THE HYPERTYPE MODEL FOR CLUSTERING IN NETWORKS

Huandong Chang
Chuming Chen

Sponsored by Nicole Eikmeier
June 16, 2021

In the field of network analysis, incorporating *higher-order features* into network models has become increasingly routine. In this paper we introduce the HyperType network model: an extension of a simple typing model with better clustering due to the focus on triangles instead of single edges. In addition to more realistic clustering, we empirically show HyperType retains many features from the original typing model. We empirically fit HyperType to real data, and show an interesting relationship to a recursive Kronecker product.

1 INTRODUCTION

Network analysis is a rich and interdisciplinary field with applications in many domains including Biology [Barabasi and Oltvai, 2004], Social Networks [Hobson and DeDeo, 2015; Foucault Welles et al., 2010], and Business [Abebe et al., 2018]. Modeling networks is a vital research area since real data is expensive and limited. Models can be used to test the performance of algorithms and predict how these algorithms behave on real (large) data. Developing models that obey properties which occur naturally can provide more realistic studies when using synthetic data in place of real data. On the flip-side, studying the properties of the data generated by models may lead to insights into the structure of the real data [Eikmeier and Gleich, 2019a].

There exist already many popular graph models. The Erdős Rényi graph model [Erdös and Alfréd, 1959] is a simple model, but does not display many real-world properties. Albert-László Barabási and Réka Albert addressed the problems in Erdős Rényi graph model and proposed Barabási–Albert model [Barabási and Albert, 1999] that tries to explain the power-law degree distribution as shown in some real networks. The Stochastic block [Holland et al., 1983] and the Kronecker [Kolda et al., 2014] models are designed to capture community structure, while the Chung-Lu graph model [Aiello et al., 2000] constructs a graph according to a prescribed degree distribution.

Recently, attention has shifted towards *higher-order* analysis [Grilli et al., 2017; Xu et al., 2016; Yin et al., 2018] - a way of incorporating and considering more complex structure in networks [Benson et al., 2016]. In particular, there has been recent work in incorporating higher-order structure directly into network models [Eikmeier and Gleich, 2019b,a; Scholtes, 2017; Chodrow, 2020].

In a similar vein to many of these efforts, we propose the HyperType model, an extension of the typing model [Akoglu and Faloutsos, 2009] (formally introduced in section 3). Typing model has a broad applications, such as in the field of detecting illicit behaviors [Savage, 2017] and outlier detection [Ranshous et al., 2016]. In section 4, we show that the HyperType model keeps most of the properties that the original typing model obeys, such as a power-law degree distribution and strong community structure. We further show that the HyperType model exhibits non-trivial clustering coefficients absent from the original typing model.

Huandong Chang, Grinnell College
huandongchang@hotmail.com
Chuming Chen, Grinnell College
chumingchen06@gmail.com
Nicole Eikmeier, Grinnell College
eikmeier@grinnell.edu

HC and CC contributed equally to this research.

We will elaborate on how we fit our model to real networks and give two specific examples in section 5.

Finally, we consider the relationship between an evolving typing model with a Kronecker product [Leskovec et al., 2010] in section 6. This gives us the advantages of having a certain number of vertices and edges while keeping the properties of the HyperType model. While not necessarily useful for implementation or features, it is a fun mathematical connection between these two ideas. We provide all of our code for reproducibility at `https://github.com/ccming1006/Typing-Model`.

2 PRELIMINARIES: 2D TYPING MODEL

The 2D typing model is proposed by Akoglu and Faloutsos [2009]. It mimics the process of typing on a keyboard with k characters and a space bar. Every key has its own probability, and the space bar is hit with probability q. The sum of the probabilities of these $(k+1)$ keys adds up to 1. A word is formed by randomly typing keys until the space bar is hit. This model is called RTG-IU: Random Typing Generator with Independent Un-equiprobable keys. A sequence of words is divided into pairs by marking words as 'source' and 'destination' alternatingly. An edge is added between the source and destination. Each unique word is a node in the graph. If two connected words are grouped again, the weight of the edge between them will be increased by one.

We can also think of this process as choosing entries in a matrix. As shown in Figure 1a, we have a $(k+1)$ by $(k+1)$ matrix, where $k=2$ in this case. Each time, we choose an entry, and then append the first key of the entry to the source, the second key to the destination. When a space character is appended to a word, the word will be terminated. This process continues until both words are terminated. Notice that using the probability matrix, the probability of choosing any character remains the same.

