Construction of Simple Graphs with a Target Joint Degree Matrix and Beyond

Minas Gjoka, Bálint Tillman, Athina Markopoulou
University of California, Irvine
{mgjoka, tillmanb, athina}@uci.edu

***Abstract*—In networking research, it is often desirable to generate synthetic graphs with certain properties. In this paper, we present a new algorithm, `2K_Simple`, for exact construction of simple graphs with a target joint degree matrix (JDM). We prove that the algorithm constructs exactly the target JDM and that its running time is linear in the number of edges. Furthermore, we show that the algorithm poses less constraints on the graph structure than previous state-of-the-art construction algorithms. We exploit this flexibility to extend `2K_Simple` and design two algorithms that achieve additional network properties on top of the exact target JDM. In particular, `2K_Simple_Clustering` produces simple graphs with a target JDM and average clustering coefficient close to a target, while `2K_Simple_Attributes` produces exactly simple graphs with a target JDM and joint occurrence of node attribute pairs. We exhaustively evaluate our algorithms through simulation for small graphs, and we also demonstrate their benefits in generating graphs that resemble real-world social networks in terms of accuracy and speed; we reduce the running time by orders of magnitudes compared to previous approaches that rely on Monte Carlo Markov Chains.**

I. Introduction

In computer network research, we often need to generate synthetic graphs with certain key properties, *e.g.,* resembling those of real networks of interest. For example, one may want to simulate a routing algorithm on an AS-like topology, to study influence on a social network-like graph, or to anonymize a known graph while preserving some key characteristics. In the first two examples, access to real networks may be restricted, measurement may be impractical, datasets -even if available- may be too large, and networks may evolve over time. In all these cases, researchers need to generate synthetic graphs that exhibit certain target properties.

When constructing a graph, one has first to choose the properties that the generated graph should resemble the real graph. This is in itself a challenging research question (and not the contribution of this paper - the target properties are an input to our construction problem). We adopt the systematic framework of dK-series [12], which characterizes the properties of a graph using a series of probability distributions specifying all degree correlations within d-sized 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. For example, $0K$ specifies the average node degree of the graph; $1K$ specifies the node degree distribution; $2K$ specifies the joint degree distribution; $3K$ specifies the degree-dependent distribution of all connected subgraphs of 3 nodes, etc. Interestingly, a dK-distribution includes all properties defined by any $d'k$-distribution, $\forall d' < d$; *e.g.,* $2K$ specifies $1K$ and $0K$. Selecting the right d to model a graph is highly application specific, and involves a tradeoff between accurate representation of the original graph and complexity.

This work has been motivated by, and builds on, prior attempts to generate graphs that resemble online social networks. In terms of structure, social networks are well known to exhibit high clustering. The 2K-distribution cannot, by definition, capture clustering. The 3K-distribution captures all information about subgraphs of three nodes, but no efficient algorithms for estimating 3K parameters or constructing graphs with a target 3K, are known yet. *2K+* (*e.g.,* the JDM *and* some notion of clustering) provides a sweet spot between an accurate representation of online social networks and practical constraints in estimation and construction.

2K. Our first contribution is `2K_Simple`: a new algorithm for constructing simple graphs with a target joint degree matrix (JDM). We prove that the algorithm can generate graphs with the exact target JDM, for any realizable JDM, and it does so in linear running time in the number of edges. Prior methods for constructing 2K graphs were based either on the configuration model [12], which can produce multigraphs; or on a balanced degree invariant [18], which significantly constrains the edges of the constructed simple graph. All three algorithms, start from a set of nodes and no edges, and iteratively add one edge at a time until the graph is completed. A major advantage of `2K_Simple` is its increased flexibility in adding the edges, which allows the algorithm to produce a larger set of simple graphs (possibly all) with the same target 2K, compared to prior approaches [18]. We demonstrate this result with simulations that exhaustively produce all seven node graphs. `2K_Simple` can essentially be used as a template algorithm that, depending on the order in which edges are added, leads to different graphs, all with the same target JDM.

2K+. We exploit this flexibility to make our second contribution: we design two algorithms that construct graphs that not only have the exact target JDM but also exhibit additional properties. `2K_Simple_Clustering` produces simple graphs with exactly the target JDM and average clustering coefficient close to a target $\bar{c}$, and it does so in approximately linear time in the number of edges. We can construct graphs with low or high $\bar{c}$, but when targeting real-world graphs with high clustering, the benefit is significant: it can reduce the running time by order of magnitudes, compared to previous approaches that rely on MCMC. `2K_Simple_Attributes` produces simple graphs with exactly the target JDM *and*, for the first time, exactly joint occurrence of node attribute pairs. This is important for social networks, where nodes typically have several attributes, and there are dependencies between attributes and network structure. We exhaustively evaluate

our algorithms through simulation for small graphs. We also demonstrate their benefits in generating graphs resembling real-world social networks; the construction time for a set of Facebook topologies is reduced from days to 10s of seconds.

The structure of the rest of the paper is as follows. Section II describes common notation, while additional notation is deferred to the relevant sections. Section III reviews related work. Section IV describes our 2K algorithm `2K_Simple`, for exact construction of JDM, proves its correctness and running time. Section V presents 2K+ algorithms, namely `2K_Simple_Clustering` and `2K_Simple_Attributes` for constructing exact JDM and clustering or node attributes, respectively. Section VI presents simulation results. Section VII concludes the paper.

II. Notation

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.

The *Joint Degree Matrix (JDM)* is defined as the number of edges connecting nodes of degree k with nodes of degree l and describes the degree correlations.

