2.5K construction converged on the order of minutes.

Number of Connected Components: 2K+CC. A single connected component (which we refer to as 1CC) can be targeted in addition to the degree sequence [4], [30], or in addition to a target JDM [18]. Our result builds on and extends

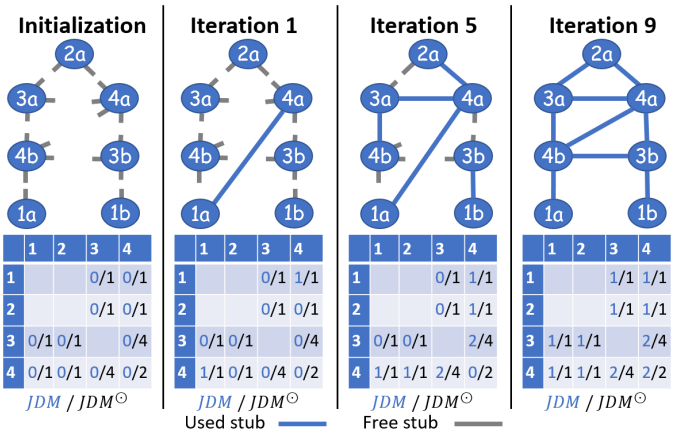


Fig. 2. Example of running `2K_Simple`. The algorithm starts from nodes with only free stubs (left). In each iteration it creates one edge by connecting 2 free stubs (between two nodes whose degree pair has not reached the target JDM yet) and it increases the corresponding entries in JDM by one. At the end (right) the graph is complete and the JDM reaches the target.

Amanatidis et al. [18] to target minimum number of connected components for a given JDM, and we show that the space of those realizations is connected under JDM-preserving double-edge swaps; a one-page abstract is at [28] and the extended version and proofs are provided in this paper.

Node Attributes: 2K+A. In order to capture attributes in addition to structural properties, we were the first to define and target the Joint Degree-Attribute Matrix (JDAM), which captures correlation between node degrees and attributes. We show that our 2K construction algorithms gracefully extends to JDAM construction [10]. This is useful not only for incorporating attributes into the model, but also for imposing additional structure by properly assigning the attributes in JDAM, such as bipartite JDAMs and community membership.

Directed Graphs: D2K, D2.1K. We were the first to define and target directed degree correlations in [29]. In our main approach, D2K, we represent directed graphs as bipartite graphs with non-chords and we target the bipartite JDAM to construct simple directed graphs. In our second approach, we further restrict the notion of directed degree correlation to D2.1K that captures in-, out-degree correlations for a partition of nodes into the same degree groups (nodes with the same in and out degree); we show that D2.1K can be solved targeting JDAM with a more granular partitioning of attributes. In addition to the above contributions, which first appeared in [29], in this paper we also (i) provide a heuristic to target the number of mutual edges in a directed graph and (ii) we show that D2K and D2.1K always have BDI realizations, similarly to the undirected case of 2K.

III. Undirected Graphs

A. 2K Construction: Target JDM

In this section, the input is the target JDM$^{\odot}$, and the goal is to create a (at least one) simple undirected graph with N nodes that exhibits that exact JDM$^{\odot}$, if it is realizable.

1) Realizability: Not every target JDM is realizable (or "graphical"): there does not always exist at least one simple

graph with this exact property. Necessary and sufficient conditions for a target JDM to be realizable, have been developed independently [9], [10], [18], [19] and are the following:

I $\forall k,\ JDM(k,k) \leq |V_k| \cdot (|V_k| - 1)$
II $\forall k, l, k \neq l,\ JDM(k,l) \leq |V_k| \cdot |V_l|$
III $\forall k : |V_k| = \sum_l \frac{JDM(k,l)}{k}$, and it is an integer.

These conditions are necessary and describe intuitive conditions for inputs to be realizable. Violating the first condition would necessarily result in a multi-graph or graphs with self-loops, since it describes the number of edges contained in a complete graph for a degree group. Similarly the second condition describes a complete bipartite graph between a pair of degree groups. The third condition ensures that size of degree groups are integers and gives the number of nodes with certain degree. Sufficiency of these conditions are shown by a constructive proof of our algorithm.

2) Algorithm: `2K_Simple` receives a target JDM$^{\odot}$ as input and creates a simple undirected graph with JDM$^{\odot}$. It is summarized next and is illustrated in the example of Fig. 2.

Algorithm 1: `2K_Simple`

Input: $JDM^{\odot}$
Initialize:
a: Create $|V|$ nodes; each $v \in V$ has $deg(v)$ free stubs
b: Set $JDM(k,l) = 0$ for every $(k,l) \in JDM^{\odot}$
Add Edges:
1: **for** $(k,l) \in JDM^{\odot}(k,l)$
2: **while** $JDM(k,l) < JDM^{\odot}(k,l)$
3: Pick any nodes $v \in V_k$ and $w \in V_l$ s.t. (v,w) is not an existing edge
4: **if** v does not have free stubs:
5: v': node in V_k with free stubs
6: NeighborSwitch(v,v')
7: **if** w does not have free stubs:
8: w': node in V_l with free stubs
9: NeighborSwitch(w,w')
10: add edge between (v, w)
11: $JDM(k,l)$++ ; $JDM(l,k)$++
Output: simple undirected graph with $JDM = JDM^{\odot}$

The initialization phase is depicted in the leftmost column of Fig. 2. We create $|V| = n$ nodes, labeled by their degree; note that $|V|$ and $D_k^{\odot}$ can be found from JDM$^{\odot}(k,l)$. Following the configuration model approach, we assign k free stubs to every node $v \in V_k$ according to their degree. Stubs are the "half" edges shown in the top-left part of Fig. 2, originally free, *i.e.,* not connected to any other nodes. We also initialize all entries of $JDM(k,l)$ to zero.

Then the algorithm proceeds in iterations by adding one edge at a time until JDM matches $JDM^{\odot}$. More specifically, we pick two nodes v and w from degree groups V_k and V_l respectively, s.t. $JDM(k,l)$ has not reached its target yet (*i.e.,* $JDM(k,l) < JDM^{\odot}(k,l)$ in line 2 of Algorithm 1) and (v,w) is not an edge. Then the algorithm connects two of their stubs to create an edge. Furthermore, the algorithm should be able to add an edge even if one or both nodes do not have free stubs. In that case, Lemma 3 guarantees that we will always be able to perform edge rewiring, which we refer to as *NeighborSwitch*, so as to free stubs for v and/or w.

More specifically, a NeighborSwitch for a given node w frees a stub for node w and preserves the current JDM without creating multi-edges or self-loops. It does so, if also given a node w' with the same degree as w and a free stub. First, we find a neighbor t of w such that t is not a neighbor of w', then removing edge (w,t) and adding edge (w',t). The following pseudocode summarizes a NeighborSwitch, which is also illustrated in Fig. 3.

NeighborSwitch(node w,w'):
1: find t: neighbor of w and not neighbor of w'
2: remove edge (w,t)
3: add edge (w',t)

Correctness. `2K_Simple` terminates and constructs a simple undirected graph with the exact $JDM^{\odot}$.

Proof. In each iteration, `2K_Simple` adds exactly one edge making sure that JDM values never exceed target $JDM^{\odot}$ value: *i.e.,* $JDM(k,l) \leq JDM^{\odot}(k,l)$. Starting from an empty graph, the algorithm adds $|E|$ edges and then terminates with $JDM = JDM^{\odot}$. Lemma 1 shows that the algorithm will not get stuck, *i.e.,* if we have not reached the target, it is possible to find $v \in V_k, w \in V_l$ nodes where an edge can be added. There are three cases depending on whether v, w have free stubs or not:

Case 1. Add a new edge between v, w nodes w/free stubs, no local rewiring needed.

Case 2. Add a new edge between a node v w/out free stubs and a node w w/free stubs. Lemma 2 shows that there is $v' \in V_k$ w/free stubs and Lemma 3 shows that NeighborSwitch can be applied for v, v' and it will free up a stub for v. Then (v,w) edge can be added without further rewiring.

We have to consider whether NeighborSwitch operation can add (v,w) edge if $k = l$ and $w = v'$ such that it remains possible to add it after the switch. Node t used during the switch is different from w, thus the edge added during the switch is different from (v,w).

Case 3. Add a new edge between v, w nodes w/out free stubs. Similar to Case 2. application of Lemma 2 gives $v' \in V_k$ and $w' \in V_l$ w/free stubs for v, w respectively. NeighborSwitches can be then applied to v, v' and w, w'. The resulting free stubs for v, w can be used to add (v,w).

Subsequent applications of NeighborSwitches will not add (v,w) even if $k = l$, because the first switch clearly uses $v' \neq w$ and the second can be handled as in Case 2. □

Lemma 1. *If $JDM(k,l) < JDM^{\odot}(k,l)$, then an edge can be added between V_k and V_l .*

Proof. This follows from realizability conditions [I] and [II]. Let us assume that it is not possible to add a new edge between nodes in V_k and V_l. This implies that nodes in V_k and V_l and current edges build a complete bipartite graph (or complete graph if $k = l$):

$$JDM(k,l) = JDM_{max}(k,l) = \begin{cases} |V_k| \cdot |V_l|, & \text{if } k \neq l \\ |V_k| \cdot (|V_k| - 1), & \text{if } k = l \end{cases} \quad (6)$$

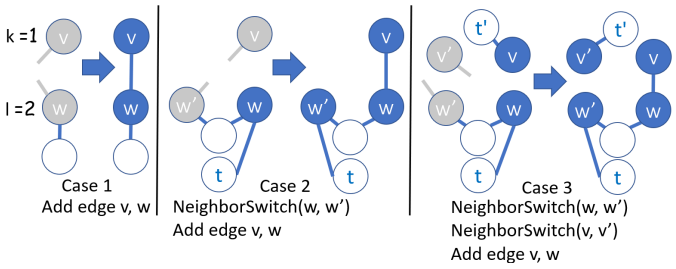


Fig. 3. All possible cases of adding an edge (v, w) between node v (of degree k) and node w (of degree l). Case 1. shows the simplest case when both nodes v and w have free stubs and no neighbor switch is necessary to add an edge between them. In Case 2., w has no free stubs and a neighbor switch is needed. (*Note:* t is the node we use to perform the switch: t is neighbor of w but not a neighbor of w'. Lemma 3 guarantees the existence of at least one such neighbor; if multiple options exist, we pick one at random.) In Case 3., both v and w have no free stubs and we need to perform two neighbor switches. Blue color nodes without free stubs and edges that resulted from connecting two stubs. Grey color indicates nodes that have free stubs and stubs that are not used yet.

Since $JDM^{\odot}$ for a realizable $JDM^{\odot}(k,l) \leq JDM_{max}(k,l)$, $\forall(k,l)$, which leads to a contradiction with $JDM(k,l) < JDM^{\odot}(k,l)$. □

Lemma 2. *If $JDM(k,l) < JDM^{\odot}(k,l)$, there is at least one node $x_k \in V_k$ with free stubs of degree k and one node $x_l \in V_l$ with free stubs.*

Proof. Let us assume that there is no node of degree k with free stubs. This means that every node $x \in V_k$ has k connected stubs and zero free stubs. This happens in two cases:

- $\forall m,\ JDM(k,m) = JDM^{\odot}(k,m)$, which contradicts $JDM(k,l) < JDM^{\odot}(k,l)$.
- $\exists m : JDM(k,m) > JDM^{\odot}(k,m)$, which contradicts the algorithm's invariant that $\forall(k,l), JDM(k,l) \leq JDM^{\odot}(k,l)$ (lines 2 and 9 of `2K_Simple` algorithm).

This is a contradiction. So does assuming that no x_l of degree l, $k \neq l$ has free stubs. □

Lemma 3. *NeighborSwitch is possible to execute and it is JDM preserving if $w, w' \in V_k$ and $deg(w') < deg(w)$.*

Proof. Since $deg(w') < deg(w)$, there exists a node t ($t \neq w', t \in V_l$, where k could be equal to l), which is a neighbor of w but not a neighbor of w'. Therefore, it is possible to remove edge (w,t) and add edge (w',t) without creating multi-edges or self-loops. Since a NeighborSwitch removes exactly one edge (w,t) and adds exactly one edge (w',t), the number of edges between nodes of degree k (*i.e.,* $w, w' \in V_k$) and nodes of degree l (*i.e.,* $t \in V_l$) will remain the same, before and after the switch, therefore the value of $JDM(k,l)$ will not change. The NeighborSwitch is JDM preserving, and no updates to JDM are needed. □

Lemma 3 guarantees that, if construction has not terminated, there will always be at least one suitable t (neighbor of w but not neighbor of w') to perform the NeighborSwitch. In case there are multiple such neighbors t, picking any one of those candidates to perform the NeighborSwitch will work, since they all preserve the JDM. Although the choice of eligible neighbor, t, to perform the NeighborSwitch does not affect the correctness of the algorithm, it may affect the exact (not asymptotic) running time and the properties of the resulting realization. In our implementation, we purposely pick one random such neighbor, to avoid introducing bias in the structure of realizations.

Running Time. The running time of `2K_Simple` is $O(|E| \cdot d_{max})$, *i.e.,* linear in the number of edges.

Proof. In each iteration of the `while` loop, one edge is always added, until we add all $|E|$ edges. However, we have to consider how much time it takes to pick nodes (v,w) and the cost of NeighborSwitch operations.

Naively chosen node pairs would become an issue for dense graphs, since there could be NeighborSwitches that remove previously added edges or add edges between the two degree groups. A simple solution is to keep track of $JDM^{\odot}(k,l) - JDM(k,l)$ many node pairs where edges can be added in a set P. For every pair of k,l, it is possible to initialize P by passing through at most $JDM^{\odot}(k,l)$ node pairs. A new (v,w) node pair can simply be chosen as a (random) element from P. If a neighbor switch for $v \in V_k$ (and similarly to w), rewires a neighbor $t \in V_l$, then $P = P \setminus (v',t) \cup (v,t)$ maintains available node pairs in P. Note: (v',t) might not be in P. This ensures that $|P| \geq JDM^{\odot}(k,l) - JDAM(k,l)$. These simple set operations can be done in constant time, and building P takes $O(E+V)$ time over all partition class pairs. Finally we remove (v,w) from P once the edge is added.

It takes $O(d_{max})$ time to choose a neighbor, t, of a node without free stubs, v, for NeighborSwitch, because the sets of neighbors can be at most $|d_{max}|$ and set difference takes linear time in the size of sets. Keeping track of nodes with free stubs allows us to pick v' for NeighborSwitch in constant time. In the worst case, there are at most two NeighborSwitches per new edge, hence the running time is $O(|E| \cdot d_{max})$. □

A tighter upper bound can be obtained by counting the running time of a NeighborSwitches for each degree group. We can express the number of edges E as a sum of stubs attached over nodes in each degree group k: $|E| = \sum_k D_k \cdot k/2$. In the worst case that each of these stubs will need a neighbor switch during an edge addition, the running time would be $O(\sum_k D_k \cdot k \cdot (k-1)/2) = O(\sum_k D_k \binom{k}{2})$, which is the number of paths of length two in the graph.

Space complexity. The input size proportional to the number of nonzero elements of the JDM, that is $O(d_{max}^2)$. The algorithm produces a graph that requires $O(V+E)$ space, while using temporary data structures. Therefore, `2K_Simple` uses the minimum space as it is required to store the final output graph. The details of the space requirements are as follows.

`2K_Simple` requires constant look up time for nodes with free stubs, this can be achieved by storing an array of sets, where each set contains the nodes with free stubs for a given degree group. The size of this data structure is initially $O(V)$ and decreases over the execution of the algorithm.

As discussed in the proof, `2K_Simple` also maintains pairs of nodes (set P) for candidates to add edges. The size of P is $O(JDM(k,l))$ for a given k,l degree group pair and $O(\sum JDM) = O(E)$ over all iterations of the algorithm. (This is easiest to see in the example of targeting k-regular

graphs, where P stores a set of $|E|$ candidate pairs initially, and we can keep track of nodes with free stubs in a single set of size $O(V)$.)

3) Relation to related work: `2K_Simple` adds an edge at a time while maintaining the following invariant for every $k, l : JDAM(k,l) < JDM^{\odot}(k,l)$ and $\forall v \in V_k : deg(v) \leq k$. This idea was also presented independently in [18]. Another approach, followed by [18], [19] and [9], is to add all edges between V_k, V_l according to $JDM^{\odot}(k,l)$ without considering degrees. This could create nodes with higher degree than their assigned degree group requires, which can then be resolved by using NeighborSwitch operations. Special realizations with the BDI property can also be constructed using these ideas: [18] adds further restriction to the first approach such that at every edge addition the BDI property is maintained, while [19] provides an algorithm using the second approach, where no NeighborSwitch operations are required.

The common idea behind all 2K construction algorithms is to add one edge at the time so as to not exceed node degrees or JDM; algorithms vary on if/how they violate any of these two properties and on when/how to correct it using local rewiring (NeighborSwitch in our terminology); which is always possible for realizable JDMs. Please see Section II-D for a timeline. The particular order used in which `2K_Simple` adds and rewires edges is essential for being able to target additional properties, during construction, thus enabling a framework beyond just 2K.

`2K_Simple`'s run time complexity is comparable to other proposed 2K algorithms. Interestingly, it is even comparable to 1K construction algorithms that produce any 1K realization with non-zero probability [16], [17]. Indeed, $O|E|$ is the minimum required to construct a graph with $|E|$ edges.

4) Space of Realizations: The order in which `2K_Simple` adds edges is unspecified. The algorithm can produce any realization of a realizable JDM, with a non-zero probability. Considering all possible edge permutations as the order in which to add the edges, the ones where no neighbor switch is required correspond to all the possible realizations. Unfortunately, the remaining orderings are difficult to quantify, thus the current algorithm cannot sample uniformly from all realizations with a target JDM during construction. An experimental validation is shown in Appendix B-A in comparison with BDI realizations and the configuration model.

Fortunately, once one realization is constructed (using `2K_Simple`), it is possible to sample from the space of all realizations, using edge-rewiring. In particular, it has been shown that JDM realizations are connected via 2K-preserving double-edge swaps [9], [18]. This method is typically used by MCMC approaches that transform one realization to another by rewiring edges so as to preserve target properties.

In the next subsections, we exploit the flexibility of `2K_Simple` and we extend it to target additional properties, in addition to $JDM^{\odot}$. In Section III-B we control (approximately) the average clustering by controlling the order in which edges are added. In Section III-C we impose (exactly) node attributes by exploiting the flexibility in the number of degree groups with the same assigned degree. In Section III-D we consider the space of realizations with a target number of connected components.

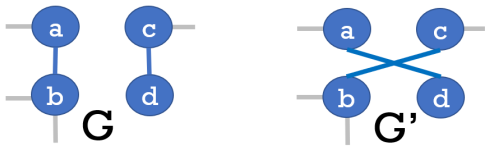


Fig. 4. A **double-edge swap** is a rewiring of edges (a,b), (c,d) to (a,d), (b,c) where a,b,c,d are four distinct nodes (to avoid self-loops) and (a,d), (b,c) are not present before rewiring (to avoid multi-edges). If deg(a) = deg(c) then the swap obviously preserves the JDM of the graph. It is referred to as **JDM-preserving double-edge swap** and it is used in MCMC to transform graph G to other realizations G' with the same JDM, while targeting other properties.