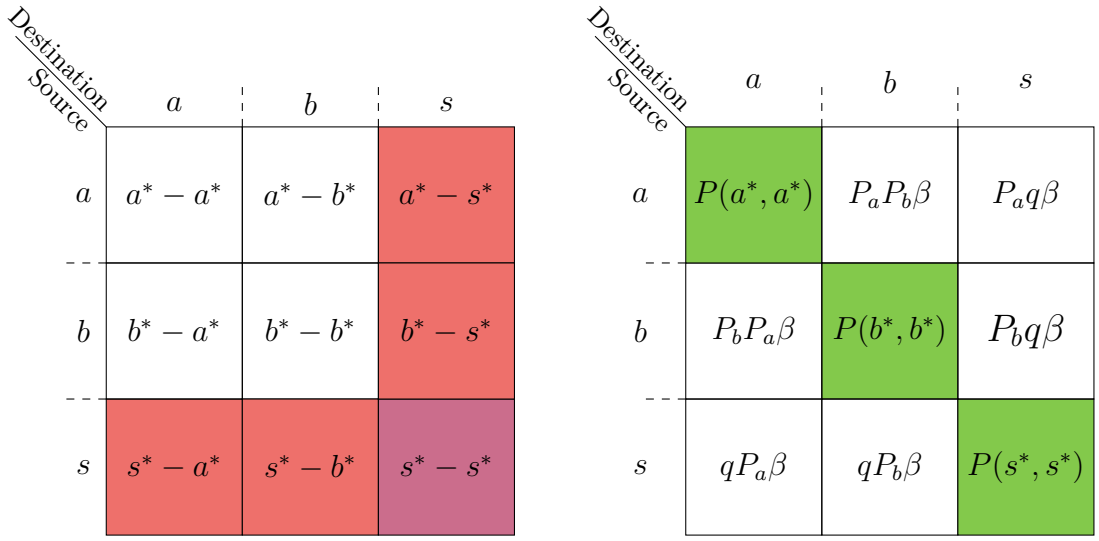


Fig. 1. An example of 2-d keyboard, this figure is adapted from Akoglu and Faloutsos [2009]. 1(a) is the matrix for keys 'a','b', and 's'. If a entry colored red is hit, one word will be terminated; if a entry colored purple is hit, both words are terminated. 1(b) is the matrix after β is introduced. Probability of white entries decreases, and probability of green entries increases. The amount that a green entry increased is the amount that the white entries on its row decreased. For instance, $P(a^*, a^*) = P_a - P_a P_b \beta - P_a q \beta$. Therefore, the overall probability of keys are unchanged.

Akoglu and Faloutsos [2009] also introduced an imbalance factor, β, for the purpose of homophily and communities[1]. This imbalance factor β is a real number between 0 and 1. The idea behind this community building approach is to increase

[1] In real networks, nodes that are similar to each other are more likely to be adjacent. We call this feature the homophily and communities of network.

the probability of a-to-a edges and to decrease the probability of a-to-b edges while maintaining the probability in each row. That is to say, if the entry has the same two keys, we increase its probability; otherwise, we decrease it. For the example in Figure 1a, the adjusted probabilities are given in Figure 1b.

After boosting the probability of the diagonal keys and decreasing the probabilities of the off-diagonal keys, nodes with similar labels will have a higher chance to be connected. In the HyperType model, we will use a similar approach to enhance model homophily.

3 HYPERTYPE MODEL

In the original typing model by Akoglu and Faloutsos [2009] sequences of randomly typed words are divided into groups of two, and two words in each group are adjacent to each other. In our proposed HyperType model, we divide words into triples instead of pairs, and all three words are adjacent to each other in each group. Just as in the original typing model, every unique word is a node in the graph. Therefore, three different words in each group form a triangle, and intuitively, most nodes are involved in at least one triangle. When we add a new edge into the graph, if it already exists, we simply increase the weight of the edge incident to these two words by 1. If the group contains only two unique words, this group becomes an edge with weight 2 in the network. When all three words in one group are the same, we add a node but no edges to the graph. Therefore, if W words are typed, the total weight of the output graph is less than but close to W, since the groups with three unique words have total weight 3 and the groups with two same words have total weight 2.

Table 1 is an example of the typing procedure where we have two keys, a and b, and a space bar in a keyboard. Suppose the randomly typed words are $s, as, bbababs, abs, bs, abs, as, s, abs, \ldots$ Table 1 shows how words are grouped by three, and Figure 2 shows the output graph.

TABLE 1. HyperType Example. Each time we generate three words—Word 1, Word 2, and Word 3—the graph has its total weight increased by 3 unless we have two or more same words in each group, such as $T2$.