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

Note that the joint degree distribution (JDD) can be derived by normalizing every entry in the JDM with the value $m = |E|$. Also, there may be more than one graphs with the same JDM but with other differences in terms of structure or attributes. A given JDM implies the degree vector D_k, the number of edges $m = |E|$, and the number of nodes $n = |V|$ in the graph. Indeed, the degree vector D_k is defined as the number of nodes of degree k, $\forall k$.

$$D_k = |V_k| = \frac{1}{k} \sum_{l=1}^{d_{max}} \text{JDM}(k,l) \quad (2)$$

where d_{max} is the maximum degree in the graph.

Clustering. If two neighbors of a node v are connected, then these three nodes form a triangle. Let T_v be the number of triangles attached to node v. Then, 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.

$$c_v = \frac{T_v}{\deg(v)(\deg(v)-1)/2} \quad \text{and} \quad \bar{c} = \frac{1}{n} \sum_v c_v. \quad (3)$$

We use $^{\odot}$ to denote the target properties, that the constructed a graph should have; absence of $^{\odot}$ denotes the actual values for the constructed, or partially constructed, graph.

We defer additional definitions to their respective sections.

III. Our Work in Perspective

dK-series. We adopt the systematic framework of dK-series [12], which characterizes 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).

Graph Construction Approaches. There are two main approaches to construct graphs for a given d. The stochastic approach [8] achieves the target distribution in expectation, by connecting random d-subgraphs, and generalizes for any d. The problem is that individual realizations may be far from the target distribution. The configuration [3] (or pseudograph) approach aims to construct a graph with the exact target distribution, and it is the approach we follow in this paper.

dK-construction. Algorithms of known time complexity exist up to $d = 2$, and Monte Carlo Markov Chain (MCMC) approaches are used for $d > 2$.

1K Construction. The configuration model was introduced in [14] to generate random graphs with a given 1K and allows multi-edges and self-loops. [5] presented a method that constructs a simple graph with a given 1K. There are several methods that attempt to construct graphs with a given 1K and additional properties: [4] targets 1K and $\bar{c}$ with an MCMC approach; [17] targets 1K and the degree-dependent average clustering coefficient; [15] proposed a model that extends the configuration model by specifying the number of edges and triangles attached to a node.

2K Construction. There are two previous algorithms [12,18] that provably construct graphs with exactly $JDM^{\odot}$ and their running time is linear in the number of edges. Therefore, they are baselines for comparison against `2K_Simple`.

(2K-I) `2K_Configuration`. Mahadevan [12] *et al.* extended the configuration approach, originally designed for 1K [3], to 2K, which we refer to as `2K_Configuration`. It starts from disconnected nodes, each with as many stubs attached as its degree. It proceeds iteratively by randomly connecting two stubs to form an edge. However, [12] allows the addition of self-loop edges and multiple edges, thus can produce multigraphs,[1] while we are interested in simple graphs, which is more challenging.

(2K-II) `2K_BDI`. Stanton [18] *et al.* presented an algorithm that provably constructs a simple graph based on the Balanced Degree Invariant (BDI), first introduced in [21]. In each iteration, the algorithm adds all edges $\text{JDM}^{\odot}(k,l)$ between a selected degree pair (k,l), while satisfying the BDI. The BDI requires that the difference in the number of free stubs for every pair of vertices (v,w) that belong to a degree group k, is no larger than one. This essentially spreads the number of edges evenly across nodes and maximizes the number of nodes with free stubs, thus maximizing the options of the algorithm in subsequent steps. In general, BDI constrains significantly the space of generated graphs.

3K Construction. There are currently no efficient algorithms for estimating 3K parameters of large graphs or for construct-

[1] We have tried to modify this approach to produce a simple graph, *e.g.,* by either forbidding the addition of self-loops or multi-edges during the edge-adding phase of the algorithm; or by removing self-loops and multi-edges at the end. In either case, the algorithm gets stuck without being able to produce a simple graph with the $JDM^{\odot}$. We omit details due to lack of space.

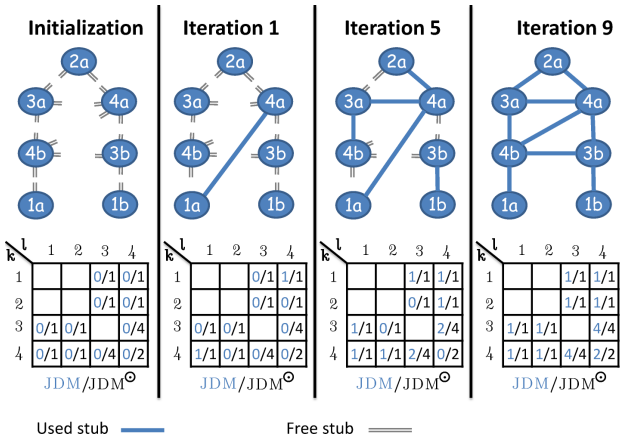


Fig. 1. Example of running `2K_Simple`. The algorithm starts from disconnected nodes with free stubs (left). In each iteration it creates one edge by connecting 2 free stubs (between two nodes whose degree pair has not reached the target JDM yet) and it increases the corresponding entries in JDM by one. At the end (right) the graph is complete and the JDM reaches the target.

ing graphs with a target 3K; only MCMC approaches are used.

Monte Carlo Markov Chains (MCMC). Methods that target a property on top of a dK-distribution typically perform edge rewiring moves that preserve the dK-distribution while exploring the space of graphs with the given dK-distribution. For example, to preserve a 2K distribution, a "double-edge swap" is typically used [12], and the move is accepted if it produces a graph closer to the target, w.r.t the additional property. In practice, MCMC methods are slow and the number of iterations needed is not known apriori. In contrast, the 2K+ class of algorithms we present in this paper have approx. linear time, provide guarantees and additional desirable properties.