B. Target JDM and Clustering

Recently, a member of our group and collaborators proved that the realizability of JDM and a fixed number of triangles is NP-Complete [11]. This motivated us to design efficient heuristics that target different notions of clustering (namely 2.25K when average clustering $\bar{c}^{\odot}$, is targeted and 2.5K when degree-dependent clustering $\bar{c}^{\odot}(k)$, is targeted).

MCMC Approach. In the original dK-series paper [5], 2K-preserving 3K-targeting was attempted via the classic JDM-preserving double edge swap as follows. Starting from a JDM realization, randomly select edges (a,b) and (c,d) such that $deg(a) = deg(c)$, as in Fig. 4; perform a double-edge swap iff it brings the graph closer to the target 3K (according to a well-defined distance metric), accept the rewiring. Unfortunately, this happens with a very small probability and the naive MCMC approach was very slow in practice, taking weeks or months to produce a single realization for large graphs.

We improved the 2K-preserving clustering-targeting MCMC by carefully selecting candidate edges to swap so as to control the number of triangles: select edges with low number of shared partners to create triangles, select random edges to destroy triangles. The rationale is that it is easier to destroy than create triangles. Although this reduced the running from weeks to days we still faced scalability problems.

Construction Approach. We modify `2K_Simple` so as to control the order in which edges are added and create the target clustering *during* the 2K construction, not with MCMC after that. Let E' be any permutation (order) of possible node pairs $\{v,w\}$. We follow the order in E' when we consider adding edges in Algorithm 1, line 3: if two node pairs $E'_i = (v_i, w_i)$ and $E'_j = (v_j, w_j)$ are s.t. $i < j$, then edge (v_i, w_i) will be considered for addition (line 5 in `2K+S`) before (v_j, w_j). The key question is: what is the right order E' of adding edges so as to control clustering?

Figure 5 depicts our approach. We assign every node $v \in V$ to a coordinate r_v randomly selected from a one-dimensional coordinate system $(0,1)$. We define the distance of v and w as $dist(v,w) = \min(|r_v - r_w|, 1 - |r_v - r_w|)$. If we add edges in increasing distance, we connect nodes near each other, thus creating many triangles among nearby nodes in the coordinate system (as on the right side of the figure). If we add edges in random order, we create very few triangles (as shown on the left side of the figure). If we control the fraction of edges that

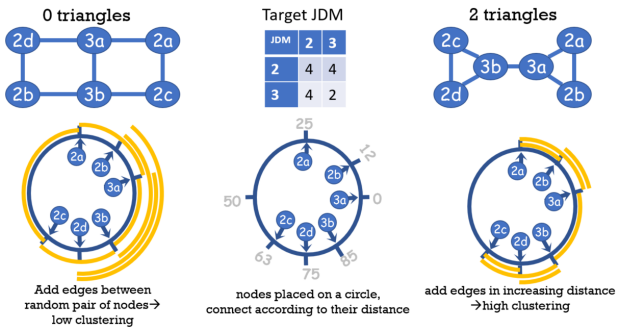


Fig. 5. Approach for targeting clustering during 2K construction. Nodes are assigned random coordinates on a circle (middle). `2K_Simple` runs with the target JDM, but we control the order in which to add edges. If we add edges between node pairs in increasing distance on the circle, then we tend to create many triangles locally (right). If we add edges in random order, then we tend to create few triangles (left). All these graphs have the same JDM (middle top) constructed by 2K, but their clustering is controlled by the order in which we consider adding edges in `2K_Simple`.

are added in increasing distance vs. at random, we can control the clustering.

1) 2.25K: Targeting JDM and Average Clustering: We introduce parameter S to control the sortedness of E'. Two node pairs E'_i and E'_j are inverted in an order E'' iff $(i < j)$ *and* $dist(v_i, w_i) > dist(v_j, w_j)$. We define the *sortedness* of a list E' as the fraction of non-inverted node pairs: sortedness(E') $= 1 - \frac{\text{number of inversions in list E'}}{|E'|(|E'|-1)/2} \in [0, 1]$.

We experimented with the effect that an order of node pairs E' has on the structure of the generated graph and we found that the sortedness S is positively correlated with the average clustering coefficient, $\bar{c}$, of the graph; see details on tuning parameter S in Appendix C. This is intuitively expected [23]: Values of sortedness(E') close to 0 produce graph instances with minimum clustering over all graph instances on average. Values of sortedness(E') close to 1 produce graph instances with maximum clustering over all graph instances on average.

Algorithm. Our algorithm for achieving exactly $JDM^{\odot}$ and approximate clustering by controlling the sortedness S of adding nodes is summarized next. In the first stage, it attempts to add edges using a given order E' of all possible node pairs defined by $order(List, S)$. The function $order(List, S)$ (used in line 2 of Algorithm 2) determines the order of node pairs E', that will be considered for addition, so as to (approximately) set as S the sortedness of the input $List$.

Similarly to `2K_Simple`, it only adds edges if the current $JDM(k,l)$ value does not exceed the target $JDM^{\odot}(k,l)$. Differently than `2K_Simple`, it has an additional constraint: it adds edges between two nodes (v, w) *only if* there are free stubs to connect the nodes $(deg(v) < k,\ deg(w) < l)$. Therefore at the end of the first stage, despite considering all possible node pairs, there might be some nodes with free stubs, since we do not allow multi-edges or self-loops. In Stage 2, we use algorithm `2K_Simple`, starting from the partially built graph at the end of Stage 1: we add edges between any remaining nodes with free stubs and complete the graph. It follows directly from the properties of `2K_Simple`, that this will produce the exact $JDM^{\odot}$.

Running Time. The time complexity of `2K+S` is similar to `2K_Simple` for adding edges (*i.e.,* $O(|E| \cdot d_{max})$), plus the time for the function order($List$, S). If the latter is properly implemented, the running time remains linear in the number of edges; see Appendix C for details.

Space Complexity. Naive implementation of `2K+S` would require to generate all possible edges taking $O(|V|^2)$ space. An improved implementation only adds $O(|E|)$ edges which reduces the space complexity to the overall $O(|V| + |E|)$ as before; see Appendix C for details.

2) 2.5K: Target $JDM^{\odot}$ and Degree-Dependent Clustering: Targeting $\bar{c}^{\odot}(k)$ is even more challenging than targeting $\bar{c}^{\odot}$. Our intuition from MCMC was that it is difficult to find a double edge swap that creates triangles while it is easier to destroy triangles. Therefore, we propose to (1) create a 2K realization with many triangles (such as a `2K+S` with $S = 1$) and (2) use the improved MCMC described above to destroy triangles. This indeed worked well in practice in terms of targeting $\bar{c}^{\odot}(k)$ and running time; see Section III-E.

Algorithm 2: `2K+S`

Input: $JDM^{\odot}, S$
Stage 1:
1: $E = \{\}$
2: $E' = order(\{(v,w) : \forall v, w \in V\}, \text{sortedness} = S)$
3: **forall** $\{v, w\} \in E'$ **do**
4: $v \in V_k,\ w \in V_l$
5: **if** JDM$(k,l) <$ JDM$^{\odot}(k,l)$ **and** $deg(v) < k$ **and** $deg(w) < l$ **do**
6: $E \leftarrow E \cup \{v, w\}$
7: $JDM(k,l)$++ ; $JDM(l,k)$++
Stage 2:
8: **if** $\sum$JDM$(k,l) <$ $\sum$JDM$^{\odot}(k,l)$ **do**
9: Finish graph construction using `2K_Simple`

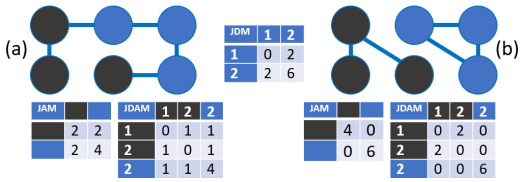


Fig. 6. Example of two different graphs, (a) and (b), with the same Joint Degree Matrix (JDM) and different Joint occurrence of Attributes Matrix (JAM), thus different JDAMs. We assign the color black to nodes with the first attribute and blue to nodes with the second attribute.

C. 2K+A: Targeting JDM and Node Attributes

JDM vs. JAM. JDM only describes correlations between the degrees of connected nodes. However, in many contexts, capturing correlations of node attributes in the network model better characterizes the graph [24], [31], [32]. For example, in social networks, the similarity of attributes between two nodes often affects the creation of an edge between them. We assume that there is a set of categorical attributes with p possible values. Each node $v \in V$ can be assigned to only one categorical attribute. Let A_i be the set of nodes that have attribute i, for $i = 1, ..., p$. We can define the *Joint occurrence*

of Attributes Matrix (JAM) as the number of edges connecting nodes in A_i with nodes in A_j.

$$\text{JAM}(i,j) = \sum_{v \in A_i} \sum_{w \in A_j} 1_{\{\{v,w\} \in E\}}. \quad (7)$$

However, JAM alone does not capture the network structure. In Fig. 6, we show a toy example. The graphs in Fig. 6(a) and Fig. 6(b) have the exact same JDM but different JAM. And conversely, examples of networks with the same JAM and different JDM can be constructed as well.

JDAM. We propose to incorporate correlations of node attributes on top of the JDM matrix as follows. If V_k is the set of nodes that have degree k for $k = 1, ..., d_{max}$ and A_i the set of nodes that have attribute i, for $i = 1, ..., p$, then let the degree-attribute group $B_{\{k,i\}} = \{v | v \in V_k, v \in A_i\}$ be the set of nodes that have degree k and attribute i. The number of degree-attribute groups is at most $d_{max} \cdot p$. We define the *Joint Degree and occurrence of Attributes Matrix* (JDAM) as the number of edges connecting nodes in $B_{\{k,i\}}$ with nodes in $B_{\{l,j\}}$ for degree-attribute groups $\{k,i\}$ and $\{l,j\}$.

$$\text{JDAM}(\{k,i\},\{l,j\}) = \sum_{v \in B_{\{k,i\}}} \sum_{w \in B_{\{l,j\}}} 1_{\{\{v,w\} \in E\}}. \quad (8)$$

Example JDAMs are shown in Fig. 6. A JDAM is similar to JDM, but each row now describes not only a degree k but a degree-attribute pair $\{k,i\}$; and similarly for the columns.

It turns out that `2K_Simple` can be gracefully extended to construct a simple graph with a target $JDAM^{\odot}$ as shown in Algorithm 3 in Appendix A. We can observe that our proofs (and others from related work) depend on the fact that within a degree group degrees are equal among nodes, however these proofs do not have restrictions on how many times a degree group appears with the same degree. We can apply and trivially extend earlier results including sufficient and necessary conditions for realizability, construction algorithms, existence of BDI realizations, importance sampling algorithm extensions from JDM, connectivity of space of realizations over $JDAM$ preserving double-edge swaps and MCMC properties.

The running time and space complexity analysis follows `2K_Simple`. The only change is that the JDAM input has a size of $O((d_{max} \cdot p)^2)$, where a sparse representation is better characterized by the number of non-zero JDAM entries or $O((\text{number of observed node degree and attribute combinations})^2)$.

D. 2K+CC: Number of Connected Components

In this section, we consider the number of connected components (CC) for a target JDM – a property which has not been explicitly targeted or characterized in the past. An algorithm for constructing graphs with a realizable JDM and a single CC, if such exist, has been provided in [18]. However, there may be JDM realizations with a different number of connected components k, s.t. $1 \leq k_{min} \leq k \leq k_{max}$. Our main result on this problem is the following:

Theorem 4. *The space of simple, undirected graphs with a target JDM and no more than $k^{\odot}$ number of CCs is connected under a sequence of JDM-preserving double-edge swaps.*

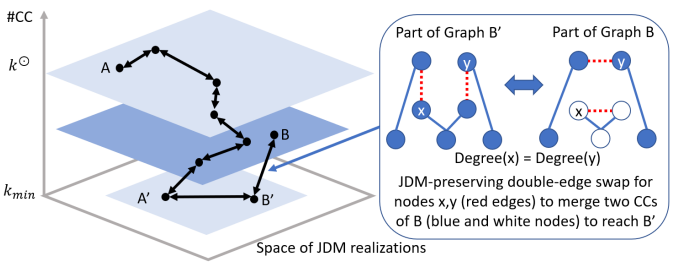


Fig. 7. Space of JDM realizations with up to $k^{\odot}$ CCs.

There are counterexamples that show that the above statement is not true for *fixed* k, *i.e,* realizations with *exactly* k CCs are not necessarily connected under double-edge swaps. However, we show that the JDM realizations with a number of CCs *up to a maximum target number,* $k \leq k^{\odot}$, is connected over double-edge swaps.

Figure 7 depicts how every pair of realizations, (A, B), can be reached via a sequence of JDM-preserving double-edge swaps where every intermediate realization has less than the maximum of A and B's number of CCs. Lemma 5 uses double-edge swaps that merge CCs and decrease k. Therefore both A and B can be transformed to A', B' with the same JDM and the minimum number of CCs, k_{min}. Lemma 6 guarantees that A' and B' can also be transformed to each other via the same types of swaps. More discussion and proofs for Lemma 5 and 6 can be found in Appendix D.

Lemma 5. *There exists a JDM-preserving double-edge swap sequence that transforms any JDM realization to a realization with minimum number of CCs, such that there is no double-edge swap that increases the number of CCs.*

Lemma 6. *The space of JDM realizations with minimum number of CCs, k_{min}, is connected under JDM-preserving double-edge swaps.*

We could also target balanced (BDI) realizations of 2K+CC. Lemma 4 and Corollary 5 from [9] show how to apply double-edge swaps to get balanced realizations from any JDM realization. We make the crucial observation for our results that in Lemma 4, while choosing neighbors for double-edge swaps, nodes have at least two options to pick from. This is sufficient for us to show how to construct balanced *and* minimum number of connected components realizations at the same time; see Appendix F.

E. Simulations for Real-World Undirected Graphs

In this section, we perform simulations targeting properties of real-world undirected graphs, and we evaluate the performance of different construction algorithms in practice.[5] This is also a use-case of our work: people often need to produce topologies that resemble graphs like the online social networks

[5]The algorithms were implemented in Python using NetworkX [33] and executed on an AMD Opteron 2.4Ghz machine. A C++ implementation would be potentially faster by a constant factor especially, if combined with a more recent, faster CPU. We also contributed our software to NetworkX [34]. However, the focus in this section, is the comparison of different algorithms.

TABLE I
REAL-LIFE TOPOLOGIES USED FOR EVALUATION.

Dataset	$\|V\|$	$\|E\|$	Avg Deg.	$\bar{c}$	# dorms	# years
FB: Rice [35]	4 087	184 828	90.45	0.294	10	22
FB: Princeton [35]	6 596	293 320	88.94	0.237	57	27
FB: UCSD [35]	14 948	443 221	59.30	0.227	40	23
FB:New Orl. [36]	63 392	816 884	25.77	0.222	-	-
amazon0601 [37]	403 364	2 443 309	12.11	0.42	-	-
youtube-links [38]	1 134 894	2 987 623	5.26	0.081	-	-

listed in Table I. The main finding of this evaluation is that our construction algorithms can target 2K+clustering well, and orders of magnitude faster than prior MCMC state-of-the-art: reducing the time from days and weeks (or not even terminating for large graphs, to minutes and tens of seconds.

1) Datasets: We evaluate all proposed algorithms in terms of accuracy and efficiency on a variety of real-world topologies. Table I summarizes the topologies, which are divided in two groups. Those in the first group are relatively smaller (*i.e.,* up to $15K$ nodes) Facebook university networks (Rice, Orinceton, UCSD) that come with several annotated node attributes. The second group consists of larger graphs (*i.e.,* more than $60K$ nodes) without node attributes (FB: New Orleans, amazon0601, youtube-links).

First, we compute the properties (JDM, clustering, etc) of the *original* real topology (last line in each topology), then we use different construction algorithms to target those properties (listed on the lines above). For each topology, we construct 20 realizations with different algorithms: `2K_Simple`, `2K+S` with two different values of the clustering parameter S ($S = 1$ and another S selected to target $\bar{c}$), and `2K_Simple_Attributes` where node attributes are "dorms" and "year of admission". Table II reports the properties of the original and constructed graphs, including properties explicitly targeted ($\bar{c}$ and the degree assortativity for dorms, and year) and non-targeted ones (average shortest path length, average closeness, number of maximal cliques), averaged over realizations.

The last two columns report the time (in sec) to construct the graph. The second column from the end ("Construction") refers to the time it takes for the construction algorithm to terminate. The last column ("MCMC") refers to the *additional time* that our improved MCMC would use, starting from the realization the construction algorithm produced, to further target degree-dependent clustering within $NMAE < 2\%$.

2) Properties: Here we examine whether targeting either $\bar{c}$ or attributes, in addition to JDM, brings the constructed graphs closer to the original w.r.t. other non-targeted properties as well. For the first three small graphs, we observe that the average shortest path and average node closeness of graphs produced by `2K_Simple` is already close to the original graph. Thus, targeting $\bar{c}$ does not match better the non-targeted properties or the assortativity of node attributes, which stays close to zero. However, targeting a given attribute significantly improves $\bar{c}$ and the assortativity of the second attribute (*e.g.,* $2K + dorms$ in Princeton) in addition to exactly achieving assortativity for "Dorms", also improves $\bar{c}$ from 0.04 to 0.08 and assortativity for the attribute "Year" from 0.01 to 0.27 when compared to `2K_Simple`.

TABLE II
GRAPHS ARE CONSTRUCTED TARGETING DIFFERENT PROPERTIES OF 6 DIFFERENT ORIGINAL TOPOLOGIES. GRAPH PROPERTIES ARE AVERAGED OVER 20 RUNS. THE LAST TWO COLUMNS REPORT THE TIME (IN SEC) FOR THE CONSTRUCTION ALGORITHMS AND FOR MCMC TO TARGET $\bar{c}(k)$.