Time	Word 1	Word 2	Word 3	Weight
$T1$	s	as	$bbababs$	$1+1+1$
$T2$	abs	bs	abs	$1+1$
$T3$	as	s	abs	$1+1+1$
$T4$	as	aas	bbs	$1+1+1$
$T5$	bas	bs	$aabbs$	$1+1+1$

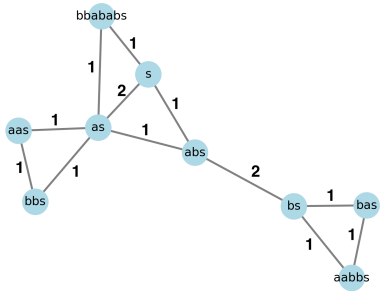


Fig. 2. This is the output graph of Table 1. Notice the edges with weight 2: $bs - abs$ and $s - as$; these two edges appear more than once in one or more groups.

3.1 EQUIPROBABLE KEYS

In this section we assume that we have k non-space keys and the probability of all non-space keys is the same; call it p. Then the probability of the space-bar is $q = (1 - kp)$. The following two lemmas are quick generalizations from Akoglu and Faloutsos [2009]. In both cases, we empirically observe that the lemmas hold for various parameters of the HyperType model. Proofs can be found in the appendix.

Lemma 1. *Let N be the number of unique nodes that are generated in the HyperType model. The expected value of N is*

$$\mathbb{E}[N] \propto W^{-\log_p k},$$

where W is the number of words.

Lemma 2. *Let T be the number of placed triangles in the HyperType model, where a placed triangle is a unique triple with consideration of order and without consideration of repetition. Then, as W grows large, the expected value of T is*

$$T = W^{\alpha}\left(1 + \left(c - c^2 \log_p q\right)n - \frac{n(n^2+2n-1)}{2}c^2\right),$$

for $\alpha = -\log_p k, c = q^{-\log_p k}, n = \log_p W$.