2K+ Construction. This paper was primarily motivated from graphs, such as social networks, for which 2K is insufficient to capture their properties and 3K is too complex to deal with. In [9], we provided heuristic algorithms for constructing simple graphs with a given JDM and as high as possible degree-dependent average clustering coefficient. However, the method that targets 2K and clustering relies on MCMC and thus has no guarantees on the running time. In contrast, our 2K construction algorithm, `2K_Simple`, provides provable guarantees for the generation of simple graphs. Our extensions `2K_Simple_Clustering` and `2K_Simple_Attributes` achieve additional properties (*e.g.,* any target average clustering and nodal attributes, on top of 2K) with running time linear in the number of edges.

Node attributes. [16] and [11] modeled network structure with observed node attributes in expectation; our `2K_Simple_Attributes` algorithm achieves exactly the joint occurrence of node attributes. [7] introduced node attributes on top of dK-series framework. However, their approach produces graphs with multi-edges and self-loops whereas we provably produce simple graphs with the exact targeted attribute correlations.

IV. 2K Construction: Target JDM

A. Construction Algorithm: `2K_Simple`

We design a new algorithm, called `2K_Simple`, that receives as input a target JDM$^{\odot}$ and creates a *simple* graph that has provably the *exact* JDM$^{\odot}$. The target JDM$^{\odot}(k,l)$ is assumed to be realizable.[2] The algorithm is summarized next in Algorithm 1 and is illustrated in the example of Fig.1.

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 disconnected nodes $v \in V_k$ and $w \in V_l$
4: **if** v does not have free stubs
5: NeighborSwitch(v)
6: if w does not have free stubs
7: NeighborSwitch(w)
8: add edge between (v, w)
9: $JDM(k,l)$ ++; $JDM(l,k)$ ++;
Output: simple graph with $JDM = JDM^{\odot}$

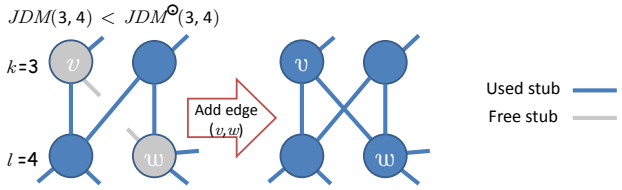
The initialization phase is depicted in the leftmost column of Fig. 1. We create $|V| = n$ nodes, labeled by their degree. (Note that $|V|$ and $D_k^{\odot}$ can be found from JDM$^{\odot}(k,l)$.) Then, we assign $deg(v)$ free stubs to every node $v \in V$ according to their degree. (Stubs are the "half" edges shown in the top-left part of Fig.1.) All nodes are initially disconnected and we initialize $JDM(k,l)$ to zero for all degree pairs (k,l).

Then the algorithm proceeds in iterations. In each iteration, we pick two disconnected nodes v and w of degree k and l correspondingly, for which $JDM(k,l)$ has not reached its target yet ($JDM(k,l) < JDM^{\odot}(k,l)$ in line 2 of Algorithm 1). We connect a free stub from v with a free stub from w to create edge (v,w) (blue color in Fig. 1) and we increase the corresponding two entries $JDM(k,l)$ and $JDM(l,k)$ by one. The algorithm continues until all entries of the current JDM have reached their target values in JDM$^{\odot}(k,l)$, at which point the construction is complete.

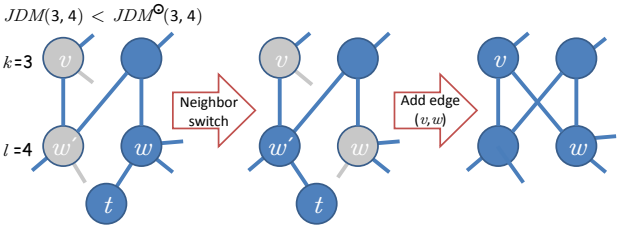
The intelligence of the `2K_Simple` algorithm lies in the fact that it is always possible to add an edge between disconnected nodes v and w, for which $JDM(deg(v), deg(w))$ has not reached its target, even if one or both nodes do not have free stubs. Before adding edge (v,w), we perform an edge rewiring, which we refer to as *neighbor switch*, for each node without a free stub.

We define a node that does not have free stubs as *saturated*, and a node that has at least one free stub as *unsaturated*. A neighbor switch 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 by (i) finding an unsaturated node w' with the same degree as w, (ii) a neighbor t of w that is not connected to w' (iii) removing edge (w,t) and adding edge (w',t). Because $deg(w) = deg(w')$, the neighbor switch preserves the $JDM(deg(w), deg(t))$. The following

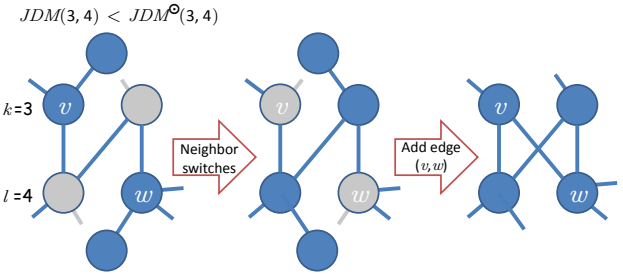
[2] A matrix must satisfy some intuitive (*e.g.,* JDM must be symmetric, etc) necessary and sufficient, conditions in order to be realizable, *i.e.,* to correspond to at least one graph. These have been derived in [18] and are not repeated due to lack of space.



(a) No neighbor switch needed: nodes v, w have a free stub each, which we can connect to create edge (v, w)



(b) 1 neighbor switch needed: v has a free stub but w does not. A neighb. switch (connecting t to w' instead of w) frees stub for node w.