Topology	Graph	Graph properties						Constr. Time (sec)	MCMC Time (sec)
		Avg Node Value			Number of	Assortativity			
		$\bar{c}$	Sh.P.	Closn.	Cliques	Dorms	Year		
FB Rice	2K Simple	0.06	2.33	0.43	425K	0.01	0.01	2.52	9091
	2K+S=1.0	0.53	2.74	0.37	11.5M	0.01	0.01	20	193
	2K+S=0.52	0.29	2.68	0.38	3.1M	0.01	0.01	21	249
	2K+dorm	0.13	2.38	0.43	793K	0.42	0.09	3.45	773
	2K+year	0.09	2.36	0.43	500K	0.08	0.28	3.43	2249
	original	0.29	2.44	0.41	1.1M	0.42	0.28	-	-
FB Princeton	**2K Simple**	0.04	2.49	0.40	530K	0.00	0.01	3.69	16K
	2K+S=1.0	**0.55**	2.97	0.34	29.0M	0.00	0.01	29	**274**
	2K+S=0.57	**0.24**	2.97	0.34	9.8M	0.00	0.01	**27**	**331**
	2K+dorm	0.10	2.56	0.40	751K	0.09	0.27	5.70	2324
	2K+year	0.08	2.59	0.39	755K	0.05	0.45	5.32	2157
	original	**0.24**	2.67	0.38	1.3M	0.09	0.45	-	-
FB UCSD	2K Simple	0.01	2.86	0.35	438K	0.00	0.01	4.90	66K
	2K+S=1.0	0.63	3.39	0.30	3.7M	0.00	0.01	46	920
	2K+S=0.61	0.23	3.46	0.29	5.4M	0.00	0.01	43	1656
	2K+dorm	0.03	2.88	0.35	526K	0.25	0.05	8.45	30K
	2K+year	0.02	2.89	0.35	476K	0.02	0.36	7.52	42K
	original	0.23	2.98	0.34	743K	0.25	0.36	-	-
FB: New Orl.	2K Simple	0.00	3.89	0.26	760K	-	-	18.65	524K
	2K+S=1.0	0.58	4.46	0.23	1.5M	-	-	79	3150
	2K+S=0.74	0.30	4.56	0.22	2.4M	-	-	74	9360
	original	0.22	4.35	0.24	1.5M	-	-	-	-
amazon0601	2K Simple	0.00	4.84	0.21	2.4M	-	-	53	∞
	2K+S=1.0	0.61	6.04	0.17	637K	-	-	239	∞
	2K+S=0.73	0.42	5.92	0.17	1.1M	-	-	214	∞
	original	0.42	6.39	0.16	1.0M	-	-	-	-
youtube-links	2K Simple	0.00	4.64	0.22	2.9M	-	-	71	∞
	2K+S=0.69	0.14	5.07	0.18	2.4M	-	-	955	∞
	2K+S=1.0	0.21	4.82	0.20	2.2M	-	-	1073	∞
	original	0.08	5.34	0.19	3.3M	-	-	-	-

In the three larger graphs, targeting a higher $\bar{c}$ than what is achieved by `2K_Simple` brings the average path length and closeness significantly closer to the original graph. For example, in the *amazon0601* topology `2K_Simple` achieves an average node closeness of 0.21, whereas $2K + S = 0.73$ achieves 0.17 which is closer to the real value of 0.16. Finally, for all graphs, the property "Number of Cliques" does not consistently improve by targeting either $\bar{c}$ or attributes.

3) Efficiency: The time needed to generate a graph using either of our three construction algorithms is similar, which is expected since they all run in linear time in $|E|$. For example, it takes tens of seconds to generate synthetic graphs for all Facebook topologies of Table I even when we target maximum clustering (*i.e.,* $2K$+$S = 1.0$). As a baseline for comparison, we also targeted $2K$+$\bar{c}$ (*i.e.,* 2.25K) with MCMC using double edge swaps, which is the previous state-of-the-art. With naive MCMC, it took approximately *a day* to generate synthetic graphs with the target $\bar{c}$ for the smallest topologies (Rice and Princeton); while simulations for the bigger graphs did not finish after several days.

Recall that the last column of Table II reports the time that a 2K-preserving MCMC needs to target the degree-dependent clustering of the original graph (2.5K), starting from the realization constructed by our (2K or 2.25K) construction algorithm. We observe that the time for the MCMC to target 2.5K increases as we decrease the average clustering in the generated graphs. We also observe that the larger the graph,

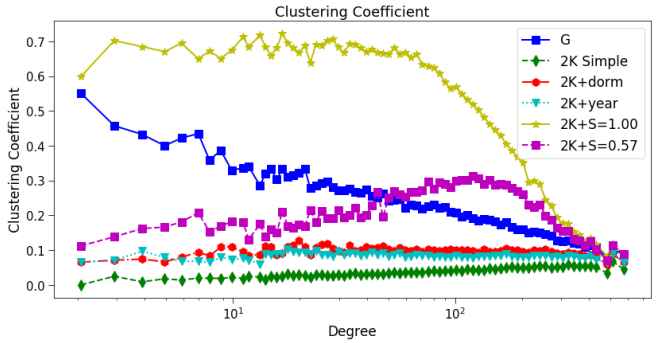


Fig. 8. Average degree-dependent clustering coefficient for the FB Princeton graph. Figure shows $\bar{c}(k)$ for the original graph (for which $\bar{c} = 0.24$) and for a realization produced by each construction method. `2K_Simple`, 2K+year, 2K+dorm achieve low clustering ($\bar{c} = 0.04, 0.1, 0.08$), much lower than the original graph ($\bar{c} = 0.24$). $2K + S = 0.57$ matches average clustering ($\bar{c} = 0.24$) but not $\bar{c}(k)$. $2K + S = 1$ significantly overshoots the real $\bar{c}(k)$ in most degree groups. Starting from the realization produced by each construction, we can target the original $\bar{c}(k)$ within $NMAE < 2\%$, in time reported in the last column of Table II. It turns out that constructing for highest clustering $2K + S = 1$ and then using MCMC to target $\bar{c}(k)$ gets there the fastest (274sec in Table I.)

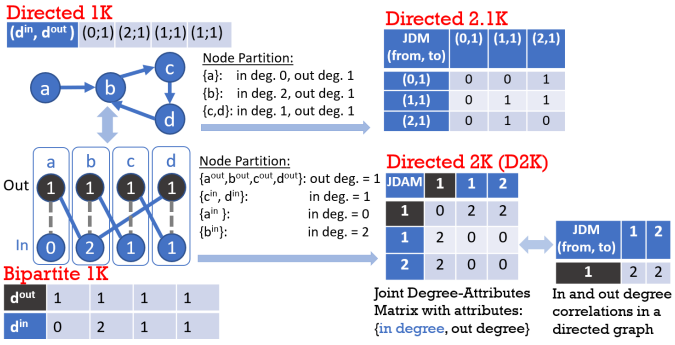


Fig. 9. Defining Directed 2K, to capture degree correlations in a directed graph. **Top left, Directed 1K:** Directed graph with a given degree sequence (DDS). **Bottom left, Bipartite 1K:** Mapping of the previous to a Bipartite undirected graph with a given bipartite degree sequence; non-chords in the bipartite graph (shown in dashed line) correspond to self-loops in the directed graph. **Bottom right, Directed 2K (D2K):** Joint-Degree-Attribute Matrix (JDAM), where nodes of the bipartite graph are partitioned by their degree-and-(in or out) attribute or equivalently the in and out degree correlations of the directed graph. **Top right: Directed 2.1K:** JDM for directed graphs, where nodes are partitioned according to their (in degree, out degree).

the worse the MCMC matches the graph. For example, in the largest examples (Amazon, Youtube graphs), our improved MCMC did not successfully target 2.5K in these cases. One reason behind is that the cost of local updates and number of swaps is large for large graphs.

Finally, we observe that construction targeting maximum average clustering (*i.e.,* $2K+S = 1$) has a faster MCMC than 2.25K construction, if the end goal is to follow up construction with MCMC to target 2.5K. Figure 8 further elaborates on this point by zooming in on the FB Princeton graph: the blue graph is the real 2.5K ($\bar{c}(k)$) of the original graph; all other curves plot the $\bar{c}(k)$ achieved by all other construction methods. The latter serves a starting point for the 2.5K-targeting MCMC, at the end of which $\bar{c}(k)$ is within $NMAE < 2\%$ of the target $\bar{c}(k)$. The yellow graph on top shows the $\bar{c}(k)$ at the end of $2K + S = 1$ (*i.e.,* maximum clustering); it turns out that starting from there and using MCMC is the fastest (274 sec) in Table II), because it is easier to destroy rather than create triangles with MCMC. The purple graph corresponds to 2.25K ($2K + S = 0.57$), which does not match the degree-dependent clustering compared to the original graph and takes longer to fix with MCMC (331sec) in Table II.

4) Discussion: The benefits of our approach, compared to prior MCMC approaches are two-fold: (i) accuracy, *i.e.,* how well we match 2K (exactly), and average clustering $\bar{c}$ (approximately) and (ii) construction running time. As we can see in Table II, both approaches and MCMC can achieve close to $\bar{c}$ (column 2), if allowed to run long enough. However, as it can be seen in the last two columns of Table II, our running time is on the order of seconds or up to tens of seconds, while MCMC running time varies from 100s of seconds (minutes) up to hundreds of thousands of seconds (several weeks); in the cases of the larger graphs (amazon, youtube), the MCMC approach does not even converge to the target (thus ∞ time). The magnitude of the reduction of running time depends on the graph characteristics, and is amplified when the target graphs (i) are large (*i.e.,* large $|V|, |E|$), (ii) exhibit high clustering (*e.g.,* see original $\bar{c}$ in Table II), and (iii) are sparse (as indicated by their average degree in Table I).

The Facebook university graphs all have almost the same $\bar{c}$ and are ordered in increasing size and sparsity in Table I: Rice, Princeton, UCSD, New Orleans. In Table II, we can see that the corresponding difference in running time is in the same order, *i.e.,* amplified with size and sparsity. The amazon dataset is an order of magnitude larger and sparser but has a higher target average clustering than the Facebook networks. In this case, the MCMC never converges (indicated by ∞ time in the last column of Table II), while our algorithms still terminate on the order of minutes. The reason is that it is highly unlikely to create triangles by chance (MCMC or pure 2K), compared to our more structured 2K+ construction (where we create as many triangles as we can using $2K + S = 1$), then we destroy triangles using an improved MCMC). Fig. 8 shows an example of how sortedness was used to overshoot degree-dependent clustering before applying MCMC. Therefore, targeting sparser graphs with higher clustering is more challenging for the MCMC approach, while 2K+S was significantly faster. Sparse (pairs of) degree groups tend to have low clustering if we only consider 2K (not 2K+S). We have not experimented with datasets where the graph is dense and the target clustering is low (but realizable); our intuition is that even 2K construction would achieve close to target clustering in that case, since it tends to generate graphs with low clustering. Finally, sparsity can affect the running time of our algorithm in practice (asymptotically it is still $O(|E| \cdot d_{max})$) in a different way: sparse graphs might require fewer NeighborSwitches (the most expensive operation in our algorithm) compared to dense graphs.

IV. Directed Graphs

A. Defining Directed 2K

Our goal in this paper is to go beyond just directed degree sequence and capture directed degree correlation. One

approach would be to simply consider the degree correlations between in and out degrees in a directed graph, as shown in Fig. 9-bottom rightmost matrix. Alternatively it is possible to work with the equivalent representation of a directed graph as an undirected bipartite graph without non-chords (Fig. 9-bottom left), and define degree correlations there. We partition in and out nodes by their degree, essentially considering that nodes in the bipartite graph can have an attribute that takes two values, "in" or "out." We can now define degree correlation using the Joint Degree-Attribute Matrix (JDAM), as shown on Fig. 9-bottom right. This leads to a JDAM with two attribute values, such that $\forall k,l = 1,...,d_{max}$ degrees and $i \in \{in, out\}$ attribute values $JDAM(\{k,i\},\{l,i\}) = 0$, *i.e.,* because the bipartite graph has no edges between two "in" or two "out" nodes. Furthermore, the number of non-chords will be noted as $f(\{k,i\},\{l,j\})$, where $k,l \in \{1,...,d_{max}\}$ and $i \neq j \in \{in, out\}$; f can be computed by passing through the directed degree sequence once and counting the number of entries with in-degree k and out-degree l.

We note that this notion of $Bipartite\ JDAM$ is a special case of $JDAM$ and inherits all the properties known for $JDAM$. It allows us to get rid of the directionality of the edges and handle a regular undirected JDAM using the 2K+A algorithm previously defined. For D2K, the main challenge is to show that the non-chords described by the directed degree sequence can be avoided. In summary we define the D2K problem as follows, and an example is shown on Fig. 9.

D2K. The input is two target properties, namely the JDAM$^{\odot}(\{k,i\},\{l,j\})$ with two attribute values (in and out) and the directed degree sequence DDS$^{\odot}$. The goal is to construct a simple directed 2K-graph with these target properties (construction) if such realizations exist (realizability).

B. Realizability and Algorithm

Recall that in our D2K definition, nodes are partitioned into at most $2d_{max}$ parts $V_{\{k,in\}}, V_{\{k,out\}}, k = 1,...,d_{max}$, according to the distinct combinations of degrees and attributes they exhibit and JDAM$(\{k,i\},\{l,j\})$ is indexed accordingly. For example, on Fig. 9 bottom-right, each node belongs to one of four parts $V_{\{0,in\}} = \{v \in V : d^{in} = 0\}$, $V_{\{1,out\}} = \{v \in V : d^{out} = 1\}$, $V_{\{1,in\}} = \{v \in V : d^{in} = 1\}$, $V_{\{2,in\}} = \{v \in V : d^{in} = 2\}$, and the JDAM is 3x3 (by removing rows and columns corresponding to any $V_{\{0,i\}}$, since there are no edges using these parts of any partition).

1) Realizability: Necessary and sufficient conditions for a target D2K, *i.e.,* JDAM$^{\odot}(\{k,i\},\{l,j\})$ and DDS$^{\odot}$, to be realizable are the following:

I $\forall k,l,i : JDAM(\{k,i\},\{l,i\}) = 0$
II $\forall k,l,i,j,$ if $JDAM(\{k,i\},\{l,j\}) > 0,$ $JDAM(\{k,i\},\{l,j\}) + f(\{k,i\},\{l,j\}) \leq |V_{\{k,i\}}| \cdot |V_{\{l,j\}}|$
III $\forall k,i : |V_{\{k,i\}}| = \sum_{\{l,j\}} \frac{JDAM(\{k,i\},\{l,j\})}{k}$ = number of times k appears in DDS as i.

These are generalizations of the conditions for an undirected JDM, JDAM to be realizable, and they are clearly necessary. The first condition states that every realization of the target JDAM is bipartite, *i.e.,* there should be no edges between two nodes both in "in" or "out" parts. The second condition considers edges between two ("in" and "out") parts and states that the number of edges defined by the $JDAM(\{k,i\},\{l,j\})$ plus the number of non-chords (shown as $f(\{k,i\},\{l,j\})$) should not exceed the total number of edges possible in a complete bipartite graph across the two parts. The last condition ensures that the target JDAM and the target DDS are consistent: the number of nodes with in (or out) degree k should be the same whether computed using the JDAM or the DDS. The conditions are shown to be sufficient via the constructive proof of the algorithm. Necessity of these conditions for simple graph construction are trivial.

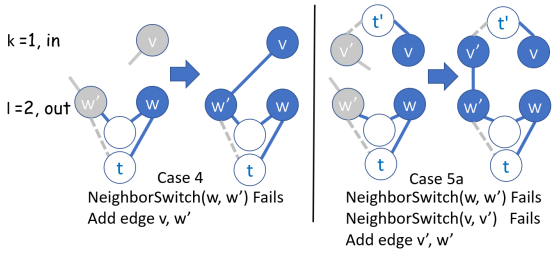


Fig. 10. New cases in Algorithm 4, while attempting to add (v,w) edge.

Algorithm 4: `D2K_Simple`

Input: $DDS^{\odot}$, $JDAM^{\odot}$
Initialization:
a: Create G with nodes, partition, stubs using $DDS^{\odot}$
b: Add non-chords to G using $DDS^{\odot}$
Add Edges:
1: **for** $(\{k,i\},\{l,j\}) \in JDAM^{\odot}(\{k,i\},\{l,j\})$
2: **while** $JDAM(\{k,i\},\{l,j\}) < JDAM^{\odot}(\{k,i\},\{l,j\})$
3: Pick any nodes $v \in V_{\{k,i\}}$, $w \in V_{\{l,j\}}$ s.t. (v,w) is not a non-chord or existing edge
4: **if** v does not have free stubs:
5: v': node in $V_{\{k,i\}}$ with free stubs
6: NeighborSwitch(v,v')
7: **if** NeighborSwitch fails, $v := v'$
8: **if** w does not have free stubs:
9: w': node in $V_{\{l,j\}}$ with free stubs
10: NeighborSwitch(w,w')
11: **if** NeighborSwitch fails, $w := w'$
12: add edge between (v, w)
13: $JDAM(\{k,i\},\{l,j\})$++; $JDAM(\{l,j\},\{k,i\})$++
Output: directed graph representation of G

2) Algorithm: First, we create a set of nodes V, where $|V| = 2|\text{DDS}|$, we assign stubs, non-chords to each node and partition nodes, as specified in the *target directed degree sequence* DDS$^{\odot}$. We also initialize all entries of JDAM to 0.

Then the algorithm proceeds by connecting two nodes (one from "in" and one from "out" side, because of Condition I, thus adding one edge (v,w) at a time, that (i) do not form an edge (ii) do not have a non-chord between them (to avoid self-loops in the directed graph representation) and (iii) for whom the corresponding entry in the JDAM has not reached its target. The added complexity from JDAM construction lies in the non-chords and the fact that NeighborSwitch operation can "fail". This failure means that there is no suitable node to

perform NeighborSwitch with due to a non-chord constraint. However, in these cases another edge can be added as shown in Fig. 10. Next, we prove that this is indeed always the case.

Proof. Condition I ensures that every realization is bipartite, Condition II guarantees that two nodes can be always chosen to add an edge following the arguments in Lemma 1 and Condition III ensures that at least one node exist with a free stub in every part of the partition if $JDAM(\{k,i\},\{l,j\}) < JDAM^{\odot}(\{k,i\},\{l,j\})$ using Lemma 2. Now, we show that every iteration can proceed by adding a new edge to the graph. The cases are identical to `2K_Simple` as long as NeighborSwitch operation can be executed without using non-chords. This leads to two additional cases to the proof earlier:

Case 4. Add a new edge between a node w w/out free stubs and a node v w/free stubs (or w/out free stubs where NeighborSwitch is possible) where NeighborSwitch is not possible for w using w' without using any non-chords. In this case w' has the same neighbors as w except the one for which it has an assigned non-chord. In this case w' is not connected to v and it is possible to add $\{w', v\}$ edge ($\{w', v\}$ is clearly not an edge since then v would be also connected to w or w could have done a NeighborSwitch).

Case 5. Add a new edge between two nodes (v, w) w/out free stubs, where neither can do a NeighborSwitch with v' and w' respectively. We break this case into two subcases, based on whether nodes v', w' (with free stubs) form a non-chord.

Case 5a. v', w' is not a non-chord. This means that we can add a new edge between v', w'. It is easy to see that v', w' edge is not already present, because otherwise v and w could have performed a NeighborSwitch.

Case 5b. v', w' is a non-chord. This case is not possible when v, w are not able to perform NeighborSwitches at the same time. Without loss of generality, let's say that v connects to every neighbor of v' and w'. This means that no NeighborSwitch is available for v. Now, if we want to construct w such that it can't perform a NeighborSwitch with w', w would connect to every neighbor of w'; however, this would include v too, and clearly that edge doesn't exist. Contradiction.