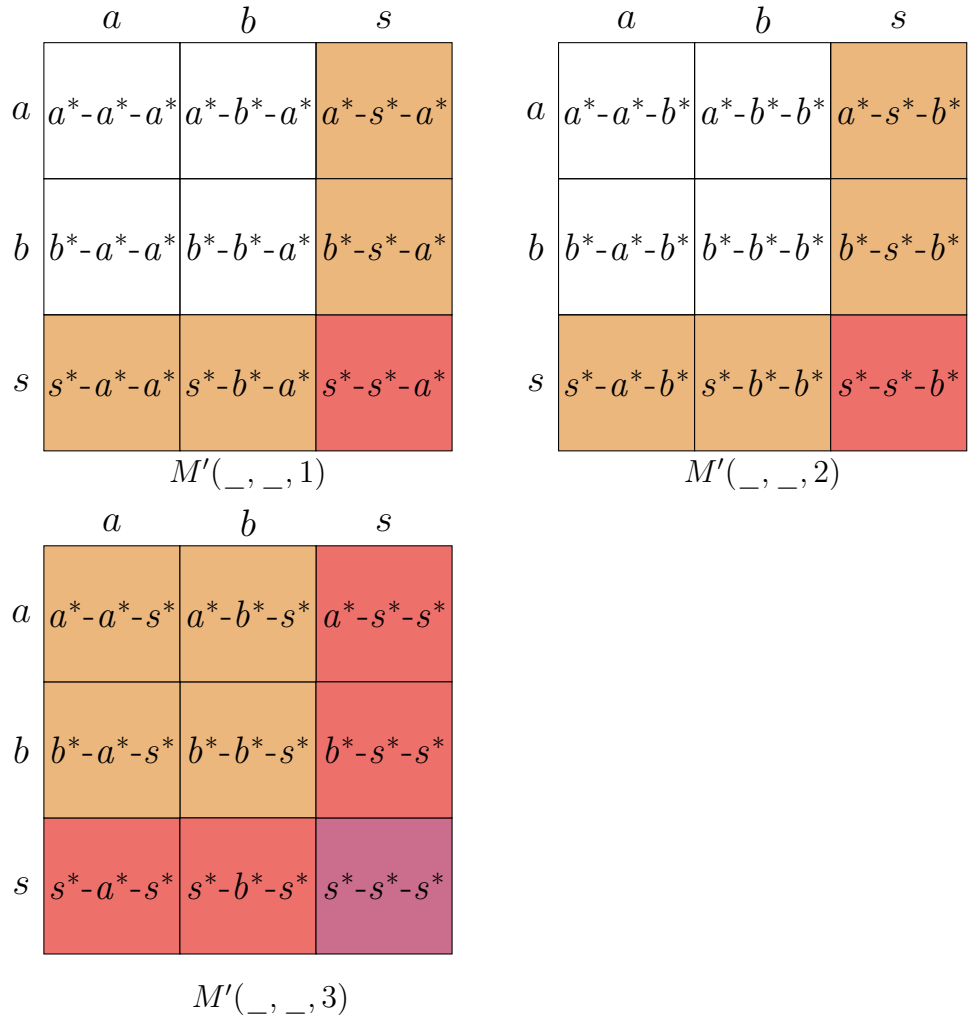
3.2 EXTENSION: UN-EQUIPROBABLE KEYS AND COMMUNITY

With equiprobable keys, the HyperType model is easy to analyze since the only two variables are the number of keys and the probability of the space bar. However, networks generated in this way lack common features of real networks (see section 4.1 and section 5). Therefore, we increase the flexibility of our model by making keys have unequal probabilities. Suppose we have k non-space keys, and we define p_i to be the probability of key i, for all non-space bars. Then $q = 1 - \sum_{i=1}^{k} p_i$ is the probability of the space bar.

Even with Un-Equiprobable keys, this model does not capture the features of homophily and modular structure that exist in real networks. In other words, randomly typing cannot form communities among nodes.

We propose to solve this problem by typing three words—$w_1, w_2,$ and w_3—in each group together, as shown in Figure 3. Our previous description generates three words independently, so we cannot improve the similarities of the three words in each group. However, when we type three words together, we can improve the probability of having two or three same letters in each position of all three words while maintaining the probabilities of hitting each key the same. In other words, for every letter of each word in a group (triple), the probability of having two or three same letters is higher than the theoretical probability in the previous description.

We will first introduce the process of generating three words together without improving the similarities of each triple of words. Suppose we have 2 non-space keys, we can visualize this process by picking one of the 27 entries in the Fig.3 tensor.



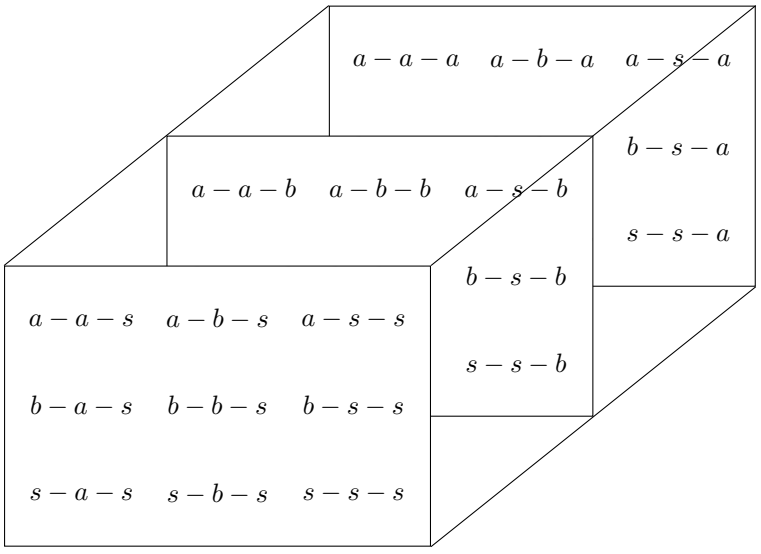


Fig. 3. Tensor for generating three words at the same time. $M(\_, \_, i)$ represents the matrix at the i^{th} slice of the tensor. Notice that slice 3 is the back layer of the tensor and slice 1 is the front layer of the tensor. The recursions for white entries are tensors, orange entries are matrices, red entries are 1-dimensional list, and purple entries are terminated.

The probability of each entry is the product of its corresponding row, column, and slice. For example, the probability of getting a, a, s for the first letter of w_1, w_2, and w_3, respectively, is $p_a * p_a * q$. We then append each of the three letters in the selected entry to the end of w_1, w_2, and w_3, respectively. We recursively repeat this process until each word has hit a space. If a space character is appended to a word, the word is terminated, which means letters will no longer be appended to it. Therefore, when none of the three words is terminated, we choose the letters from a tensor, which returns three letters each time; when one word is terminated, we choose from a matrix, which returns two letters each time; when two words are terminated, we choose letters from an array, which returns one letter each time. In this way, we are only choosing letters for non-terminated words. Fig. 4 is the recursion tensor for entry[2] (1,1,3) in Fig. 3, an example for which none of the three words has terminated. Also, Fig. 1 b) is the recursion matrix of any orange entry in Fig. 3, with only one word terminated.