(c) 2 neighbor switches are needed: neither v or w have free stubs.

Fig. 2. All possible cases of adding an edge (v, w) between node v (of degree k) and node w (of degree l). Blue color nodes and edges correspond to saturated nodes and used stubs. Grey color nodes and stubs correspond to unsaturated nodes and free stubs.

pseudocode summarizes a neighbor switch, which is also illustrated in Fig.2(b).

NeighborSwitch(node w):
1: $w' \leftarrow$ unsaturated node s.t. $deg(w')$=$deg(w)$
2: $t \leftarrow$ neighbor of w, not connected to w'
3: remove edge (w, t)
4: add edge (w', t)

Fig. 2 shows all possible cases of adding an edge between disconnected nodes v (of degree k) and w (of degree l). Fig. 2(a) shows the simplest case when both nodes v and w are unsaturated and no neighbor switch is necessary to add an edge between them. In Fig. 2(b), one neighbor switch is needed, because node w is saturated. In Fig. 2(c), both v and w are saturated and we need to perform two neighbor switches.

B. Proof of correctness

Theorem 1. `2K_Simple` *terminates and constructs a 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 2 shows that the algorithm will not get stuck, *i.e.,* , if we have not reached the target, it is possible to find v, w nodes where an edge can be added. Lemmas 3 and 4 show that if v or w are saturated, there are unsaturated nodes to perform a neighbor switch and make v, w unsaturated. The neighbor switch preserves the current JDM while it frees one stub for the node of interest, so we can then add the edge (v, w). □

Lemma 2. *If* $JDM(k,l) < JDM^{\odot}(k,l)$*, then an edge can be added between* V_k *and* V_l .

Proof. Let us assume that it is not possible to add a new edge between nodes in V_k and V_l. This means that nodes in V_k and V_l and current edges build a complete bipartite graph (or complete graph if $k = l$), *i.e.,* all $|V_k| * |V_l|$ possible edges between V_k and V_l have been added:

$$JDM(k,l) = JDM_{max}(k,l) = \begin{cases} |V_k| * |V_l|, & \text{if } k \neq l \\ |V_k| * (|V_k| - 1)/2, & \text{if } k = l \end{cases}$$

Since the target $JDM^{\odot}$ is assumed realizable, $JDM^{\odot}(k,l) \leq JDM_{max}(k,l)$, $\forall (k,l)$. This contradicts the assumption $JDM(k,l) < JDM^{\odot}(k,l)$. □

Lemma 3. *If* $JDM(k,l) < JDM^{\odot}(k,l)$*, there is at least one unsaturated node* x_k *of degree* k *and one unsaturated node* x_l *of degree* l.

Proof . Let us assume that there is no unsaturated node x_k of degree k. This means that every node $x \in V_k$ has k connected stubs and zero free stubs. This can happen 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).

Therefore, assuming that there is no unsaturated node x_k leads to contradiction. The same argument applies to assuming no unsaturated node x_l of degree l. □

Lemma 4. *It is always possible to perform a neighbor switch for a saturated node* w *with an unsaturated node* w' *of the same degree,* i.e., $deg(w) = deg(w')$.

Proof. The number of used stubs at node w is larger than the number of used stubs at node w', since $deg(w) = deg(w')$, w is saturated and w' is unsaturated. This means that there always exists a node t, which is connected to w but not connected to w'. Therefore, it is always possible to remove edge (w, t) and add edge (w', t).

Since node w' is unsaturated, we can use one of its free stubs. Adding edge (w', t) does not create multi-edges because node t was not previously connected to w'. Removing edge(w, t) frees one stub for node w. The value of $JDM(deg(w), deg(t))$ will not change, before and after the switch, since the number of edges between nodes with $deg(w)$ and $deg(t)$ remain the same. □

C. Running Time

Lemma 5. *The running time of* `2K_Simple` *is linear in the number of edges,* i.e., *is* $O(|E| * d_{max})$.

Proof. Theorem 1 proves that we add exactly one edge in line 8 of the 2K_Simple algorithm. The algorithm terminates after adding $|E|$ edges, by doing $|E| = \sum_{k,l} JDM^{\odot}(k,l)/2$ iterations over lines 3-9.

In Line 3 we have to find a pair of disconnected nodes (v, w) between a degree pair (k, l). When we start with an empty graph, we can find each pair of disconnected nodes in constant time, by iterating over the first $JDM^{\odot}(k,l)$ node pairs formed between nodes in degree group k and degree group l. Lines 4-5, and 6-7 contain two neighbor switches, that may or may not be needed. It takes constant time to determine whether a neighbor switch is required (Lines 4,6). If a neighbor switch for a node w of degree $k = deg(w)$ is needed: (i) it takes constant time to find an unsaturated node w' of degree k if we keep track of unsaturated nodes (ii) it takes $O(k)$ time to find a neighbor t of w not connected to w' since in the worst case we need to look up to $k-1$ nodes (because there are only k stubs) (iii) it takes constant time to remove edge (w, t) and add edge (w', t). Therefore, a neighbor switch for a node of degree k takes time $O(k)$. Finally, lines 8-9 of the algorithm take constant time. In summary, the running time of a single iteration (lines 3-9) is $O(k + l) = O(d_{max})$, where d_{max} is the maximum degree in the graph, and is dominated by the neighbor switches. Therefore, the running time of 2K_Simple is approximately linear in the number of edges, *i.e.,* is $O(|E| * d_{max})$. [3] □

V. 2K+ Construction: JDM and Beyond

A key advantage of 2K_Simple compared to the only previous approach that builds a simple graph with a $JDM^{\odot}$, 2K_BDI [18], is its increased flexibility in adding the edges, which allows the algorithm to produce a larger set of simple graphs (possibly all) with the same $JDM^{\odot}$. In particular, 2K_Simple can visit any order of degree pairs and any order of node pairs (v,w), even before completing a degree pair; it can start with an empty or partially built graph, as long as $JDM(k,l) \leq JDM^{\odot}(k,l)$ for every $(k,l) \in JDM^{\odot}$. In this section, we exploit this flexibility and we extend 2K_Simple to impose additional properties, in addition to $JDM^{\odot}$. By controlling the order in which edges are added, we can construct different graphs, all with the same $JDM^{\odot}$. In Section V-A we control (approximately) the average clustering $\bar{c}$, and in Section V-B we impose (exactly) node attributes.