This concludes our proof and shows that the algorithm will terminate and generate a bipartite graph after adding $|E|$ edges without using non-chords. □

Running Time. Since the algorithm is essentially the same as before, the running time is $O(|E| \cdot d_{max})$. The only difference is when a NeighborSwitch fails to free up stubs, we use a node with free stubs. However, this takes only a constant operation when compared to `2K_Simple`. The final directed graph can be constructed from the bipartite representation by collapsing nodes with non-chords and assigning directions to edges appropriately, this takes $O(|V| + |E|)$ time.

Space Complexity. The `D2K_Simple` algorithm requires an additional $O(|V|)$ space compared to `2K_Simple` to store non-chords. However, the overall space complexity remains unchanged: $O(|V| + |E|)$.

3) Space of realizations: The algorithm for the directed case has the same properties for generating any realization as `2K_Simple`. In this section we focus on double-edge swaps for MCMC-based sampling.

D2K is a special case of an undirected JDAM, and thus inherits the property that JDAM realizations are connected via 2K-preserving double-edge swaps [9], [18] if non-chords are allowed (equivalently, self-loops in directed graphs). However, we cannot use the known swaps to sample from the space of simple directed graphs, for more details see Appendix E-A.

C. D2K+: Additional Properties in Directed Graphs

We show three examples that capture more information of graphs by imposing more restrictions on the realizations. The first approach, D2.1K, fixes average in (and out) degree neighborhoods in directed graphs; the second approach, D2K+M, is a simple heuristic to achieve high number of mutual edges in realizations, the third approach considers Balanced Degree Invariant D2K realizations. Other properties can be considered as well, similarly to the undirected graph construction.

D2.1K: correlation between (in, out) degree pairs. Instead of working with the bipartite representation, we can work directly with the directed graph, as in Fig. 9-top right. We partition nodes by both their in and out degrees (d_v^{in}, d_v^{out}), and we can define the joint degree matrix to capture the number of edges $JDM((k^{in}, l^{out}), (m^{in}, n^{out}))$, between nodes with (k^{in}, l^{out}) and (m^{in}, n^{out}) degrees.

D2.1K is a natural extension of the undirected 2K and captures a more restrictive notion of degree correlation than our main D2K definition. We use the notation D2.1K, since it already contains the information for a corresponding D2K. D2.1K fixes the average degree neighborhoods, since for a given node, v, (with known in degree) D2.1K describes the in degrees of nodes that connect to v (similarly to out degrees as well). This is not specified in D2K, since it doesn't consider the in and out degree at the same time. However, we can also observe that D2.1K can be transformed into a D2K instance with additional attributes. D2K with additional attributes (D2K+A) is the same kind of generalization used to get from JDM to JDAM, and our results from D2K carry over to D2K+A. If we use the additional attribute to capture the nodes' in and out degree, then the resulting D2K+A instance is equivalent to D2.1K. Therefore a simple extension of `D2K_Simple` can solve D2.1K instances.

D2K+M: Number of Mutual Edges. This work was motivated by the observation that, in sparse graphs, D2K produced an order of magnitude less mutual (reciprocated) edges than in the original social networks. We use a heuristic approach to target number of mutual edges in a directed graph during construction, by greedily adding mutual edges when permitted by degree and JDAM constraints. In `D2K_Simple` line 12-13, we can check if the non-chord pairs of v, w can form an edge and add it if possible. We denote this approach as D2K+M or D2.1K+M following the notation from UMAN where "M" represents the number of mutual edges in a graph. This heuristic works well in practice as shown in Section IV-D, but exact solutions might be difficult to achieve.

D2K+BDI: Balanced Realizations. Using our observation from Section III-D, we can find a swap sequence from any D2K realization to a balanced realization for D2K graphs. Since there will be two nodes to pick from at every double-edge swap when applying Lemma 4 from [9], it is possible

TABLE III
INPUT GRAPHS FROM SNAP [37]

Name	#Nodes	#Edges	Generation (sec)
p2p-Gnutella08	6,301	20,777	0.474
Wiki-Vote	7,115	103,689	1.894
AS-Caida	26,475	57,582	2.066
Twitter	81,306	1,768,135	44.884

to avoid self-loops while transforming a D2K realization to a D2K+BDI. Since every node has one non-chord assigned, we can simply pick a node for the double-edge swap that does not from a non-chord. Details are provided in Appendix F.

D. Simulations for Real-World Directed Graphs

We have the same simulation setup as in Section III-E. We compare realizations generated by Directed ER (D0K), UMAN, Directed Degree Sequence (D1K), Directed 2K, Directed 2K+M, Directed 2.1K, Directed 2.1K+M with the corresponding target properties captured on input graph (G). Since we are the first to introduce D2K, we focus on how well D2K targets various graph properties, rather on evaluating the algorithm efficiency.

We used examples of directed graphs from SNAP [37]: p2p-Gnutella08, Wiki-Vote, AS-Caida (without customer relations), Twitter. Table III provides an overview of the graphs (without self-loops and multi-edges) used in our experiments and reports the average time to construct realizations using D2K for these examples. In the rest of this section, we consider several well known graph properties (described in Appendix E-B) also used by Orsini et. al. [6] to study the convergence of dK-series for undirected networks and we report those that are more natural for directed graphs, such as the triad census. We average results over 20 realizations for every construction method and specific property.

Results. The size of these graphs matches the inputs by definition. We can also observe in Fig. 11 that Directed Degree Distributions and Degree Correlations are captured by D2K, D2.1K as expected by definition. On the other hand, D0K, D1K and UMAN capture Degree Correlations poorly, thus D2K graphs have a chance to capture other properties more accurately than D0K or D1K.

Dyad Census is not well captured for Twitter, as we can see in Fig. 11. However, there are order of magnitude improvements in the number of mutual edges between D2.1K (123,040.4) and D2K (3,628.7), D1K (2,155.95) or D0K (233.05). Of course, UMAN preserves this property by definition. D2K+M and D2.1K +M does not meet the target exactly since the current implementation is a heuristic, but it significantly boosts the number of mutual edges.

Triad Census is surprisingly well captured by UMAN, the reason being the exact match for the Dyad Census in the previous point. On the other hand, a convergence can be seen between dK-series generators with significant improvements in dense triadic structures from D1K to D2K and from D2K to D2.1K. Targeting the mutual edges helps D2K and D2.1K in the dense triadic structures like “201”, “210” or “300”.

Dyad-wise Shared Partners follow similar trends to other properties, such that D2.1K is significantly more accurate than D2K. D2K improves over D1K in terms of ”outgoing shared partners” but that improvement decreases at ”independent two-paths” and disappears at ”incoming shared partners”.

Expansion is again best approximated by D2.1K and D2.1K even matches *Average Neighbor Degree* exactly if marginalized by degrees as in Fig. 11. D2K also follows the general shape of these distributions but includes larger error, while D1K has systematic difference compared to G.

Betweenness Centrality CDF has no significant improvements after matching degree distributions with D1K in Twitter; other examples reached target closer with D1K. Interestingly UMAN performs almost identically to D0K, even though the number of mutual edges is significantly different.

Shortest Path Distribution has slow convergence to target across different methods, but the average shortest path is shorter than the observed in G.

K-Core Distribution is best captured by D2.1K, and there is a small improvement from D1K to D2K using Twitter. However, the dense core using D1K or D2K is almost an order of magnitude lower core index. Targeting mutual edges for D2K helps in reconstructing better structure in terms of coreness, and gets the results closer to D2.1K.

Eigenvalues of Twitter is again best targeted by D2.1K. There is a difference between leading eigenvalues in graph realizations of the other methods but starting at the second eigenvalue the difference between D1K and D2K quickly decreases. D2K+M shows significantly lower error than D2K.

The Twitter network showcased most of our general findings using the directed dK-series. Due to lack of space we defer the remaining datasets to Appendix E-C.

V. CONCLUSION

Our 2K+ framework advances the state-of-the-art in modeling and simulation of complex networks, especially in the context of online social networks that exhibit high clustering and are affected by node attributes. It provides an efficient way to construct simple, directed and undirected graphs, that exhibit exactly a target degree correlations and potentially additional properties, including: clustering, number of connected components, node attributes for undirected and average neighbor degree, number of mutual edges or balanced realizations for directed graphs, etc. Key strengths of this work include: (1) a principled approach to graph synthesis, with exact guarantees when possible (2K, 2K+A, 2K+CC, D2K) and efficient heuristics when justified (*e.g.,* the 3K problem is NP-hard motivating 2.5K and 2.25K heuristics); (2) extensibility to target additional properties by exploiting the insights we developed, namely the under-defined nature of the 2K algorithm (*e.g.,* order of adding edges), manipulating attributes in JDAM, speeding up MCMC, and connections between all these related problems; (3) efficiency: the time for constructing large graphs reduced from weeks and days to minutes and seconds. We have also contributed our implementations to the Python NetworkX library, both for undirected [34] and for directed [39] graphs.

REFERENCES

[1] P. Erdős and T. Gallai, “Gráfok előírt fokú pontokkal,” *Mat. Lapok*, vol. 11, pp. 264–274, 1960.

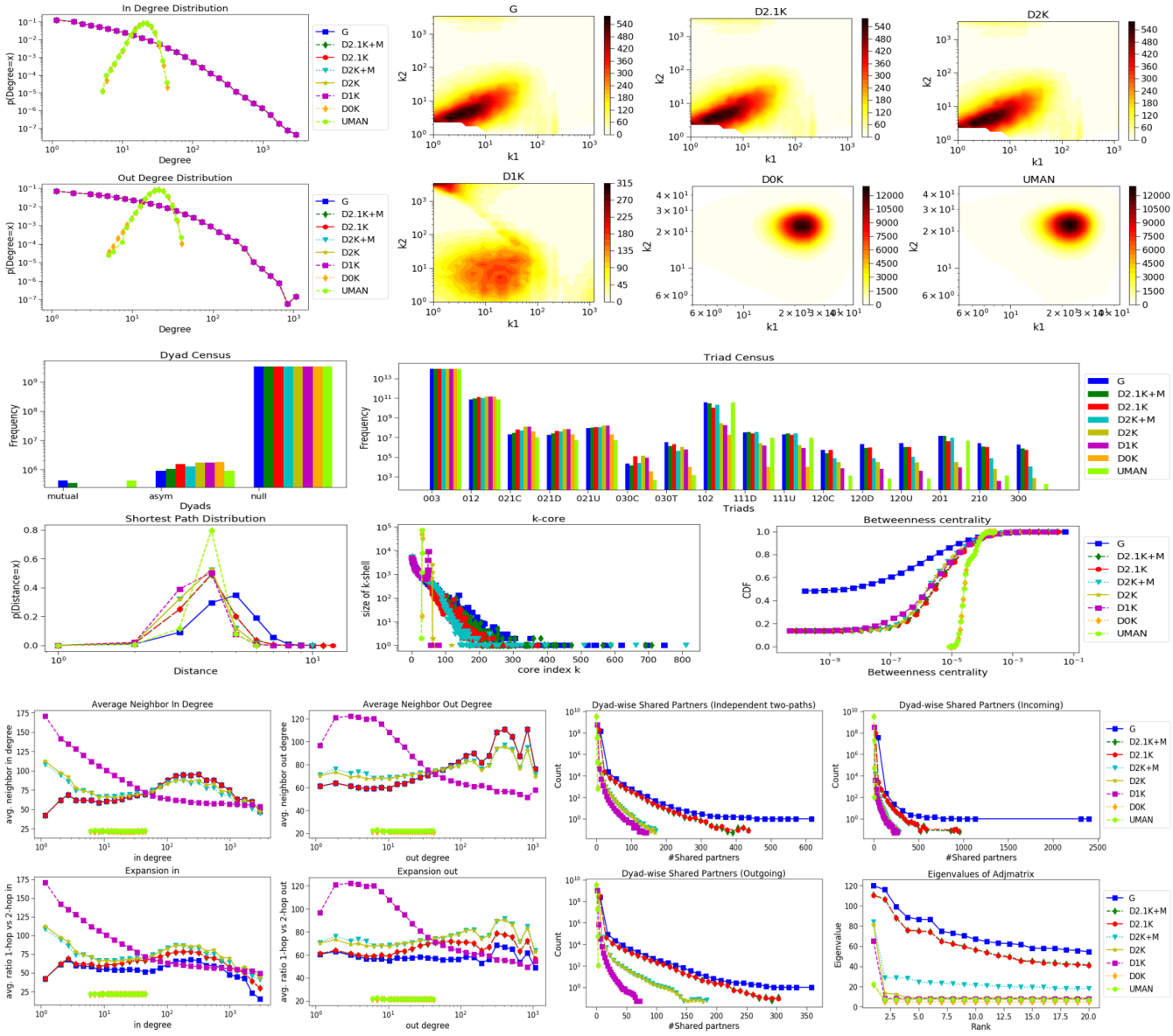


Fig. 11. Results for Twitter graph: Directed Degree Distribution, Degree Correlation, Dyad-, Triad Census [the order of bars (left-to-right) is the same is in the legend (top-to-bottom)], Shortest Path Distribution, K-core distribution, Betweenness Centrality, Expansion, Average Neighbor Degree, DSP and top 20 Eigenvalues

[2] V. Havel, "Poznámka o existenci konecnych grafu," *Časopis pro pěstování matematiky*, vol. 80, no. 4, pp. 477–480, 1955.

[3] S. L. Hakimi, "On realizability of a set of integers as degrees of the vertices of a linear graph. i," *Journal of the Society for Industrial and Applied Mathematics*, vol. 10, no. 3, pp. 496–506, 1962.

[4] R. Taylor, *Constrained switchings in graphs*. University of Melbourne, Department of Mathematics, 1980.

[5] P. Mahadevan, D. Krioukov, K. Fall, and A. Vahdat, "Systematic topology analysis and generation using degree correlations," in *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4. ACM, 2006, pp. 135–146.

[6] C. Orsini, M. M. Dankulov, P. Colomer-de Simón, A. Jamakovic, P. Mahadevan, A. Vahdat, K. E. Bassler, Z. Toroczkai, M. Boguñá, G. Caldarelli *et al.*, "Quantifying randomness in real networks," *Nature communications*, vol. 6, 2015.

[7] P. L. Erdős, S. G. Hartke, L. van Iersel, and I. Miklós, "Graph realizations constrained by skeleton graphs," *arXiv preprint arXiv:1508.00542*, 2015.

[8] Y. Amanatidis and B. Green and M. Mihail., "Graphic realizations of joint-degree matrices," *Unpublished manuscript*, 2008.

[9] É. Czabarka, A. Dutle, P. L. Erdős, and I. Miklós, "On realizations of a joint degree matrix," *Discrete Applied Mathematics*, vol. 181, pp. 283–288, 2015.

[10] M. Gjoka, B. Tillman, and A. Markopoulou, "Construction of simple graphs with a target joint degree matrix and beyond," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 1553–1561.

[11] W. Devanny, D. Eppstein, and B. Tillman, "The computational hardness of dk-series," in *NetSci 2016*, 2016.

[12] D. Gale *et al.*, "A theorem on flows in networks," *Pacific J. Math*, vol. 7, no. 2, pp. 1073–1082, 1957.

[13] D. R. Fulkerson *et al.*, "Zero-one matrices with zero trace," *Pacific J. Math*, vol. 10, no. 3, pp. 831–836, 1960.

[14] S. Dorogovtsev, "Networks with desired correlations," *arXiv preprint cond-mat/0308336*, 2003.

[15] W. Aiello, F. Chung, and L. Lu, "A random graph model for massive graphs," in *Proceedings of the thirty-second annual ACM symposium on Theory of computing*. Acm, 2000, pp. 171–180.

[16] J. Blitzstein and P. Diaconis, "A sequential importance sampling algorithm for generating random graphs with prescribed degrees," *Internet Mathematics*, vol. 6, no. 4, pp. 489–522, 2011.

[17] C. I. Del Genio, H. Kim, Z. Toroczkai, and K. E. Bassler, "Efficient and

exact sampling of simple graphs with given arbitrary degree sequence," *PloS one*, vol. 5, no. 4, p. e10012, 2010.
[18] G. Amanatidis, B. Green, and M. Mihail, "Graphic realizations of joint-degree matrices," *arXiv preprint arXiv:1509.07076*, 2015.
[19] I. Stanton and A. Pinar, "Constructing and sampling graphs with a prescribed joint degree distribution," *Journal of Experimental Algorithmics (JEA)*, vol. 17, pp. 3–5, 2012.
[20] K. E. Bassler, C. I. Del Genio, P. L. Erdős, I. Miklós, and Z. Toroczkai, "Exact sampling of graphs with prescribed degree correlations," *New Journal of Physics*, vol. 17, no. 8, p. 083052, 2015.
[21] P. L. Erdős, I. Miklós, and Z. Toroczkai, "New classes of degree sequences with fast mixing swap markov chain sampling," *arXiv preprint arXiv:1601.08224*, 2016.
[22] P. L. Erdos, I. Miklós, and Z. Toroczkai, "A decomposition based proof for fast mixing of a markov chain over balanced realizations of a joint degree matrix," *SIAM Journal on Discrete Mathematics*, vol. 29, no. 1, pp. 481–499, 2015.
[23] M. Gjoka, M. Kurant, and A. Markopoulou, "2.5 k-graphs: from sampling to generation," in *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2013, pp. 1968–1976.
[24] X. Dimitropoulos, D. Krioukov, A. Vahdat, and G. Riley, "Graph annotations in modeling complex network topologies," *ACM Trans. Model. Comput. Simul.*, vol. 19, no. 4, pp. 17:1–17:29, Nov. 2009. [Online]. Available: http://doi.acm.org/10.1145/1596519.1596522
[25] P. W. Holland and S. Leinhardt, "Local structure in social networks," *Sociological methodology*, vol. 7, pp. 1–45, 1976.
[26] H. Kim, C. I. Del Genio, K. E. Bassler, and Z. Toroczkai, "Constructing and sampling directed graphs with given degree sequences," *New Journal of Physics*, vol. 14, no. 2, p. 023012, 2012.
[27] M. Gjoka, B. Tillman, A. Markopoulou, and R. Pagh, "Efficient construction of 2k+ graphs," in *NetSci 2014*, 2014.
[28] B. Tillman and A. Markopoulou, "On the number of connected components of joint degree matrix realizations," in *abstract submitted to NetSci 2018*, 2018.
[29] B. Tillman, A. Markopoulou, C. T. Butts, and M. Gjoka, "Construction of directed 2k graphs," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17. New York, NY, USA: ACM, 2017, pp. 1115–1124. [Online]. Available: http://doi.acm.org/10.1145/3097983.3098119
[30] F. Viger and M. Latapy, "Efficient and simple generation of random simple connected graphs with prescribed degree sequence," in *International Computing and Combinatorics Conference*. Springer, 2005, pp. 440–449.
[31] D. R. Hunter, M. Handcock, C. Butts, S. M. Goodreau, and M. Morris, "ergm: A package to fit, simulate and diagnose exponential-family models for networks," *Journal of Statistical Software*, vol. 24, no. 3, 2008.
[32] J. J. Pfeiffer III, S. Moreno, T. La Fond, J. Neville, and B. Gallagher, "Attributed graph models: modeling network structure with correlated attributes," in *Proc. of WWW*, 2014.
[33] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in Science Conference (SciPy2008)*, Pasadena, CA USA, Aug. 2008, pp. 11–15.
[34] M. Gjoka, "2k simple implementation in networkx," https://networkx.github.io/documentation/latest/reference/generated/networkx.generators.joint_degree_seq.joint_degree_graph.html, 2016.
[35] A. Traud, P. Mucha, and M. Porter, "Social Structure of Facebook Networks," *Arxiv preprint arXiv:1102.2166*, 2011.
[36] B. Viswanath, A. Mislove, M. Cha, and K. Gummadi, "On the evolution of user interaction in facebook," in *Proc. WOSN*, 2009.
[37] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.
[38] A. Mislove, M. Marcon, K. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *IMC*, 2007.
[39] B. Tillman, "D2k simple implementation in networkx (in progress)."
[40] N. Developers, "Networkx," *networkx. lanl. gov*, 2010.
[41] "Simple Graph: http://mathworld.wolfram.com/SimpleGraph.html."
[42] P. Holme and B. J. Kim, "Growing scale-free networks with tunable clustering," *Physical review E*, vol. 65, no. 2, p. 026107, 2002.
[43] P. L. Erdős, Z. Király, and I. Miklós, "On the swap-distances of different realizations of a graphical degree sequence," *Combinatorics, Probability and Computing*, vol. 22, no. 3, pp. 366–383, 2013.
[44] T. A. B. Snijders, P. E. Pattison, G. L. Robins, and M. S. Handcock, "New specifications for exponential random graph models," *Sociological Methodology*, vol. 36, pp. 99–153, 2006.