[2] Notice that we need three indices to represent an entry in a tensor. In this paper, the first index represents the slice, the second represents the x-axis, and the third represents the y-axis.

	a	b	s
a	$aa^*\text{-}aa^*\text{-}aa^*$	$aa^*\text{-}ab^*\text{-}aa^*$	$aa^*\text{-}as^*\text{-}aa^*$
b	$ab^*\text{-}aa^*\text{-}aa^*$	$ab^*\text{-}ab^*\text{-}aa^*$	$ab^*\text{-}as^*\text{-}aa^*$
s	$as^*\text{-}aa^*\text{-}aa^*$	$as^*\text{-}ab^*\text{-}aa^*$	$as^*\text{-}as^*\text{-}aa^*$

$M'(\_, \_, 1)$

	a	b	s
a	$aa^*\text{-}aa^*\text{-}ab^*$	$aa^*\text{-}ab^*\text{-}ab^*$	$aa^*\text{-}as^*\text{-}ab^*$
b	$ab^*\text{-}aa^*\text{-}ab^*$	$ab^*\text{-}ab^*\text{-}ab^*$	$ab^*\text{-}as^*\text{-}ab^*$
s	$as^*\text{-}aa^*\text{-}ab^*$	$as^*\text{-}ab^*\text{-}ab^*$	$as^*\text{-}as^*\text{-}ab^*$

$M'(\_, \_, 2)$

	a	b	s
a	$aa^*\text{-}aa^*\text{-}as^*$	$aa^*\text{-}ab^*\text{-}as^*$	$aa^*\text{-}as^*\text{-}as^*$
b	$ab^*\text{-}aa^*\text{-}as^*$	$ab^*\text{-}ab^*\text{-}as^*$	$ab^*\text{-}as^*\text{-}as^*$
s	$as^*\text{-}aa^*\text{-}as^*$	$as^*\text{-}ab^*\text{-}as^*$	$as^*\text{-}as^*\text{-}as^*$

$M'(\_, \_, 3)$

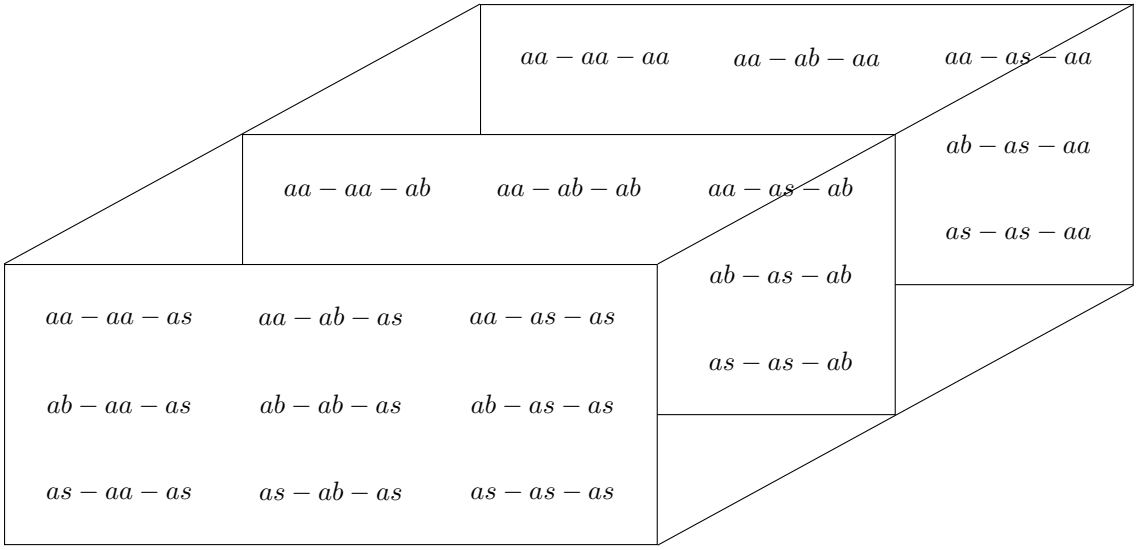


Fig. 4. Recursion Tensor for entry (1,1,3) in Fig. 3, so the first letter of each key in each entry is a. $M'(\_,\_,i)$ represents the matrix at the i^{th} slice of the recursion tensor. Same as Fig. 3, the recursions for white entries are tensors, orange entries are matrix, red entries are 1-dimensional list, and purple entries are terminated.

Similar to the imbalance factor, β, used by Akoglu and Faloutsos [2009], we introduce two imbalance factors β and α in our community algorithm, where β and α are two real numbers between 0 and 1. We classify entries into three types. Type 1 entries are entries with three same keys, i.e. only one letter appears in such entries; for example, (a, a, a). Type 2 entries are entries with two same keys, i.e. two letters appear in such entries; for example, (a, b, a). Type 3 entries are entries with three different keys, i.e. three letters appear in such entries; for example, (b, s, a). In real networks, nodes are more likely to be adjacent to each other if they share some similar properties. In our model, we also want to have this feature. Thus, the goal is to increase the probability of type 1 entries and decrease the probability of type 2 and 3 entries. All type 2 and 3 entries are multiplied by β, and all type 3 entries are multiplied again by α to reduce their probability. In this way, connected nodes will be more similar to each other. Algorithm1 is the pseudo code for the community algorithm. This algorithm is modified based on *Algorithm*1 in **RTG: A Recursive Realistic Graph Generator using Random Typing** by Akoglu and Faloutsos [2009].