A. JDM and Clustering

Approach. We can control the order of node pairs that we consider when adding edges in the graph (Algorithm 1, lines 3 and 8), and we choose to do that based on the following process. 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 $dist(v,w) = \min(|r_v - r_w|, 1 - |r_v - r_w|)$. Let E' be a list of all possible node pairs $\{v, w\}$ in a given order. We use the order in the list to decide which edges to add first: 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 before (v_j, w_j). The *sortedness*[4] of a list E' is defined as:

$$\text{sortedness(E')} = 1 - \frac{\text{number of inversions in list E'}}{|E'|(|E'|-1)/2} \in [0,1]$$

where, a set of two node pairs E'_i and E'_j is inverted iff

$$(i < j) \;\; and \;\; dist(v_i, w_i) > dist(v_j, w_j).$$

We introduce parameter S to control the sortedness of E'. 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. Fig. 3 shows an example of this correlation for three types of graph inputs: two graph models and one Facebook network are used to generate the graphs. 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. [5] In summary, intuition and experimental results show that by controlling the sortedness S (determined by the order of E'), we can control the amount of clustering, in addition to $JDM^{\odot}$.

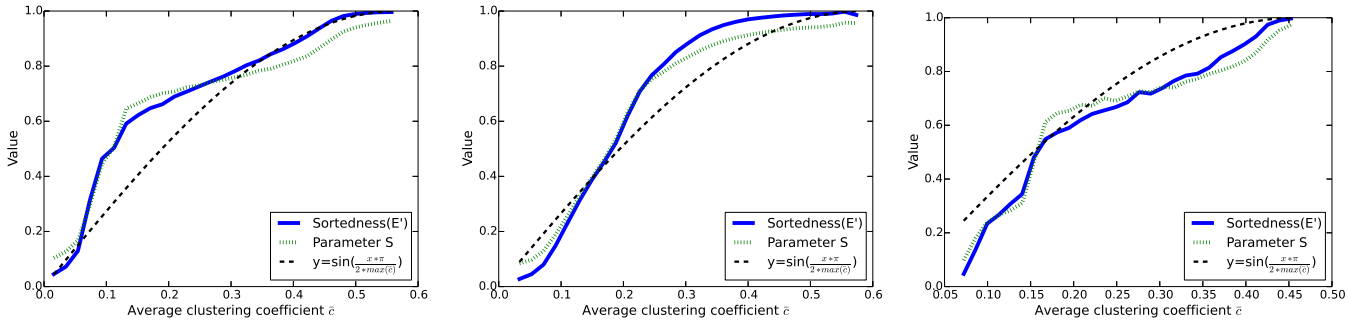
Algorithm. Our algorithm for achieving exactly $JDM^{\odot}$ and approximate clustering is summarized next. In the first stage, it attempts to add edges using a given order E' of all possible node pairs. Similarly to 2K_Simple, it only adds edges if the current $JDM(k,l)$ value does not exceed the target $JDM^{\odot}(k,l)$. 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 ($k < k^{\odot}$ and $l < l^{\odot}$). Therefore at the end of the first stage, despite considering all possible node pairs, there might be some unsaturated nodes, 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 $JDM^{\odot}$.

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 $List$, depending on the target $\bar{c}^{\odot}$. The function $reversed(List)$ returns a reversed list of the input $List$ and the function $partialSort(List, Percent)$ returns a list which has approximately $1-Percent\%$ inverted node pairs. The intuition is that $Percent$ controls the fraction of node pairs that are

[3] A tighter upper bound can be obtained by counting the running time of a neighbor switch 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: $|E| = \sum_k D_k * k/2$. In the worst case that each of these stubs will need a neighbor switch during an edge addition, the running time would be $O(\sum_k D_k * k * (k-1)/2) = O(\sum_k D_k \binom{k}{2})$, which is the number of paths of length two in the graph.

[4] Here are some examples for the sortedness of list E': (i) $sortedness(E') = 1$ when there are no inversions; the list is sorted in increasing distance; (ii) $sortedness(E') = 0.5$ when there are half possible inversions; the list is randomly shuffled; (iii) $sortedness(E') = 0$ when there are all possible inversions; the list is sorted in decreasing distance.

[5] The latter observation is intuitive and has also been made in our prior work [9]: closest nodes have been selected first to connect with an edge, and this resulted in many triangles among "nearby" nodes in the coordinate system. The goal of [9] was to achieve the highest possible clustering, while our goal is to control the degree of clustering spanning the whole range.



(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. 3. We use two different graph models (Erdős-Rényi, Powerlaw-Cluster [10]) to generate two graph instances and we select one real-world network, Facebook Caltech [19]. 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.