Bálint Tillman received the B.Sc degree in Business Information Technology from the Corvinus University of Budapest, Hungary, in 2008, the M.Sc degree in Software Development and Technology from IT University of Copenhagen, Denmark, in 2012. He was an Intern with Google in 2016, 2017, and 2018. He is currently pursuing a Ph.D degree in Networked Systems Program from the University of California at Irvine since 2014.

He received the Henry Samueli Fellowship for Networked Systems 2015-2016.

His research interests include graph algorithms and machine learning in computer networks.

Minas Gjoka received his B.S. (2005) degree in Computer Science at the Athens University of Economics and Business, Greece, and his M.S. (2008) and Ph.D. (2010) degrees in Networked Systems at the University of California, Irvine. He is currently with Google, Santa Monica. His research interests are in the general areas of networking and distributed systems, with emphasis on graph construction algorithms, network security, network measurements, sampling and analysis of massive online graphs, and the design of networked systems.

Carter T. Butts is a Professor in the departments of Sociology, Statistics, and Electrical Engineering and Computer Science (EECS) and the Institute for Mathematical Behavioral Sciences at the University of California, Irvine. His research involves the application of mathematical and computational techniques to theoretical and methodological problems within the areas of social network analysis, mathematical sociology, quantitative methodology, and human judgment and decision making. His work has appeared in a range of journals, including Science, Sociological Methodology, the Journal of Mathematical Sociology, Social Networks, and Computational and Mathematical Organization Theory.

Athina Markopoulou (S'98-M'02-SM'13) is currently an Associate Professor in the EECS Department, at the University of California, Irvine. She has held short term/visiting positions at SprintLabs (2003), Arista Networks (2005), and IT University of Copenhagen (2013). She received the Diploma degree in Electrical and Computer Engineering from the National Technical University of Athens, Greece, in 1996, and the M.S. and Ph.D. degrees in Electrical Engineering from Stanford University in 1998 and 2003, respectively.

She received the OCEC Educator Award (2017), the Henry Samueli School of Engineering Faculty Midcareer Award for Research (2014) and the NSF CAREER Award (2008). She has served as an Associate Editor for IEEE/ACM Transactions on Networking (20132015), an Associate Editor for ACM CCR, a General CoChair for ACM CoNext 2016, a Local Arrangements Chair for ACM SIGMETRICS 2018, and a TPC Co-Chair for NetCod 2012.

Her research interests are in the general area of networking, including mobile and social networks, network measurement, network security and privacy, and network coding.

APPENDICES TO IEEE/ACM TRANSACTIONS ON NETWORKING PAPER: "2K+ GRAPH CONSTRUCTION FRAMEWORK" BY TILLMAN ET AL.

APPENDIX A
2K+A ALGORITHM

Section III-C describes the algorithm for targeting a given JDAM and the details are provided below.

Algorithm 3: `2K_Simple_Attributes`

Input: $JDAM^{\odot}$
Init: $JDAM(\{k,i\},\{l,j\}) = 0,\ \forall\ (\{k,i\},\{l,j\}) \in JDAM^{\odot}$
1: **for** $(\{k,i\},\{l,j\}) \in JDAM^{\odot}(\{k,i\},\{l,j\})$
2: **while** $JDAM(\{k,i\},\{l,j\}) < JDAM^{\odot}(\{k,i\},\{l,j\})$
3: Pick any nodes $v \in V_{\{k,i\}}$ and $w \in V_{\{l,j\}}$
s.t. (v,w) is not an existing edge
4: **if** v does not have free stubs:
5: v': node in $V_{\{k,i\}}$ with free stubs
6: NeighborSwitch(v,v')
7: **if** w does not have free stubs:
8: w': node in $V_{\{l,j\}}$ with free stubs
9: NeighborSwitch(w,w')
10: add edge between (v, w)
11: $JDAM(\{k,i\},\{l,j\})$++ ; $JDAM(\{l,j\},\{k,i\})$++
Output: simple graph with $JDAM = JDAM^{\odot}$

APPENDIX B
SPACE OF CONSTRUCTED UNDIRECTED GRAPHS

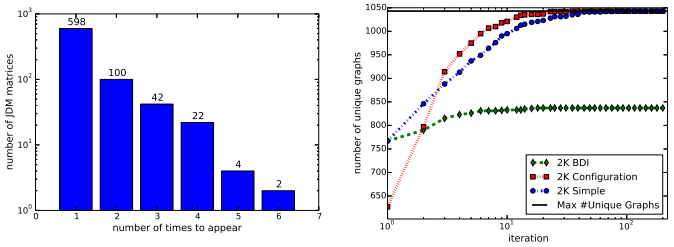
We evaluate the space of graphs that our algorithms can construct through simulations. We compare against two main baselines for comparison: `2K_BDI` and `2K_Configuration` (for the latter, after throwing away self-loops and multi-edges). We show that `2K_BDI` can produce significantly less graphs than `2K_Simple`, due to the BDI constraints.

A. All Seven Node Graphs

We experimentally show that `2K_Simple` can construct *all* possible graphs with up to seven nodes, while `2K_BDI` can produce much less.

We use the library NetworkX [40] to generate all 1044 non-isomorphic graph instances that contain seven nodes. We should note that the number of such graph instances increases exponentially with size e.g. for n=24 it is $\sim 1.95 \times 10^{59}$ [41]. For this reason we use in our experiment a small size of n=7 that gives 1044 instances. For each graph instance we calculate the corresponding JDM, which results in 768 unique JDM matrices (because there are cases where several graph instances correspond to the same JDM matrix). Fig. 12(a) shows the frequency of JDM matrices. We see that 598 matrices appear only once. Therefore, if a generator received as input one of those JDM matrices it would always produce the same graph. On other extreme, two JDM matrices appear 6 times each. Therefore, if a generator received as input one of those JDM matrices it could produce either of the 6 corresponding graphs.

We conduct the following experiment. In each iteration, we feed the three algorithms with all 768 JDM matrices and we observe how many unique matrices each algorithm has generated, cumulatively since the first iteration. Fig. 12(b) shows the results. We observe that both simple graph construction algorithms (`2K_Simple` and `2K_BDI`) generate 768 unique graphs in the first iteration; `2K_Configuration` generates less graph instances due to multi-edges, which we removed. As the number of iterations increases, we observe that both `2K_Simple` and `2K_Configuration` reach 1044 (*i.e.,* the total number of unique graphs corresponding to the 768 unique JDMs) in less than 100 iterations. However, the `2K_BDI` algorithm is unable to create more than 837/1044 (=80%) unique graphs after 200 iterations, due to the BDI constraint, discussed in Section II.



(a) Frequency of JDM matrices.

(b) Cumulative number of unique graphs with seven nodes.

Fig. 12. Generation of all non-isomorphic graph instances with 7 nodes.

B. Scale-free Graphs with Clustering

We now look at several graph properties beyond the targeted JDM. We show that our 2K algorithm generates graphs that have a quite large range of these other properties, much larger than `2K_Configuration` and `2K_BDI`. Put differently, `2K+S` explores the space of graphs much faster than existing 2K construction algorithms.

We select the Holme and Kim algorithm [42], implemented in NetworkX [40] by function powerlaw_cluster, to generate graphs with a powerlaw degree distribution and fine-tuned clustering. The algorithm requires three parameters: the number of nodes n, the number of random edges m to add for each new node, and the probability p of adding a triangle after adding a random edge. We set $n = 100$ and $m = 2$ and we vary p between $0 - 1$ with a step size of 0.01. As a result, we produce 100 graphs from this model with exactly 100 nodes, 196 edges, and a varying amount of average clustering $\bar{c}$. For each of the 100 graphs, we calculate the JDM and produce 10^5 graph instances using the algorithms `2K+S`, `2K_BDI`, and `2K_Configuration`. We produce a total of $3 \cdot 100 \cdot 10^5 = 3 \cdot 10^7$ graphs. For each graph, we compute the following properties: (i) the average clustering coefficient $\bar{c}$, (ii) the average shortest path length over all node pairs, (iii) the average node closeness centrality; the closeness centrality of a node v is defined as the inverse sum of distances of v to all other nodes and measures the speed of information

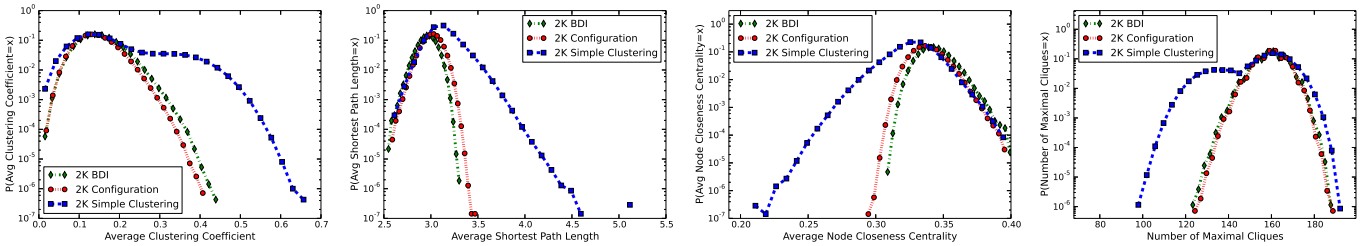


Fig. 13. We use the Holme-Kim algorithm [42] to produce 100 graphs, each with exactly 100 nodes, 196 edges and a range of $\bar{c}$ values. We then calculate the JDM matrix for each produced graph instance and set it as the JDM$^{\odot}$ for each 2K construction algorithm. Overall we generate 10^7 graph instances per construction algorithm and empirically calculate the pdf for each graph property.

spreading from v, and (iv) the total number of maximal cliques $\sum C_i$, where C_i is the number of maximal cliques of order i.

We should note that in this experiment we chose to generate small graphs with fixed number of nodes and edges for practical reasons. First, it is computationally complex to generate 30 million graph instances and their corresponding properties, and the small size of the produced graphs facilitates that process. Second, we fixed the number of edges so as to constrain the variability of the considered properties over all possible graphs with given input JDMs. That makes the overall space of graphs smaller and thus easier to sample. Nevertheless, the number of all possible graphs is still enormous to exhaustively enumerate. Thus we provide a relative comparison of the construction algorithms in regard to the achieved range of given properties.

Figure 13 shows the empirical probability density for each of the considered properties. `2K+S` covers a significantly larger range for all properties when compared to `2K_Configuration` and `2K_BDI`. *e.g.,* `2K+S` produces graphs with $\bar{c}$ that ranges in $[0, 0.67]$ whereas the range of values for `2K_Configuration` is $[0, 0.41]$, and for `2K_BDI` $[0, 0.44]$.

Appendix C
Targeting JDM and Clustering

Following up on Section III-B of the main paper, we provide details on 2K+S paramaters and implementation.

Tuning Sortedness parameter. Fig. 14 shows an example of the correlation between the sortedness parameter, S, and average clustering coefficient for three types of graph inputs: two graph models and one Facebook network are used to generate the graphs. Fig. 14 shows that, for a fixed $JDM^{\odot}$, by setting parameter S between 0 and 1, we can also control the average clustering coefficient $\bar{c}^{\odot}$ of the produced graph between $min(\bar{c})$ and $max(\bar{c})$. We use function $y = sin(\frac{x \cdot \pi}{2 \cdot max(\bar{c})})$ to approximate the observed relation between parameter S and $\bar{c}$. Therefore, setting parameter $S = s$ roughly corresponds to an average graph instance with average clustering coefficient equal to $\frac{2 \cdot max(\bar{c}) \cdot arcsin(s)}{\pi}$.

Running Time. The time complexity of `2K+S` is similar to `2K_Simple` for adding edges (*i.e.,* $O(|E| \cdot d_{max})$), plus the time for the function order($List$, S). If naively implemented, the time to sort the input list E' is $O(|E'|log(|E'|))$. However, the list E' is consumed by lines 4-7 in Algorithm 2), which require at most $|E|$ edges that pass the condition in Line 5. Therefore, we argue that we do not need to enumerate all elements of E' because only some of the node pairs in E' will not be rejected by the condition in Line 5. In practice, we observed that enumerating the first $k|E|$ node pairs of list E', where k is some small number, suffices to add the overwhelming majority of edges in the graph. The small number of remaining edges (if any) will be taken care of by Stage 2. Furthermore, we use the coordinate system r_v to sort nodes in each degree group k which takes time $O(\sum_k D_k log(D_k))$. After this initial sorting phase, each node v can find its closest k neighbors in linear time. In summary, the running time of a smart implementation of the function order($List$, S) that returns $k|E|$ elements is $O(k|E| + \sum_k D_k log(D_k))$. The expression is dominated by the term $k|E|$ in real-world graphs, which makes the running time approximately linear in the number of edges.

Appendix D
2K+MinCC Proofs

This appendix provides the details for Lemma 5 and 6 from Section III-D of the main paper.

A. Any Realization to MinCC Swap Sequence, Extended Proof

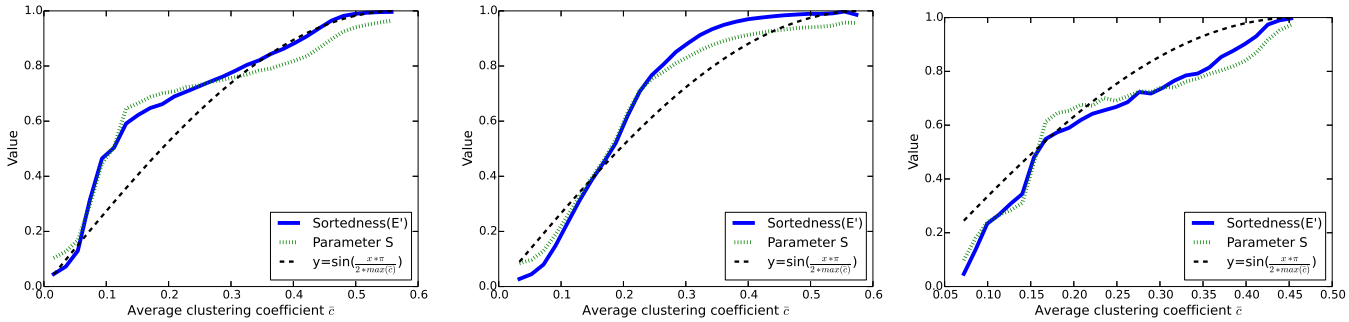
First, we show how to construct a realization with minimum number of connected components from any JDM realization using JDM-preserving double-edge swaps (while not increasing the number of connected components at any step). The main algorithm and the notation follows the Valid Tree Construction algorithm described by Amanatidis *et al.* in [18], but the key difference is that the modified algorithm starts from a JDM realization (instead of a $V-1$ "valid" edges) and only uses JDM-preserving double-edge swaps to construct a realization with minimum number of connected components. This change will require some adjustments in the algorithm and in the certificate that is produced to show that there is no realization with less than c number of connected components.

We use the following notation:

- V_i is a degree group (nodes with degree i) $\mathsf{V} = \cup V_i$.
- $F \subset \mathsf{V}$, and A is a partition of F.

Define a weighted graph $G^{cert.}(V', E', w)$ with a node for each element A_i in A and one node for each $V_x \notin F$, assign the following edges and weights:

- If $\exists V_x \in A_i$ and $\exists V_y \in A_j$ such that $JDM(x, y) > 0$, then add edge (i, j) with weight $w(i, j) = 1$.



(a) Erdős-Rényi generator, $n = 100, m = 483, p = 0.1$

(b) Powerlaw-Cluster generator, $n = 100, m = 196, p = 0.1$

(c) Facebook Caltech network, $n = 769, m = 16,656$

Fig. 14. We use two different graph models (Erdős-Rényi, Powerlaw-Cluster [42]) to generate two graph instances and we select one real-world network, Facebook Caltech [35]. For each graph instance, with n nodes and m edges, we calculate its JDM and set it as our target. We then generate 10^5 graphs with sortedness and parameter S values varied between [0,1], and record the average clustering coefficient $\bar{c}$ of each generated graph.

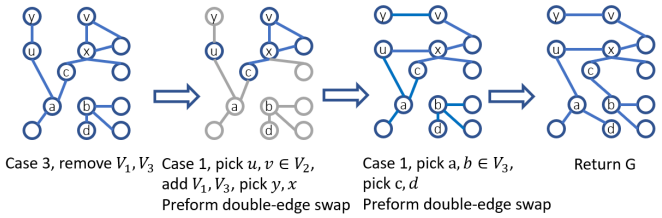


Fig. 15. Example execution of Find-MinCC algorithm.

- If $V_x, V_y \notin F$, then add edge (x, y) with weight $w(x, y) = JDM(x, y)$.
- If $V_x \notin F$ and $JDM(x, x) > 0$ add self-loop with weight $w(x, x) = JDM(x, x)$.
- For any A_i, V_x, add an edge with weight $w(i, x) = \sum_{z:V_z \in A} JDM(z, x)$, if $w(i, x) > 0$.

Given a realization $G(V, E)$ of a JDM with minimum number of connected components ($c \geq 1$), we can construct $G^{cert.}(V', E', w)$ for every F, A combination, by collapsing nodes corresponding to A and V_i, then the following inequality holds $\sum_{e \in E'} w(e) \geq |A| + \sum_{V_i \notin F} |V_i| - c$ in the constructed weighted graph, that has at most c components. This follows from the existence of the realization. If there is partition F and c, where the above inequality doesn't hold, it is a certificate to show that there is no realization with c connected components.