Figure 6 shows how the community algorithm applies to one slice of the tensor. Figure 6(a) is the original slice of entries. We color the type 1 entry green, type 2 entries blue, and remain type 3 entries white. Figure 6(b) shows their probabilities before the community algorithm. Figure 6(c) shows the result after the community algorithm is implemented. Notice that the probability of the type 1 entry is q minus probability of the rest eight entries in that slice. Figure 5 is an example of the HyperType graph after we apply the community algorithm.

Importantly, using these imbalance factors in the way we described does not change the probability of each key.

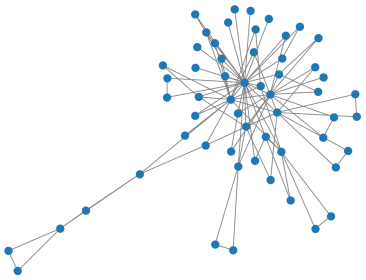


Fig. 5. An example graph generated by the HyperType model (non-space keys= 3, key list = [a,b,c,s], key probabilities = [0.2,0.3,0.2,0.3], and triples = 80). Notice that some edges are not members of any triangles.

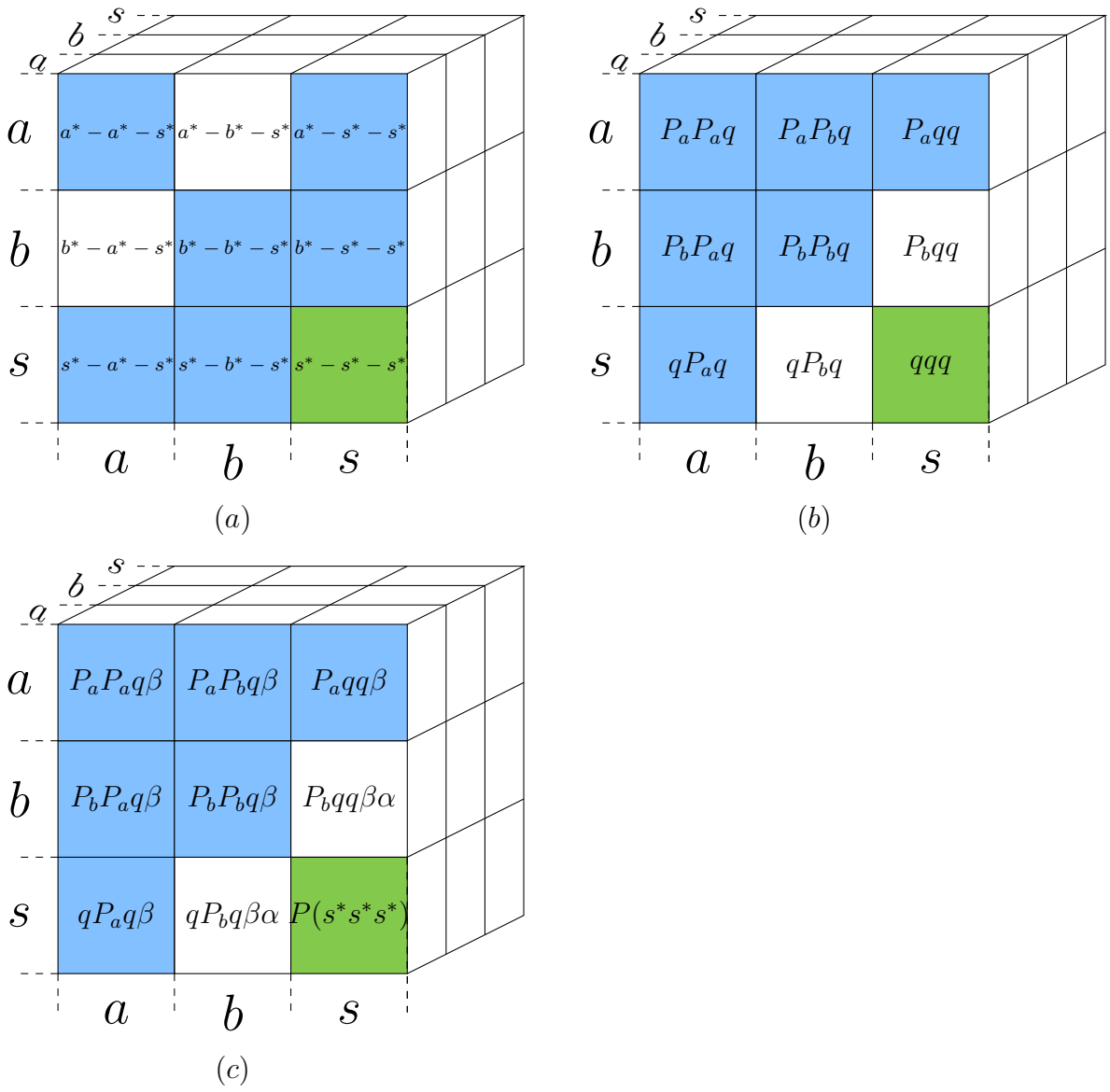
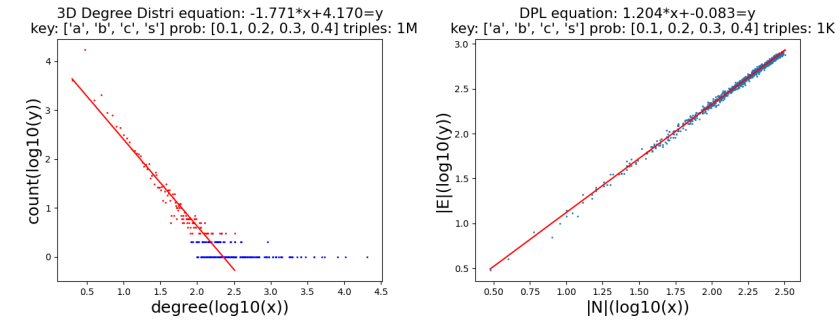