Algorithm 2: `2K_Simple_Clustering`

Input: $JDM^{\odot}$

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: $k = deg(v)$, $l = deg(w)$, $k^{\odot} = deg(v)^{\odot}$, $l^{\odot} = deg(w)^{\odot}$

5: **if** JDM$(k,l) <$ JDM$^{\odot}(k,l)$ **and** $k < k^{\odot}$ **and** $l < l^{\odot}$ **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`

order(List, S):

1: **if** $S <= 0.5$

2: **return** $reversed(\ partialSort(List, 1 - 2 * S)\ )$

3: **else**

4: **return** $partialSort(List, 2 * S - 1)$

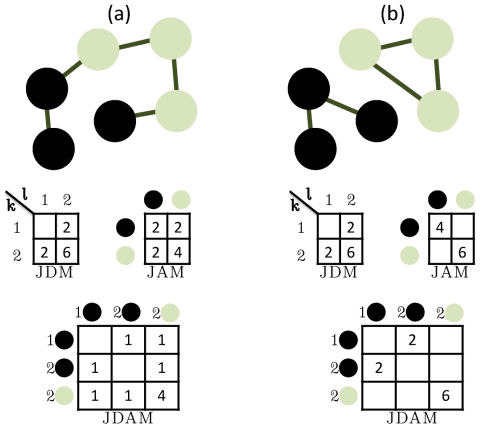


Fig. 4. Example of two different graphs, (a) and (b), with the same Joint Degree Matrix (JDM) and different Joint occurrence of Attributes Matrix (JAM), thus different JDAMs. We assign the color black to nodes with the first attribute and green to nodes with the second attribute.

ordered (thus, if connected with an edge, will lead to maximum clustering) and the remaining node pairs that are picked at random (thus, if connected with an edge, will lead to minimum clustering). Therefore, this determines the sortedness S, based on the target clustering. Further implementation details, of this admittedly heuristic approach, are omitted due to lack of space.

Fig. 3 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_Simple_Clustering` 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, 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 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.

B. 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

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 disconnected nodes $v \in V_{ki}$ and $w \in V_{lj}$
4: **if** v does not have free stubs
5: NeighborSwitch(v)
6: if w does not have free stubs
7: NeighborSwitch(w)
8: add edge between (v, w)
9: $JDAM((k,i),(l,j))$ ++ ; $JDAM((l,j),(k,i))$ ++
Output: simple graph with $JDAM = JDAM^{\odot}$

better characterizes the graph [7,11,16]. 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.

$$\text{JAM}(i,j) = \sum_{v \in A_i} \sum_{w \in A_j} 1_{\{\{v,w\} \in E\}}. \quad (4)$$

However, JAM alone does not capture the network structure. In Fig. 4, we show a toy example. The graphs in Fig. 4(a) and Fig. 4(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 $B_{ki} = \{v | v \in V_k, v \in A_i\}$ be the set of nodes that have degree k and attribute i. The number of degree-attribute groups is at most $d_{max} * p$. We define the *Joint Degree and occurrence of Attributes Matrix* (JDAM) as the number of edges connecting nodes in B_{ki} with nodes in B_{lj} for degree-attribute groups (k,i) and (l,j).

$$\text{JDAM}((k,i),(l,j)) = \sum_{v \in B_{ki}} \sum_{w \in B_{lj}} 1_{\{\{v,w\} \in E\}}. \quad (5)$$

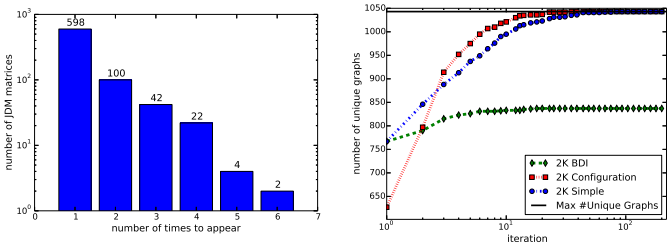
Example JDAMs are shown in Fig. 4. A JDAM is similar to JDM, but each row now describes not only a degree k but a degree-attribute pair (k,i); and similarly for the columns.

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. The same arguments for correctness, termination and running time, as in Section IV, apply here as well.

VI. Simulations

A. Space of Constructed 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`



(a) Frequency of JDM matrices.

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

Fig. 5. Generation of all non-isomorphic graph instances with 7 nodes.

(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.

1) 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` [6] to generate all 1044 non-isomorphic graph instances that contain seven nodes[6]. 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. 5(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. 5(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 III.

2) 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_Simple_Clustering` explores the space of graphs much faster than existing 2K construction algorithms.

We select the Holme and Kim algorithm [10], implemented in networkx [6] by function powerlaw_cluster, to generate

[6]We should note that the number of such graph instances increases exponentially with size e.g. for n=24 it is $\sim 1.95 \times 10^{59}$ [1]. For this reason we use in our experiment a small size of n=7 that gives 1044 instances.

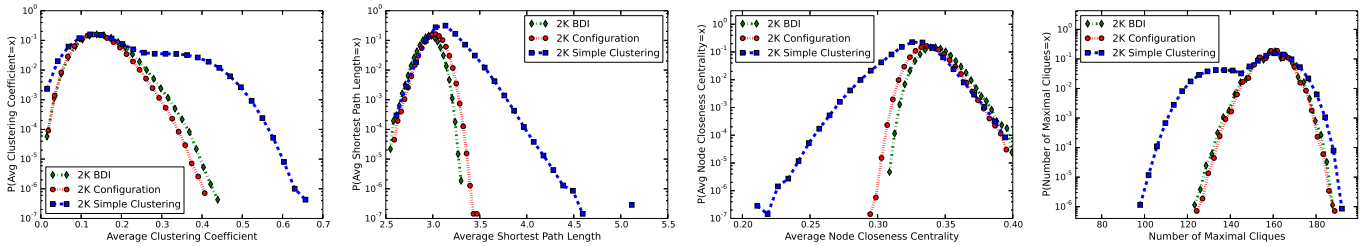


Fig. 6. We use the Holme-Kim algorithm [10] to produce 100 graphs, each with exactly 100 nodes, 196 edges and a range of $\bar{c}$ values. We then calculate the JDM matrix for each produced graph instance and set it as the JDM$^{\odot}$ for each 2K construction algorithm. Overall we generate 10^7 graph instances per construction algorithm and empirically calculate the pdf for each graph property.