We refer to the modified algorithm from [18] as Find-MinCC here. Find-MinCC finds a swap sequence instead of building a partial realization of a spanning tree for input JDM. Fig. 15 shows a simple example to highlight the intuition behind how the algorithm uses double-edge swaps to move and break cycles to connect different connected components in a JDM realization.

Theorem 7. *The Find-MinCC algorithm finds a realization with the minimum number of connected components of JDM in polynomial time.*

Proof. It is trivial to show, that if G_0 is a forest or a single connected component realization, then the algorithm found a minimum number of connected component realization. Consider the three cases of Algorithm Find-MinCC.

Find-MinCC(G)
begin
V= $\cup_{i=1}^{k} V_i$; j=0; G=G_0
while G_0 is not connected or not a forest:
begin
$O_j = \{v$: nodes on cycles in $G_j\}$
$C_j = \{V_i : O_j \cap V_i \neq \emptyset\}$
$P_j = \{V_i : V_i$ intersects at least two connected components of $G_j\}$
$Z_j = \{e \in G_j$: at least one endpoint of e is in some $V_i \in P_j\}$
Case 1: If $C_j \cap P_j \neq \emptyset$
pick u, v in some $V_i \in C_j \cap P_j$ from different components in G and $u \in O_j \cap V_j$;
j = max(j-1,0); $G_j = G \cup P_j \cup Z_j$; $G = G_j$;
pick x: neighbor of u from a cycle in G;
pick y: neighbor of v in G;
remove xu, yv from G; add xv, yu to G;
Case 2: else if $P_j = \emptyset$
let $K_1, K_2, ..., K_\lambda$ be components of G_j
let $A_i = V_x : V_x \subset V(K_i)$ for $1 \leq i \leq \lambda$;
let $F = \cup_{i=1}^{\lambda} A_i$; let $A = \{A_i, ..., A_\lambda\}$;
output (F, A); $G \cup_{i=0}^{j} P_i \cup_{i=0}^{j} Z_i$; terminate
Case 3: else $G_{j+1} = G_j \setminus P_j; G = G_{j+1}; j = j + 1$
end
output G_0;
end

First, we show that the algorithm terminates after polynomial many iterations: The recursion can only go to depth k - the number of distinct degrees - using **Case 3**, because at that point Case 2 will happen and the algorithm terminates. On the other hand, we will shortly show, that when Case 1 happens at depth j, then Case 1 will happen j consecutive times and the number of connected components will be decreased by one in G_0. This means that the while-loop can iterate at most $O(k|V|)$, using $2k$ iterations to merge pairs of connected components (loosely upper bounded by $|V|$).

Similar to [18], we notice that if G_0 is not a realization

with minimum number of connected components, then for any j such that a G_j is constructed by the algorithm we have $O_j \neq \emptyset$ (and thus $C_j \neq \emptyset$). G_0 either is a forest or contains cycles. Also any G_j if created will have $O_{j-1} \in V(G_j)$, because otherwise Case 1 would happen. Intuitively cycles are preserved as j increases.

Case 1. First, notice that Case 1 is exactly a JDM-preserving double-edge swap, thus any realization after performing these swaps will be a realization of the same JDM as the input graph. The argument is similar to [18]: the number of connected components will decrease by 1 in G_0, if Case 1 happens at G_j. Notice that two components that got connected of $G = G_j$, K, K', must be subgraphs of the same connected components in G_{j-1} otherwise they would have been connected in an earlier iteration (*i.e.,* when G was still G_{j-1}). After the double-edge swap, we call G'_{j-1} the graph that got back P_{j-1}, Z_{j-1} (*i.e.,* the nodes and edges previously removed in at $j-1$ depth). The key observation is that, in G'_{j-1}, a new cycle is created for some v from $V_i \in P_{j-1}$, because v was on a path connecting K, K' in G_{j-1} before. This will result in another Case 1 in G'_{j-1} until j reaches 0 and results in a decrease of number of connected components by one in G_0. Notice that differently from [18], we first add back the P_{j-1}, Z_{j-1} and then perform the double-edge swap, in order to ensure that v has neighbors to perform double-edge swaps with at G'_{j-1}.

Case 2. The analysis of Case 2 is the same as Case 3 in [18], except we already know that all the required edges are present. Let K_1, K_2, K_λ be the connected components of G_j and let A and F be defined as in the algorithm. This assignment makes sense for G_j: if a node $v \in V_x$ is in K_i then all nodes of V_x are also in K_i (since $P_j = \emptyset$). If we consider the current graph, G_0, all cycles in G_0 are contained in the subgraph of G_0 induced by nodes of $\cup_i K_i$.

Following the proof from [18], we only have to notice that, if we identify all A_i with one single node to get H from G_0, H will have c connected components but contains no cycles. That means $|E(H)| = |V(H)| - c$, to have $c' = c - 1$ connected components it changes the above equality to $|E(H)| < |V(H)| - c'$, but we also know that $|V(H)| = |A| + \sum_{V_i \notin F} |V_i|$, and $|E(H)| = \frac{1}{2} \sum_{i:V_i \notin F} \sum_{j:V_j \notin F} JDM(i,j) + \sum_{i:V_i \notin F} \sum_{x=1}^{\lambda} \sum_{y:V_y \in A_x} JDM(i,y)$.

In conclusion, if we use (F, A) to construct the weighted graph $G^{cert.}(V', E', w)$ as before, then $\sum_{e \in E'} w(e) < |A| + \sum_{V_i \notin F} |V_i| - c'$, showing that no realization exists with less than c connected components. □ □

The running time of Find-MinCC will depend on the number of connected components of G and the minimum number of connected components achievable c. An input realization can be constructed in $O(k|E|)$ time. Each iteration in Find-MinCC can be easily done in $O(|E|+|V|)$ time using Tarjan's bridge finding algorithm to identify nodes in cycles and using sets appropriately to do operations for O, C, P, Z. Find-MinCC runs in $O(k|V||E|)$ which dominates the input construction time. However, `2K_Simple` returns with realizations using c' connected components, this results in a better overall running time of $O(k(c'-c)|E|)$ and $O(k|E|)$ running time if we assume that $c' - c$ is some small constant. The Find-MinCC algorithm can be efficiently applied for realizations that have close to minimum number of connected components for a target JDM.

Since Find-MinCC only used JDM-preserving double-edge swaps without increasing the number of connected components at any step, the union of the swaps used during the algorithm will transform the input realization to a minimum number of connected component realization.

B. MinCC to MinCC Swap Sequence, Extended Proof

In this section we show that every pair of JDM realizations with minimum number of connected components are connected over double-edge swaps. Again, this can be shown by using the proof for the space of JDM realizations from [18]. Here we have to make an additional constraint, such that during every double-edge swap the number of connected components will not increase.

The proof is based on induction on the size of the symmetric difference, k, between two JDM realizations with MinCC (G, G'). The key idea follows the results from [18], however, here we have to consider the number of connected components at every step. First we start by several simple observations and definitions about the problem.

We use a red-blue graph to describe the symmetric difference of edges where red represents edges only in G, blue represents edges only in G', black edges are present in both G, G' and edges not present in either graphs will be left empty. Red (blue) path is defined as a simple path in the red-blue graph that only contains red (blue) and black edges.

There are two simple but crucial points, shown by Amanatidis *et al.* [18] and others in earlier work: (1) the number of red and blue edges for a node is the same and (2) there always exists a red-blue circuit decomposition for the red-blue graph using red and blue edges, *i.e.,* the symmetric difference. Our proof is similar to proofs based on alternating red-blue cycles for degree sequences, but our cases also correspond to cases found in [18].

Similar to Amanatidis *et al.* [18], our proof is based on pairing nodes, that can be defined as two nodes from the same degree group on an alternating red-blue cycle of distance 2 (along the cycle). It is trivial to see that double-edge swaps around these nodes will be JDM-preserving.

There is only one case where the number of connected components would increase after performing a double-edge swap: $v - w - p - u - z$ (where p is the only (simple) path between w, u). A double-edge swap using w, z would return new paths $v - z$ and a new $w - p - u - w$ cycle, thus increasing the number of connected components. For simplicity, we will call these configurations red and blue "cut-paths" depending on whether they appear in red or blue graphs. However, degree 1 nodes cannot participate in a cycle, thus any double-edge swap using two degree one nodes will maintain the number of connected components. This observation will significantly shorten our proof since it means, that we can apply any available double-edge swap from Amanatidis *et al.* [18] to handle degree 1 nodes; for the rest of this discussion we assume that the pairing nodes have at least degree 2.

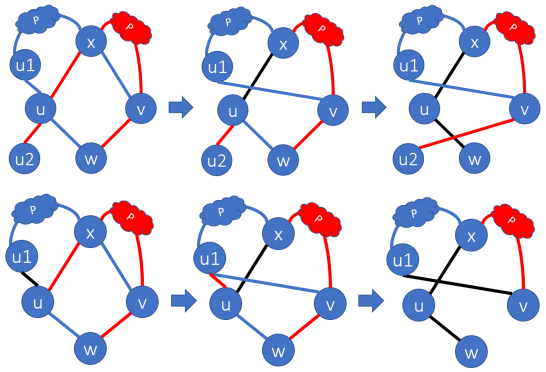


Fig. 16. Case 1, subcase 1: (u, u_1) is blue (top), (u, u_1) is black (bottom).

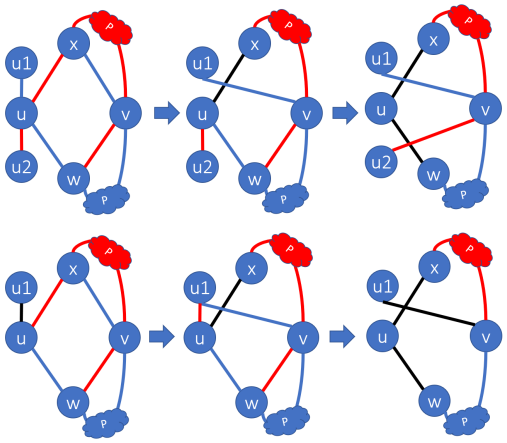


Fig. 17. Case 1, subcase 2: (u, u_1) is blue (top), (u, u_1) is black (bottom).

Double-edge swaps across different components cannot increase the number of connected components, only decrease if at least one edge was initially in a cycle. This observation also simplifies our proof, since double-edge swaps of this nature can be done without additional consideration of the connected components and solved by Amanatidis *et al.* [18].

The proof will show that starting from any symmetric difference ($k \geq 4$), we can reduce this difference by at least 2 using double-edge swaps while both red and blue graphs preserve the number of connected components. In the main part of the proof, pairing nodes have degree at least two, thus when a cut-path exists, there is going to be a usable node either on the cut-path or another neighbor which we can pick. The following cases show that in every configuration (independently whether the neighbor has red, blue or black edge), we can progress to decrease k using double-edge swaps that preserve the number of connected components.

Base case $k = 4$**:** The red-blue graph will only contain a single alternating 4-cycle, $0 < k < 4$ is not possible. Since both realizations have the same CC, both red or blue double-edge swaps along the 4-cycle will maintain the number of connected components for both the red and blue graphs.

We break our cases into 3 main groups depending on the availability of pairing nodes and the length of alternating cycles where pairing nodes exist:

Case 1: If $k > 4$ and there are pairing nodes (u,v) in a 4-cycle: we have to consider red and blue cut-paths of the type discussed earlier around these nodes. There are two configurations of red and blue cut-paths that would cause an increase in CC if naive double-edge swaps were executed. This means that u,v do not share neighbors in the same colored graph.

Subcase 1: Red and blue cut-paths meet at a non-pairing node (x): depending on whether the first edge on blue cut-path from u to u_1 is blue or black we have to cases that we can handle similarly as shown in Fig. 16.

If (u, u_1) is blue, perform blue double-edge swap $(u, u_1), (v, x) \rightarrow (v, u_1), (u, x)$, this results in (v, u_1) is blue or black depending whether (v, u_1) was red or empty before; and (u, x) becoming black from red. In addition, we can perform another red double-edge swap $(u, u_2), (v, w) \rightarrow (v, u_2), (u, w)$ that behaves the same way as the first swap.

If (u, u_1) is black, perform blue double-edge swap $(u, u_1), (v, x) \rightarrow (v, u_1), (u, x)$, this results in (v, u_1) is blue (in this case (v, u_1) has to be empty before); (u, u_1) becoming red from black; and (u, x) becoming black from red. In addition, we can perform another red double-edge swap $(u, u_1), (v, w) \rightarrow (v, u_1), (u, w)$ that removes all the red and blue edges.

Subcase 2: Red and blue cut-paths meet at a pairing node (v): since no paths cross the other pairing node u, and u has degree at least 2, there will be either two neighbors (one with red u_2 and one with blue edge u_1) or a neighbor with black edge. This case is very similar to the previous subcase 1, and swaps are shown in Fig. 17.

If (u, u_1) is blue, perform blue double-edge swap $(u, u_1), (v, x) \rightarrow (v, u_1), (u, x)$, this results in (v, u_1) is blue or black depending whether (v, u_1) was red or empty before; and (u, x) becoming black from red. In addition, we can perform another red double-edge swap $(u, u_2), (v, w) \rightarrow (v, u_2), (u, w)$ that behaves exactly the same way as the first swap.

If (u, u_1) is black, perform blue double-edge swap $(u, u_1), (v, x) \rightarrow (v, u_1), (u, x)$, this results in (v, u_1) is blue (in this case (v, u_1) has to be empty before); (u, u_1) becoming red from black; and (u, x) becoming black from red. In addition, we can perform another red double-edge swap $(u, u_1), (v, w) \rightarrow (v, u_1), (u, w)$ that removes all the red and blue edges.

In all of these subcases the double-edge swaps will decrease k by 4, because we resolve the 4-cycle without creating more red or blue edges. More importantly, these double-edge swaps will not increase the number of connected components in neither the red nor the blue graph.

Case 2: If $k > 4$ and there exists pairing nodes (u,v) on an alternating cycle with length more than 4: We are only interested in the two neighbors along the cycle for each node u,v: x,y and the shared one: w. Here we have 3 major subcases depending on whether 0,1 or 2 black edges are formed along the cycle:

Subcase 1: there are no black edges: u, v have at least one neighbor difference, depending on whether it is y or another neighbor v_1 there are different cases, as follows:

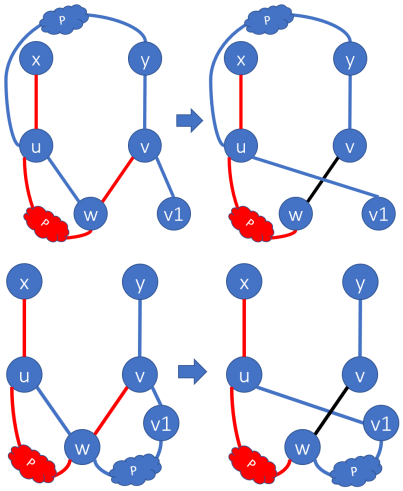


Fig. 18. Case 2, subcase 1: (v, v_1) is blue and two configurations of blue cut-paths.

If v has a blue neighbor, v_1, not on the cycle: we can perform a blue double-edge swap $(v, v_1), (u, w) \rightarrow (v, w), (u, v_1)$; this makes (v, w) black and (u, v_1) blue ((u, v_1) was not red since that would be handled by case 1 with an alternating 4-cycle). This swap preserves the number of components if the blue cut-path goes through $wu(p)yvv_1$, or $uw(p)v_1vy$ and in any other case we could have performed the blue double-edge swap along the cycle. In addition, this decreases k by 2 without change in the red graph. Two examples shown in Fig. 18.

If v has a black neighbor, v_1, and red cut-path on $xupwv_1$, blue cut-path on $wv(p)yvv_1$ or $vwpb_1vy$: we can perform a blue double-edge swap $(v, v_1), (u, w) \rightarrow (v, w), (u, v_1)$; this makes (v, w) black and (u, v_1) blue, (v, v_1) red. Now perform red double-edge swap $(v, v_1), (u, x) \rightarrow (v, x), (u, v_1)$, this makes (u, v_1) black, (v, x) red. These swaps preserve the number of connected components as shown in Fig. 19 top and middle.

If v has a black neighbor v_1, and red cut-path on $uxpv_1vw$, blue cut-path on $wv(p)yvv_1$: we can perform a blue double-edge swap $(v, v_1), (u, w) \rightarrow (v, w), (u, v_1)$; this makes (v, w) black and (u, v_1) blue, (v, v_1) red. Now perform red double-edge swap $(v, v_1), (u, x) \rightarrow (v, x), (u, v_1)$, this makes (u, v_1) black, (v, x) red. These swaps preserve the number of connected components as shown in Fig. 19 bottom.

Remaining subcases are symmetric from the point of view of u using its neighbors connected through red or black edges. All of these cases are maintaining number of connected components while decreasing k by 2.

Subcase 2: there is only 1 black edge: The single edge present leads to symmetric cases, here we consider the black edge present between (v, x).

If blue cut-path is on $wvpyv$, the naive double-edge swap using only blue edges would create a cycle, but using blue double-edge swap $(v, x), (u, w) \rightarrow (v, w), (u, x)$, maintain CC as shown in Fig. 20.

If blue cut-path is on $vwpvy$, u will have either a red or black neighbor u_1 where (v, u_1) is empty. In either

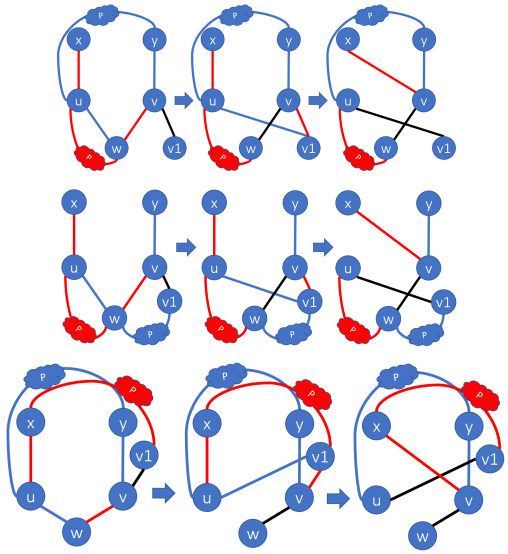


Fig. 19. Case 2, subcase 1: (v, v_1) is black and possible configurations of blue (red) cut-paths.

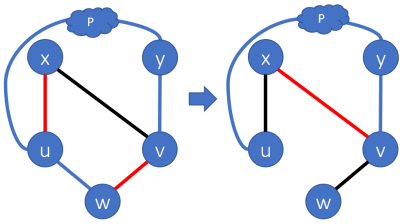


Fig. 20. Case 2, subcase 2: blue cut-path is on $wvpyv$

case, first perform red double-edge swap $(u, u_1), (v, w) \rightarrow (u, w), (v, u_1)$, this makes (u, w) black, (v, u_1) red. If (u, u_1) was red we can stop here, otherwise (u, u_1) became blue after the swap. However, we can now perform a blue double-edge swap $(u, u_1), (v, x) \rightarrow (u, x), (v, u_1)$, this makes (u, x) and (v, u_1) black, (v, x) red. Cases shown in Fig. 21.