Fig. 6. An example of the community algorithm. Figure (a) shows three types of entries: type 1 in green, type 2 in blue, and type three in white. Figure (b) shows how we calculate the probability of an entry: the product of the probability of all the keys in corresponding to the 3 indices. Figure (c) shows the probability of entries after the community algorithm. The green entry (s, s, s) has probability $P(s^*s^*s^*) = q - P(a^*a^*s^*) - P(a^*b^*s^*) - P(a^*s^*s^*) - P(b^*a^*s^*) - P(b^*b^*s^*) - P(b^*s^*s^*) - P(s^*a^*s^*) - P(s^*b^*s^*)$ which is the sum of its original probability and the amount of probability that other entries on this layer reduced.

Algorithm 1 HyperType Constructor Algorithm

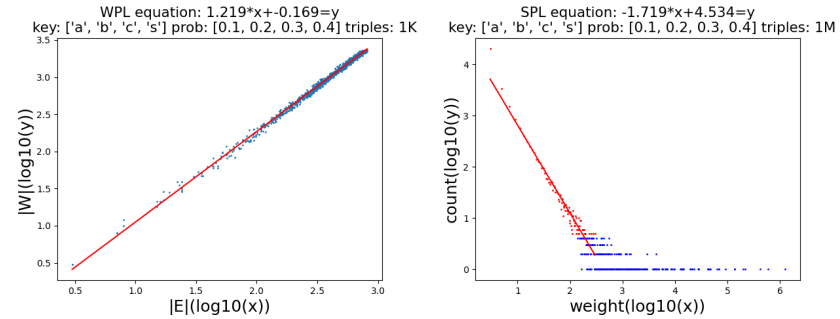
Initialize $(k+1)$ by $(k+1)$ by $(k+1)$ tensor T, a three dimensional array, with cross-product probabilities; $(k+1)$ by $(k+1)$ matrix M, a two dimensional array, with cross-product probabilities; edge list L.
Input: keyList, keyProbList, k, NumberOfWords,α, β
Output: edge-list L for output graph G
for entry e in T **do**
    if e is type 2 entry **then**
        $P(e) = P(e) \cdot \beta$
    if e is type 3 entry **then**
        $P(e) = P(e) \cdot \beta \cdot \alpha$
for $i = 1$ to k **do** ▷ Increase probability of type 1 entry
    $P(i,i,i) = P(i,i,i) + (keyProbList[i] - SumOfLayer[i])$
for 1 to NumberOfWords **do**
    L1, L2, L3 ← SelectNodeLabels T
    Append L1, L2, L3 to L

function SelectNodeLabels(T)(L1, L2, L3)
    Initialize L1, L2, L3 to empty string
    while no word is terminated **do**
        randomly choose an entry $T(i,j,l)$ based on the probability of entries
        if $i \leqslant k$ **then**
            Append character 'i' into L1
        else terminate L1
        if $j \leqslant k$ **then**
            Append character 'j' into L2
        else terminate L2
        if $l \leqslant k$ **then**
            Append character 'l' into L3
        else terminate L3
    while one word is terminated **do**
        randomly choose an entry $M(i,j)$
        if $i \leqslant k$ **then**
            Append character 'i' into first non-terminated word
        else terminate L1
        if $j \leqslant k$ **then**
            Append character 'j' into second non-terminated word
        else terminate L2
    while two words are terminated **do**
        randomly choose an element $keyProbList(i)$
        if $i \leqslant k$ **then**
            Append character 'i' into non-terminated word
        else terminate L1
    return L1,L2,L3