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_Simple_Clustering`, `2K_BDI`, and `2K_Configuration`. We produce a total of $3 * 100 * 10^5 = 3 * 10^7$ 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 spreading from v, and (iv) the total number of maximal cliques $\sum C_i$, where C_i is the frequency 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.

Fig. 6 shows the empirical probability density for each of the considered properties. `2K_Simple_Clustering` covers a significantly larger range for all properties when compared to `2K_Configuration` and `2K_BDI`. *e.g.,* `2K_Simple_Clustering` 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` $[0, 0.44]$.

B. Evaluation on Real-World Graphs

Next, 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 Facebook university networks that come with several annotated node attributes. The second group consists of larger graphs without node attributes.

For each real topology, we construct several synthetic graphs with different algorithms: `2K_Simple`, `2K_Simple_Clustering` with two different values of the clustering parameter S, and `2K_Simple_Attributes`

Dataset	$\|V\|$	$\|E\|$	Avg Deg.	$\bar{c}$	# dorms	# years
FB: Rice [19]	4 087	184 828	90.45	0.294	10	22
FB: Princeton [19]	6 596	293 320	88.94	0.237	57	27
FB: UCSD [19]	14 948	443 221	59.30	0.227	40	23
FB:New Orl. [20]	63 392	816 884	25.77	0.222	-	-
amazon0601 [2]	403 364	2 443 309	12.11	0.42	-	-
youtube-links [13]	1 134 894	2 987 623	5.26	0.081	-	-

TABLE I
REAL-LIFE TOPOLOGIES USED FOR EVALUATION.

Topology	Graph Model	Graph properties						Generat. Time (sec)
		Avg Node Value			Number of	Assortativ.		
		$\bar{c}$	Sh.P.	Closn	Cliques	Dorms	Year	
Rice	original	0.29	2.45	0.41	1.1M	0.42	0.28	-
	2K Simple	0.07	2.37	0.43	480K	0.01	0.01	5
	2K+S=0.70	0.29	2.65	0.38	4.1M	0.01	0.01	14
	2K+S=1.0	0.53	2.74	0.37	12.9M	0.01	0.01	31
	2K+dorms	0.13	2.38	0.42	797K	0.42	0.09	10
	2K+year	0.09	2.37	0.43	500K	0.08	0.28	11
Princeton	original	0.24	2.67	0.37	1.3M	0.09	0.45	-
	2K Simple	0.05	2.54	0.40	598K	0.00	0.01	7
	2K+S=0.61	0.24	2.85	0.35	5.2M	0.00	0.01	23
	2K+S=1.0	0.56	2.96	0.34	33.6M	0.00	0.01	48
	2K+dorms	0.10	2.55	0.40	751K	0.09	0.27	19
	2K+year	0.08	2.59	0.39	755K	0.05	0.45	17
UCSD	original	0.23	3.03	0.34	743K	0.25	0.36	-
	2K Simple	0.03	2.98	0.34	500K	0.00	0.01	9
	2K+S=0.54	0.23	3.32	0.30	3.1M	0.00	0.01	46
	2K+S=1.0	0.63	3.36	0.30	4.1M	0.00	0.01	53
	2K+dorms	0.03	2.88	0.35	524K	0.25	0.05	50
	2K+year	0.02	2.89	0.35	476K	0.02	0.36	50
FB: New Orl.	original	0.22	4.33	0.24	1.5M	-	-	-
	2K Simple	0.02	7.06	0.15	806K	-	-	28
	2K+S=0.6	0.30	4.46	0.23	1.5M	-	-	82
	2K+S=1.0	0.58	4.35	0.23	1.0M	-	-	88
amazon0601	original	0.42	6.35	0.16	1.0M	-	-	-
	2K Simple	0.05	13.9	0.01	2.3M	-	-	181
	2K+S=0.6	0.47	5.61	0.18	1.1M	-	-	254
	2K+S=1.0	0.60	5.71	0.18	666K	-	-	291
youtube-links	original	0.08	5.23	0.19	3.3M	-	-	-
	2K Simple	0.02	22.0	0.00	2.9M	-	-	1582
	2K+S=0.6	0.11	4.76	0.21	2.5M	-	-	3944
	2K+S=1.0	0.21	4.83	0.20	3.0M	-	-	4724

TABLE II
GRAPH PROPERTIES OF GENERATED GRAPHS FOR EACH TOPOLOGY AND GRAPH MODEL AVERAGED OVER 20 RUNS. LIST OF GRAPH PROPERTIES (FROM LEFT TO RIGHT): (I) AVERAGE CLUSTERING COEFFICIENT $\bar{c}$ (II) AVERAGE SHORTEST PATH LENGTH (III) AVERAGE NODE CLOSENESS (IV) NUMBER OF MAXIMAL CLIQUES (V-VI) ASSORTATIVITY FOR NODE ATTRIBUTES "DORMS" AND "YEAR". LAST COLUMN CONTAINS THE GRAPH GENERATION TIME IN SECONDS.

for node properties "dormitories" and "year of admission". Table II reports the properties of these graphs, both targeted ($\bar{c}$ and the assortativity coefficient for dorms, and year) and non-targeted (average shortest path length, average closeness, number of maximal cliques), averaged over all generated graph instances. The last column reports the graph generation time in seconds. The algorithms were implemented in Python and executed on an AMD Opteron 2.4Ghz machine.