Subcase 3: there are 2 black edges: Double-edge swaps using pairing nodes will not affect connectivity, example shown in Fig. 22.

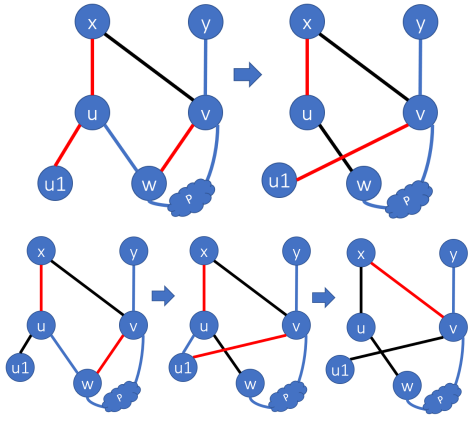


Fig. 21. Case 2, subcase 2: blue cut-path is on $vwpvy$, (u, u_1) is red (top) or black (bottom).

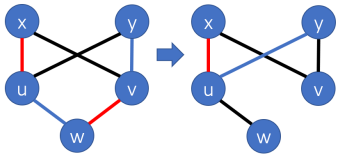


Fig. 22. Case 2, subcase 3: 2 black edges

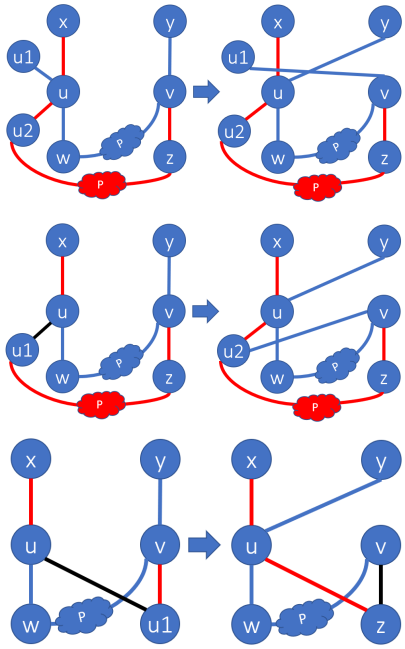


Fig. 23. Case 3, subcase 1: possible cases when cut-path is $uw(p)vy$

After performing any of these subcases the number of connected components will not change, but we have decreased k by 2 in every case.

Case 3: No pairing nodes exists: We can create pairing nodes without increasing the number of connected components. There will be a red, (u, x), and blue, (v, y), edge with same degree endpoints (u,v) and (x,y) in either a single large cycle (longer than 4) or 2 alternating-cycles. (u, y) and (v, x) can be only black or empty, otherwise there would be available pairing nodes. u, v will have at least one neighbor difference in both red and blue graphs.

Subcase 1: there is no black edge between endpoints: if trivial swaps increase CC, we can focus on blue cut-paths and blue connectivity and perform only blue double-edge swaps and use symmetric cases for red connectivity.

If cut-path is $uw(p)vy$, then there exists another neighbor of u not connected to v (in blue graph), u_1. We can perform blue double-edge swap $(u, u_1), (v, y) \rightarrow (v, u_1), (u, y)$, that makes (v, u_1) and (u, y) blue; when (u, u_1) was black it turns red, and when (v, u_1) was red it turns black. The double-edge swap creates pairing nodes x,y without change in k and other pairing nodes u,v at an increase in k by 2 when (u, u_1) was black while (v, u_1) was empty (before the swap), Fig. 23. If cut-path is $wuu_1(p)yv$, then the same blue double-edge swap can be performed using the first node u_1 on path, as shown in Fig. 24.

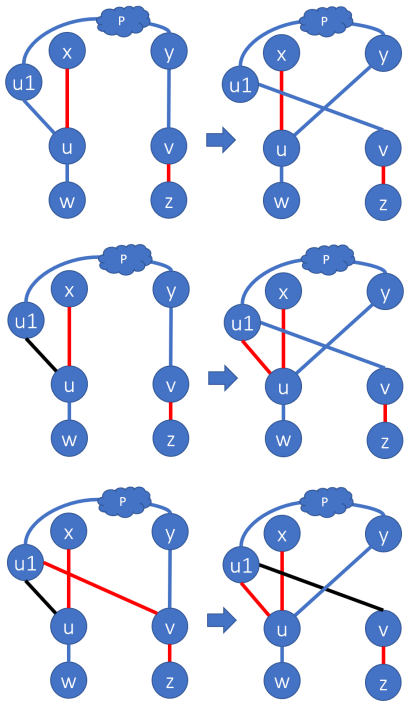


Fig. 24. Case 3, subcase 1: possible cases when cut-path is $wuu_1(p)yv$

Subcase 2: there is a single black edge between endpoints, (v, x) (symmetric cases exists if for (u, y)): If blue edges are from different components blue double-edge swap (using blue edges only) is viable and creates pairing nodes y,x, without increasing k or number of components. (u, y) is always empty and u, v have the same neighbors in the blue graph. Now we consider if they are in the same component in blue graph and where the blue cut-paths occur:

If cut-path is $uw(p)vy$, then there exist another blue or black neighbor of u, u_1 not connected to v, and not on any critical blue path. We can perform blue double-edge swap: $(u, u_1), (v, x) \rightarrow (v, u_1), (u, x)$, this makes (u, x) black, (v, x) red. If (u, u_1) was blue, then this creates pairing nodes x,y, while not increasing k. If (u, u_1) was black and (v, u_1) was empty, then this creates two pairing nodes x,y and u,v while increasing k by 2. If (v, u_1) was red, only pairing nodes are x,y and k was not increased. Of course, the number of components have not changed. Cases shown in Fig. 25.

If cut-path is $uu_1(p)yv$ where u_1 is not connected to v, we can perform blue double-edge swap: $(u, u_1), (v, y) \rightarrow (v, u_1), (u, y)$, that makes (u, y), (v, u_1) blue, (u, u_1) red if (u, u_1) was black initially. It creates pairing nodes, x,y without increasing k. When (u, u_1) was black, it also creates u,v pairing nodes and k increases by 2, Fig. 26.

If cut-path is $uu_1(p)xv$ where u_1 is not connected to v, we can perform blue double-edge swap: $(u, u_1), (v, x) \rightarrow (v, u_1), (u, x)$, that makes (u, x) black, (v, u_1) blue, (v, x) red, and (u, u_1) red if (u, u_1) was black initially. It creates pairing nodes, x,y without increasing k. When (u, u_1) was black, it also creates u,v pairing nodes and k increases by 2, Fig. 27.

Subcase 3: there are two black edges between endpoints: We discuss the possible cases from the point of view of v's

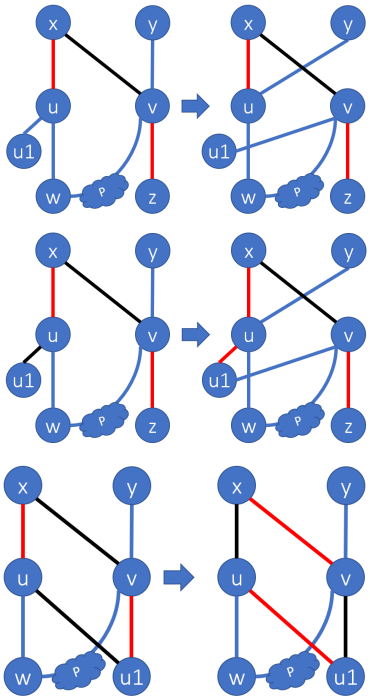


Fig. 25. Case 3, subcase 2: possible cases when cut-path is $uw(p)vy$

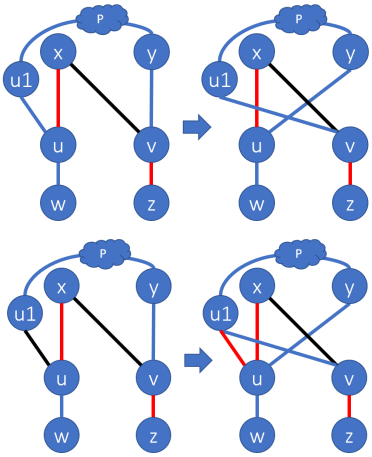


Fig. 26. Case 3, subcase 2: possible cases when cut-path is $uu_1(p)yv$

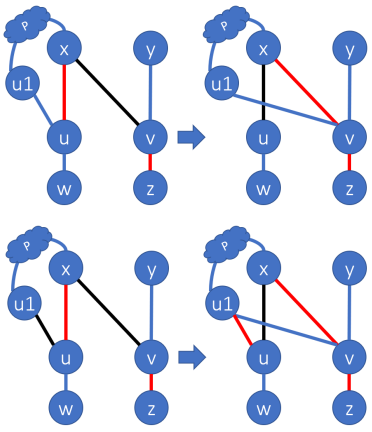


Fig. 27. Case 3, subcase 2: possible cases when cut-path is $uu_1(p)yv$

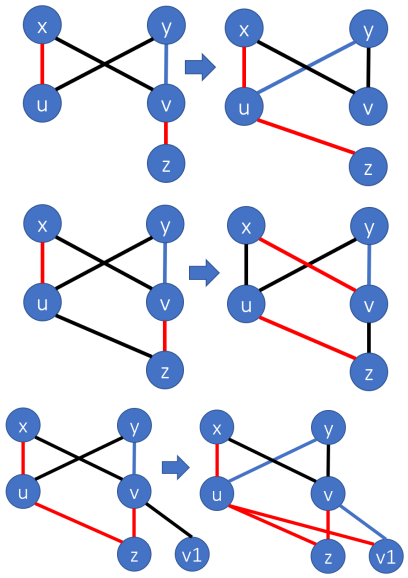


Fig. 28. Case 3, subcase 3: possible cases depending on the color of (u, z) edge.

neighbors in the red graph, however there are symmetric cases for u as well. v has at least one red neighbor, z:

If z has no edge to u, then the red double-edge swap $(v, z), (u, y) \rightarrow (v, y), (u, z)$ makes (u, y) blue, (y, v) black, (v, z) red; and it creates pairing nodes: x,y. The swap maintains the number of components and does not increase k, Fig. 28 top. If z has a black edge to u, then the blue double-edge swap $(v, x), (u, z) \rightarrow (v, z), (u, x)$ makes $(v, x), (u, z)$ red and $(v, z), (u, x)$ black; and it creates pairing nodes: x,y. The swap maintains the number of components and does not increase k, Fig. 28 middle. Main case 2 will decrease k afterwards by 2 (4-cycles are not present since that would mean x,y were already pairing nodes). If z has a red edge to u, then there must exist a black neighbor v_1 that is not connected to u; and the red double-edge swap $(v, v_1), (u, y) \rightarrow (v, y), (u, v_1)$ makes $(u, y), (v, v_1)$ blue, (y, v) black, (u, v_1) red; and it creates pairing nodes: x,y and u,v. The swap maintains the number of components and increases k by 2, Fig. 28 bottom.

In all subcases for case 3, the two pairs of pairing nodes can be handled sequentially x,y first (main case 2) without changing u's or v's edges connecting to their neighbors except x, y. Then the pairing nodes, u,v can be resolved by main case 2, thus decreasing k by 2=+2-4. □

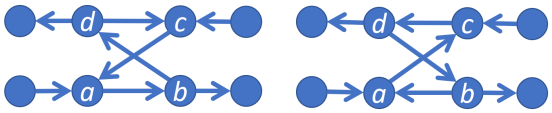


Fig. 29. Two realizations with the same degree sequence and JDAM. There is no JDAM preserving double-edge swap that would not use any self-loops and the C_6 swaps are not preserving JDAMs. This shows that the edges along the directed 4-cycle must change their direction simultaneously.

APPENDIX E
ADDITIONAL D2K SIMULATIONS AND RESULTS

This Appendix supplements Section IV-B3 and IV-D of the main paper.

A. D2K: Space of realizations

The space of simple realizations of directed degree sequences (D1K) is connected over double edge swaps, that preserve (in and out) degrees, and triangular C_6 swaps. If the difference between two realization is the orientation of a directed three-cycle, then the triangular C_6 swap consists of edge rewirings such that the orientation of the cycle is reversed in a single step. The sufficiency of only these two types of swap was shown in [43]. The necessity of these swaps also carries over to (simple) directed 2K realizations. However, Fig. 29 shows a counterexample (a directed 4-cycle) where the classic swaps are not sufficient to transform one realization to the other, thus requiring a more complex swap. We leave it as an open question whether tight upper bounds can be derived on the swap size for Directed 2K realizations.

There are possibly other cases where swaps must be more complex and include more edges at once, for example larger directed cycles with specific in/out degree order. In this paper, we do not provide tight upper bounds on the number of self-loops (in the directed graph representation) or the size of swaps required, but we do emphasize that no multi-edges are required and the number of self-loops are of course bounded by $|N|$.

B. List of Properties used in D2K simulations

Here we elaborate on the definition of the properties we used to evaluate how well D2K performs.

1) *Dyad Census* counts the different configurations for every pair of nodes: ”mutual” - edges in both direction, ”asymmetric” - edge only in one direction and ”null” - no edge present.
2) *Triad Census* counts the non-isomorphic configuration for every triplet of nodes. A complete list of configurations and naming conventions can be found in [25]. Configurations are identified by three numbers (mutual, asymmetric, and null counts) and a letter in case of different non-isomorphic configuration with the same number of edges. For example ”003” is a triplet of nodes where none of the edges are present, ”030C” is a directed 3-cycle and ”300” is a triplet of nodes where all directed edges are present.
3) *Dyad-wise Shared Partners* for pairs of nodes can be defined in three ways for directed graphs: using independent two-paths, using shared outgoing neighbors or using shared incoming neighbors between pairs of nodes [44]. Dyad-wise shared partners (DSP) count node pairs by the number of shared partners appearing in a network.
4) *Average Neighbor Degree* captures the average degree of a nodes’ neighbors, and we split this property for in - and out degrees. Similarly, we refer to *Expansion* for directed graphs as the ratio of the first hop and second hop neighborhoods’ sizes going out, or coming in to a node. These properties capture similar aspects of a network, but expansion excludes any mutual edges or edges between nodes in the first hop neighbors.
5) *Betweenness Centrality CDF*, *Shortest Path Distribution*, *K-Core Distribution*, *Eigenvalues*

C. Additional D2K Simulation Results: beyond Twitter

In Section IV.D, we presented results from applying our D2K framework to generate synthetic graphs that resemble a Twitter dataset obtained from SNAP, which is a representative and interesting real-world directed graph. In this section, we present additional results for other real-world directed graphs obtained from SNAP, namely: p2p-gnutella, Wiki-Vote, AS-Caida. These were omitted from the main manuscript due to lack of space.

TABLE IV
SUMMARY OF RESULTS: SHOWING IMPROVEMENTS BY FIXING MORE PROPERTIES. LABELS: ”.” - NO IMPROVEMENT, ”-” - DECREASED ACCURACY, ”+” - INCREASED ACCURACY, ”EXACT” - MATCHED BY DEFINITION.

Property	UMAN→D1K	D1K→D2K	D2K→D2.1K
Degree Distribution	Exact	Exact	Exact
Degree Correlation	+	Exact	Exact
Dyad Census	-	+	+
Triad Census	+	+	+
Betweenness Centrality	+	.	.
Shortest Path Distribution	+	+	+
Eigenvalues	+	+	+
DSP	+	+	+
Expansion	+	+	+
Avg. Neighbor degrees	+	+	Exact
S. Connected Components	.	.	.
K-Core Distribution	+	.	+

Table IV gives an overview of how network properties are affected by the different dK graph construction methods for the other remaining networks. The Twitter network showcased most of our general findings, but individually some of these networks have characteristics that makes them different from Twitter, *e.g.,* p2p-Gnutella08 does not contain any mutual edges. The most interesting question is whether D2K or D2.1K capture network properties more accurately. The answer is yes in most cases, but it might not be a significant improvement in targeting certain properties.

Local structures are generally better captured by D2K and even more precisely for D2.1K, but global properties might not be significantly affected depending on the original network. However, this result is not surprising, since one of the main assumptions of the dK-series is that it is not necessary to target high d values for every graph [6].

Figures 30-38 show detailed results for additional graphs (p2p-Gnutella08, Wiki-Vote, AS-Caida) in the same format as we have seen in Section IV-D for Twitter.

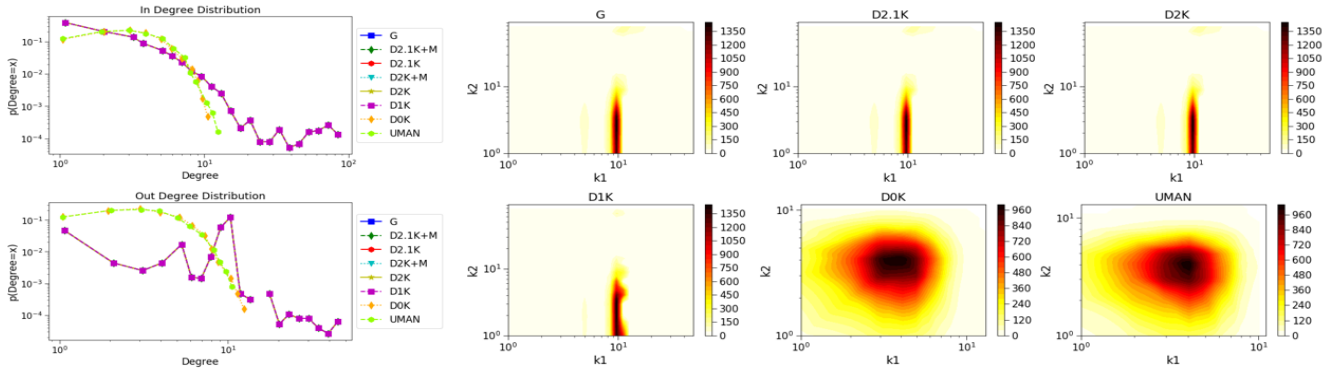


Fig. 30. Results for p2p-Gnutella08 graph: Directed Degree Distribution and Degree Correlation

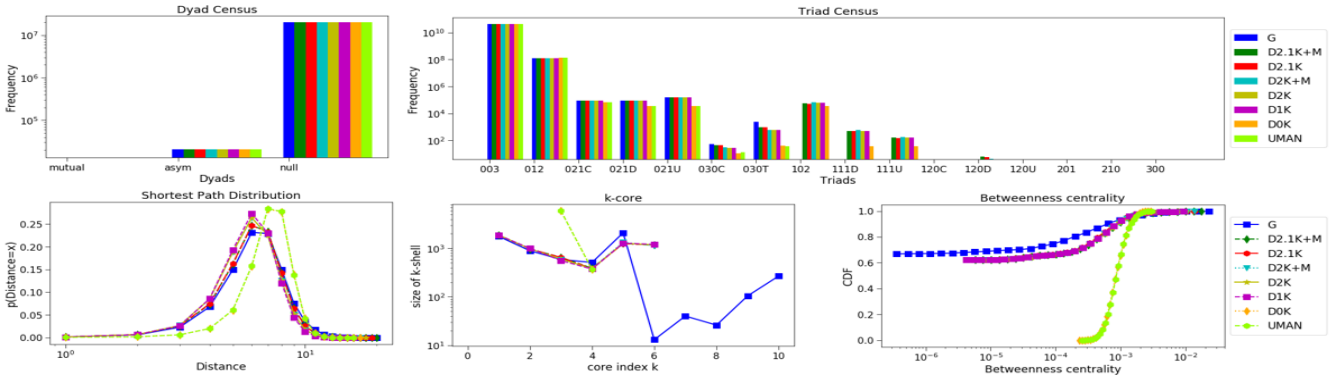


Fig. 31. Results for p2p-Gnutella08 graph: Dyad-, Triad Census, Shortest Path Distribution, K-core distribution, Betweenness Centrality

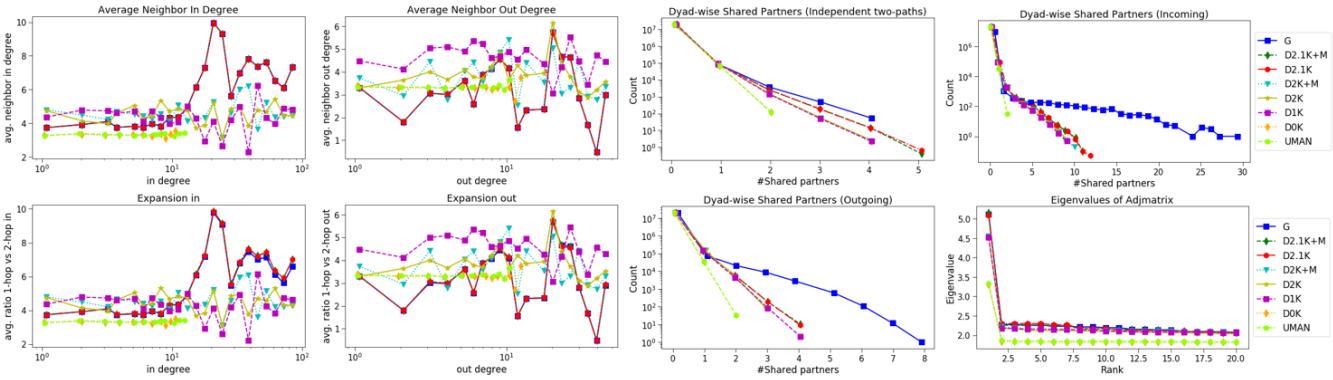


Fig. 32. Results for p2p-Gnutella08 graph: Expansion, Average Neighbor Degree, DSP and top 20 Eigenvalues

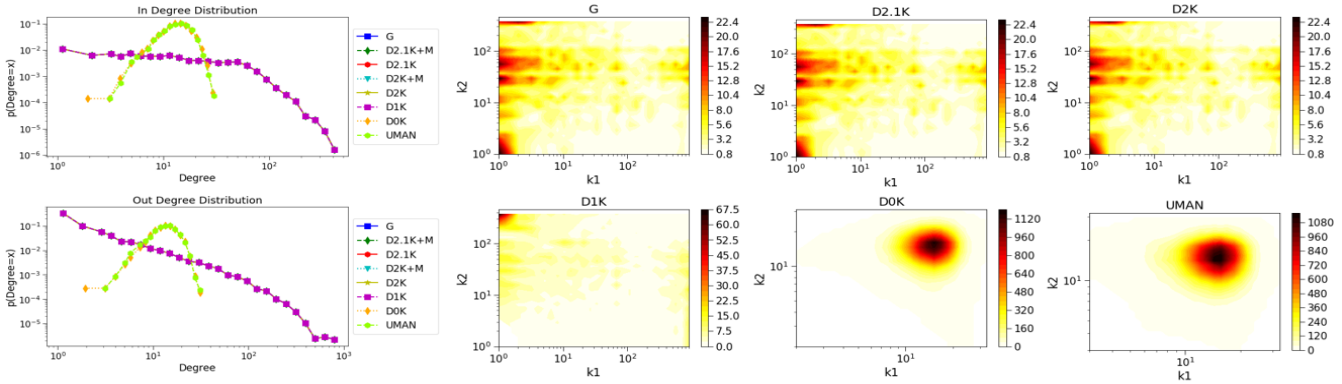


Fig. 33. Results for Wiki-Vote graph: Directed Degree Distribution and Degree Correlation

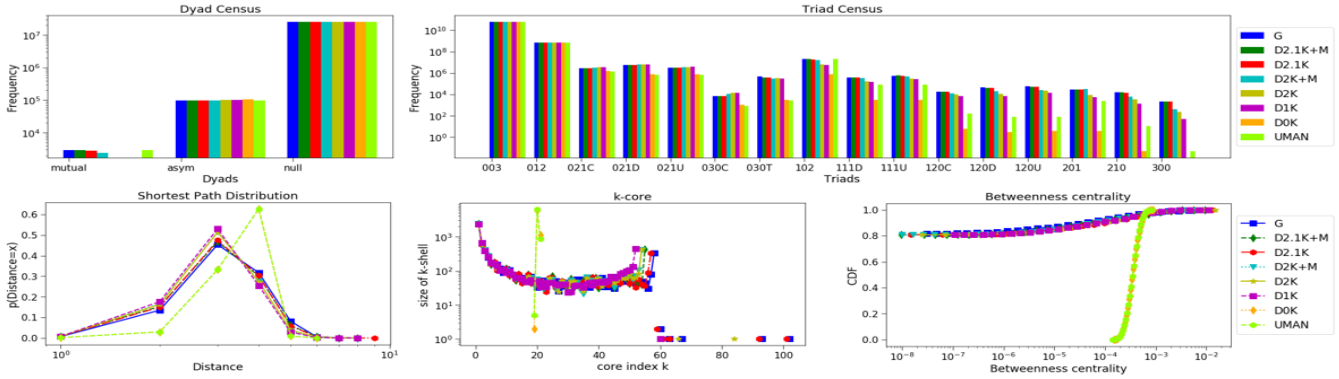


Fig. 34. Results for Wiki-Vote graph: Dyad-, Triad Census, Shortest Path Distribution, K-core distribution, Betweenness Centrality

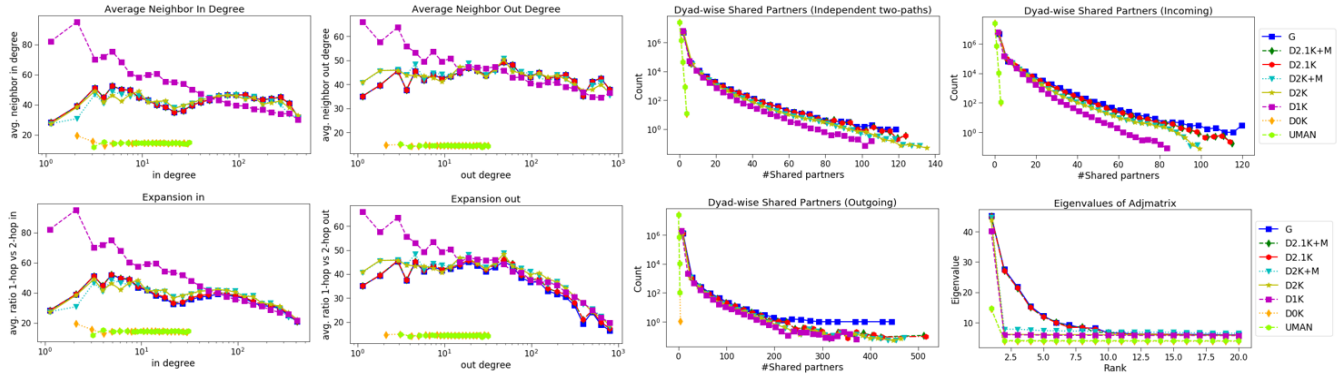


Fig. 35. Results for Wiki-Vote graph: Expansion, Average Neighbor Degree, DSP and top 20 Eigenvalues

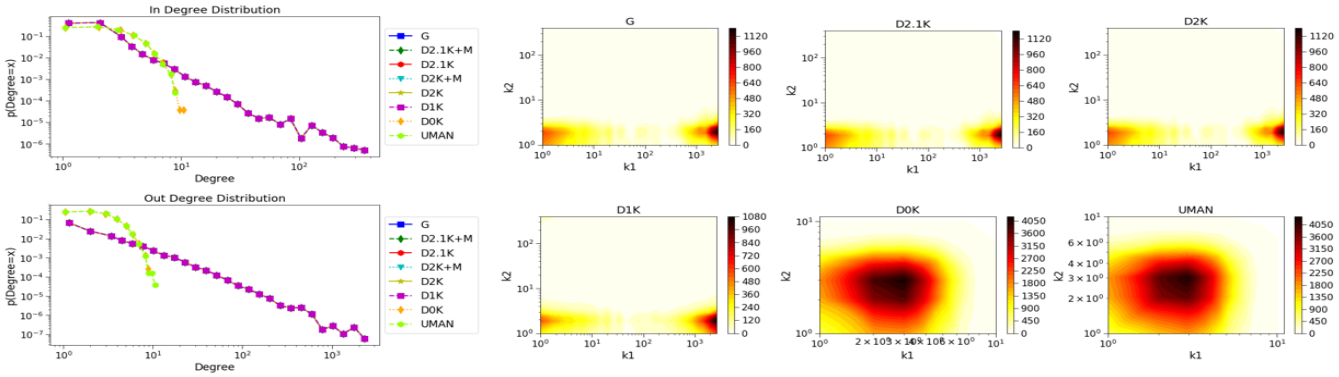


Fig. 36. Results for AS-Caida graph: Directed Degree Distribution and Degree Correlation

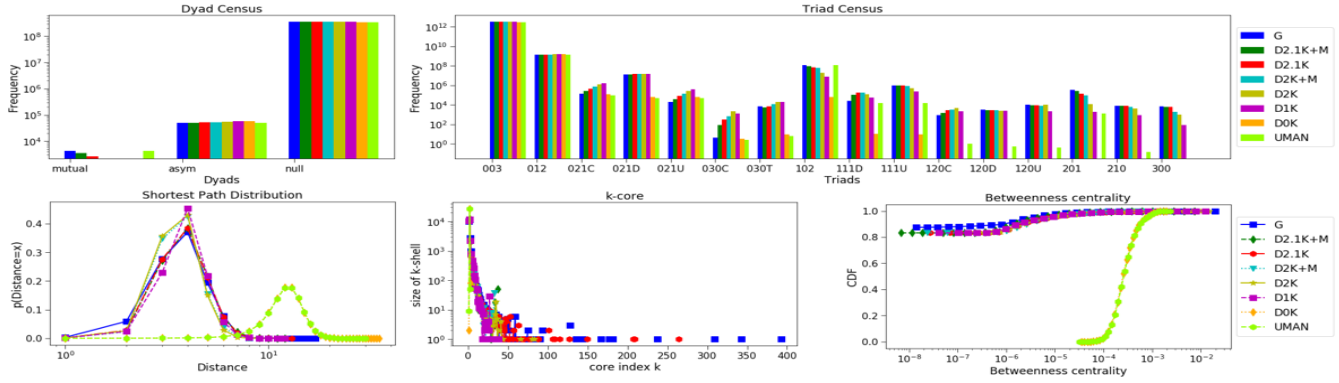


Fig. 37. Results for AS-Caida graph: Dyad-, Triad Census, Shortest Path Distribution, K-core distribution, Betweenness Centrality

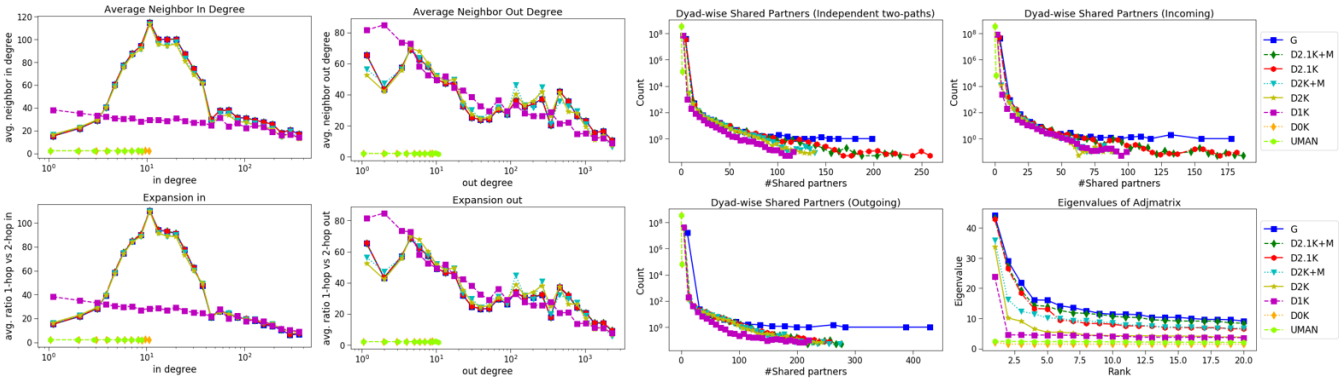


Fig. 38. Results for AS-Caida graph: Expansion, Average Neighbor Degree, DSP and top 20 Eigenvalues

APPENDIX F
BALANCED REALIZATIONS FOR 2K+MINCC AND D2K

This appendix provides additional details skipped from Section III-D and IV-C of the main paper.

In this section, we follow the notation from Czabarka *et al.* [9]. A graph, G, has a node partition according to their degree V_0 to $V_{d_{max}}$. V_i is the set of nodes with degree i. For every V_j, set $A_j(j) := JDM(j,j)/|V_j|$ and for $i \neq j, A_j(i) := JDM(i,j)$. Now define $\forall i, s_G(v)_i$ for every $v \in V_j$ as the number of edges from v to nodes with degree i. A realization is balanced if for all i,j pairs $s_G(v)_i \in \{\lfloor A_j(i) \rfloor, \lceil A_j(i) \rceil\}$ for all $v \in V_j$. Identically it is balanced if the floor of the difference from the average connectivity (defined by matrix A) of every node is 0, formally we define C_G the difference from balanced for a node v to a degree group i as $c_G(v,i) := \lfloor \|A_{deg(v)}(i) - s_G(v)_i\| \rfloor$; and for a degree group j as $C_G(j) = \sum_{v \in V_j} \sum_{i=1}^{d_{max}} c_G(v,i)$.

Balanced Degree Invariant realizations of JDMs always exists as shown in [9]. Here we show that Lemma 4 and Corollary 5 from [9] can be applied with minor modifications to find balanced realizations of 2K with minimum number of connected components or D2K.

Lemma 8. *If $\exists u, v \in V_j : s_G(u)_i < \lfloor A_j(i) \rfloor < s_G(v)_i$ or $\lceil A_j(i) \rceil$ handled similarly) for a simple graph, G , then $\exists w, w' \in V_i : \{(v,w),(v,w')\} \in E, \{(u,w),(u,w')\} \notin E$ and $w, w' \neq u$; $\exists z, z' \in V_k, k \neq i$ or $\exists z \in V_k, z' \in V_{k'}, k, k' \neq i : \{(u,z),(u,z')\} \in E, \{(v,z),(v,z')\} \notin E$ and $z, z' \neq v$.*

Proof. From the initial conditions follow that $u, v \in V_i$, $deg(u) = deg(v) = i$ and $s_G(v)_i - s_G(u)_i \geq 2$. Since the sum of $s_G(v)$ equals to i and every value is an (non-negative), integer, in worst case ($s_G(v)_i - s_G(u)_i = 2$) there $\exists k \neq i$ such that $s_G(v)_k - s_G(u)_k = 2$ or $\exists k, k' \neq i$ such that $s_G(v)_k - s_G(u)_k = 1$ and $s_G(v)_{k'} - s_G(u)_{k'} = 1$.

It follows from the previous statement, that $\exists w, w' \in V_i$ such that $\{(v,w),(v,w')\} \in E, \{(u,w),(u,w')\} \notin E$; and $\exists z, z' \in V_k$, or $\exists z \in V_k, z' \in V_{k'}$ such that $\{(u,z),(u,z')\} \in E, \{(v,z),(v,z')\} \notin E$.

We have to consider whether $w, w' \neq u$ and $z, z' \neq v$: if $w = u$, then $i = j$ and $(u,v) \in E$ ($z = v$ handled similarly). By removing (u,v) edge, the difference $s_G(v)_i - s_G(u)_i$ remains the same, which means that there exists $w, w' \neq u$. □

We just showed, that Lemma 4 in [9] has the option for both u, v to choose between two nodes for the RSO.

Theorem 9. *If $C_G(j) \neq 0$, there are nodes $u, v \in V_j$ and an RSO $vw, uz \to vz, uw$ transforming G into G' such that $C'_G(j) < C_G(j)$; $\forall l \neq j, C'_G(l) = C_G(l)$ and $|CC(G')| \leq |CC(G)|$.*

Proof. Now that there are at least two neighbors for both u, v two use in an RSO while applying Lemma 4 [9], we can identify cases to maintain number of connected components, based on the path between u, v and w, z:

Case 1. There is no path between u, v, *i.e.,* u, v are in different connected components. Any RSO will not increase the number of connected components, thus $|CC(G')| \leq CC(G)|$.

Case 2. There is a path $v - w - p - z - u$ (where p is a simple path between w, z). An RSO using w, z would return a new path $v - z - p - w - u$ thus $|CC(G')| = CC(G)|$.

Case 3. There is a path $w - v - p - u - z$ (where p is a simple path between v, u). An RSO using w, z would return a new path $z - v - p - u - w$ thus $|CC(G')| = CC(G)|$.

Case 4. There is a path $v - w - p - u - z$ (where p is a simple path between w, u). An RSO using w, z would return new paths $v - z, w - p - u$. If there are no other paths connecting these subgraphs, then $|CC(G')| + 1 = CC(G)|$. However, using our previous observation, we can use w' that would lead to a path $w' - v - w - p - u - z$. This is in fact Case 3 using w' instead of w. □

These extensions to Lemma 4 do not change Corollary 5 in [9]; which means that application of Corollary 5 will result in the necessary sequences to return balanced realizations without increasing number of connected components. If the input G was already a MinCC realization, then the resulting G' will be both MinCC and balanced realization.

Next we use the same observation to construct balanced realizations for D2K graphs as well.

Lemma 10. *If $C_G(j) \neq 0$, then there are nodes $u, v \in V_j$ and an RSO $vw, uz \to vz, uw$ transforming G into G' such that $C'_G(j) < C_G(j)$ and $\forall l \neq j, C'_G(l) = C_G(l)$ if every node participates in exactly one non-chord.*

Proof. Now that there are at least two neighbors for both u, v two use in an RSO while applying Lemma 4 [9], we can find an RSO without using non-chords. Since u has exactly one

non-chord, it can pick (at least) one of w, w' and similarly v can pick one of z, z' for an RSO. □

As before, we can apply Lemma 4 and Corollary 5 from [9] in combination with the previous lemma to produce a balanced realizations for any realizable D2K inputs.