Accuracy. 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, when targeting $\bar{c}$ we do not observe any significant benefits for those two non-targeted properties. Neither do we observe any effect on the assortativity of node attributes which stays close to zero. However, we notice that 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 0.05 to 0.10 and assortativity for the attribute "Year" from 0.01 to 0.27 when compared to `2K_Simple`. In the three large 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 0.01, whereas $2K + S = 0.6$ achieves $S = 0.18$ which is very close to the real value of 0.16. Finally, for both small and big graphs the property "Number of Cliques" does not consistently improve by targeting either $\bar{c}$ or attributes.

Efficiency. The time needed to generate a graph using either of our three algorithms does not differ significantly; this was expected since they all have approximately linear time in $|E|$. For example, it takes tens of seconds to generate synthetic graphs for all Facebook topologies of Table I ($4K - 63K$ nodes) even when we target maximum clustering (model $2K$+$S = 1.0$). To put those numbers in context, we targeted $2K + \bar{c}$ with an MCMC approach using double edge swaps, which is the previous state-of-the-art. With MCMC, it took approximately *a day* to generate synthetic graphs with the same clustering as the graph input, for the smallest topologies Rice and Princeton; simulations for the bigger graphs did not finish after several days. In contrast, using our methods it took approximately *an hour* to generate synthetic graphs with similar or higher clustering than the largest topology (YouTube-links, 1.13M nodes). In summary, in real-world graphs with high clustering our approach can reduce the running time by orders of magnitudes compared to methods that rely on $MCMC$.

VII. Conclusion

Our goal in this work was to generate realistic synthetic graphs, primarily motivated by -but not limited to- online social networks. We designed a class of algorithms, which we refer to as "2K+", to construct simple graphs with a desired JDM and possibly additional properties. First, we presented `2K_Simple`, an algorithm that provably constructs simple graphs with an exact $JDM^{\odot}$, in linear time in the number of edges. We show that `2K_Simple` has increased flexibility in selecting edges compared to prior approaches, which we then exploit to impose additional structure on top of the JDM, still in approximately linear time. `2K_Simple_Clustering` targets exact $JDM^{\odot}$ *and* average clustering $\bar{c}^{\odot}$. `2K_Simple_Attributes` guarantees 2K *and* joint occurrence of node attribute pairs, together captured in $JDAM^{\odot}$. Both clustering and node attributes are of essential importance for social network graphs, beyond just JDM. We use simulations to generate graphs that resemble real-world social networks, and we demonstrate the benefits of our algorithms in terms of accuracy (how close we are to targeted and non-targeted properties) and speed (the linear running time of our approach is order of magnitude faster in practice compared to MCMC). We believe that our approach is a step forward, towards efficient graph construction, which remains an open problem beyond 2K.

Acknowledgments

This work was supported by NSF Award 1028394 and Marie Curie CIG Grant 321802. We would like to thank Rasmus Pagh, at IT University (ITU) of Copenhagen, Denmark, for insightful discussions on this problem. Athina Markopoulou was on sabbatical, and Balint Tillman was a student, at ITU, for part of the time this research was conducted. All authors are currently affiliated with CPCC and CalIT2 at UC Irvine.

References

[1] Simple Graph: http://mathworld.wolfram.com/SimpleGraph.html.
[2] SNAP Graph Library: http://snap.stanford.edu/data/.
[3] W. Aiello, F. Chung, and L. Lu. A random graph model for massive graphs. In *Proc. of STOC*, 2000.
[4] S. Bansal, S. Khandelwal, and L. Meyers. Exploring biological network structure with clustered random networks. *BMC Bioinformatics*, 2009.
[5] 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*, 5(4):e10012, 2010.
[6] N. Developers. Networkx. *networkx. lanl. gov*, 2010.
[7] X. Dimitropoulos, D. Krioukov, A. Vahdat, and G. Riley. Graph annotations in modeling complex network topologies. *TOMACS*, 2009.
[8] S. Dorogovtsev. Networks with desired correlations. *arXiv 0308336 '03*.
[9] M. Gjoka, M. Kurant, and A. Markopoulou. 2.5K-Graphs: from Sampling to Generation. In *Proc. of IEEE INFOCOM*, 2013.
[10] P. Holme and B. J. Kim. Growing scale-free networks with tunable clustering. *Physical review E*, 65(2):026107, 2002.
[11] 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*, 24(3), 2008.
[12] P. Mahadevan, D. Krioukov, K. Fall, and A. Vahdat. Systematic topology analysis and generation using degree correlations. In *SIGCOMM*, 2006.
[13] A. Mislove, M. Marcon, K. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *IMC*, 2007.
[14] M. Molloy and B. Reed. A critical point for random graphs with a given degree sequence. *Random structures and algorithms*, 1995.
[15] M. Newman. Random graphs with clustering. *Phys. review letters*, 2009.
[16] 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.
[17] M. Serrano and M. Boguñá. Tuning clustering in random networks with arbitrary degree distributions. *Physical Review E*, Sept. 2005.
[18] I. Stanton and A. Pinar. Constructing and sampling graphs with a prescribed joint degree distribution using markov chains. *JEA*, 2011.
[19] A. Traud, P. Mucha, and M. Porter. Social Structure of Facebook Networks. *Arxiv preprint arXiv:1102.2166*, 2011.
[20] B. Viswanath, A. Mislove, M. Cha, and K. Gummadi. On the evolution of user interaction in facebook. In *Proc. WOSN*, 2009.
[21] Y. Amanatidis and B. Green and M. Mihail. Graphic realizations of joint-degree matrices. *Unpublished manuscript*, 2008.