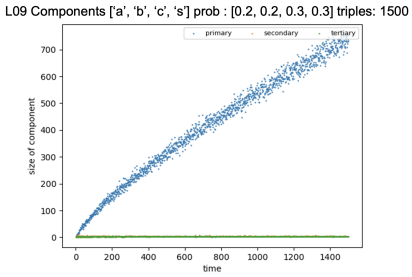
(a) Degree Distribution

(b) Densification Power Law



(c) Weight Power Law

(d) Snapshot Power Law



(e) Components size plot

Fig. 7. Empirical Results of the property tests. (a) shows the logarithmic graph of degree distribution. The red line shows the linear regression of red dots. (b) is the logarithmic graph of number of nodes and the number of edges. (c) shows the log graph of total weight vs number of edges. (d) shows the log graph of number of edges attached to nodes (y-axis) and the total weight of edges attached to nodes (x-axis). (e) shows the plot of the size of the primary, secondary and tertiary components of the graph as it grows.

4 PROPERTIES OF HYPERTYPE MODEL

4.1 LAWS FULFILLED IN 3D RTG-IU WITH COMMUNITIES

In Akoglu and Faloutsos [2009], a list of laws is given that real graphs obey, and we chose some of them to test the reliability of our model. The laws we chose to test are those widely used and frequently tested to ensure the usability of models. One of the most prominent properties they list is that various distributions should follow a *power-law*. A power law distribution has the form

$$y = kx^{\alpha},$$

where x and y are variables of interest, α is the exponent of the power-law, and k is a constant Glen [2017]. We note that there has been disagreement about whether or not power-laws truly exist in real data [Broido and Clauset, 2019; Sala et al., 2010]. Nevertheless, it remains standard practice to create graph models that have power-law distributions. To test whether the various features have significant power-laws, we use the code by Tamás [2017], which uses the method of Clauset et al. [2009]. This method generates a significance parameter, p. We say that the data has a significant power-law if $p > 0.1$. (See [Clauset et al., 2009] for more details of how this works.)

We first consider whether there is power-law relationship in the degree distribution, and between total weight of the edges attached to each node and the number of such edges. To test these first two properties, we generated 20 samples with different parameters. We find there are 15 and 13 samples which passed the power-law test, and the average α for power-law distribution is around 2.0 for both properties. This suggests that the HyperType model is likely to produce a graph with a significant power-law distribution in the degree distribution and the edge weights. See Figure 7 (a) and (d) for a visual. Next, as the model generates more nodes, we should see a power-law relationship between the number of nodes and the number of edges, and between the total weight of nodes and the number of edges. Finally, while the largest component keeps growing in this process, the second and the third largest components tend to remain constant in size with small oscillations. For these laws, we generate graphs shown in Figure 7 (b), (c), and (e). While not robust guarantees, the log-log plots display a linear appearance, which suggests that the properties fit to a power-law distribution or similar.

4.2 CLUSTERING COEFFICIENT

Next, we show empirically that the HyperType model has significant clustering, which does not exist in the original typing model. We will first introduce the average clustering coefficient (ACC) and the global clustering coefficient (GCC). ACC is the average of each node's local clustering coefficient defined as $|K_3(u)|/|W(u)|$, where u is a node, $K_3(u)$ is the number of triangles with u as a node, and $|(W(u)|$ is the number of wedges (triplets) with u as a node. GCC is defined as $3|K_3|/|W|$, where $|K_3|$ is the number of triangles and $|W|$ is the number of wedges (triplets). A wedge is three nodes with two (Figure 8 a) or three edges (Figure 8 b) incident to these three nodes.

In the original typing model, both GCC and ACC are relatively low. Specifically, GCC remains at a very low value (below 0.05) regardless of the parameters, and ACC can only reach 0.4 by using parameters which result in unrealistic graphs. Therefore, compared to other graph models and the original typing model, one important property of our model is its high Average Clustering Coefficient (ACC). However, similar to the original typing model, the GCC of the HyperType model remains at a very low value (below 0.05) for any combination of parameters.

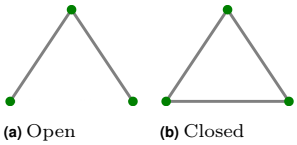


Fig. 8. Two types of wedges (triplets)

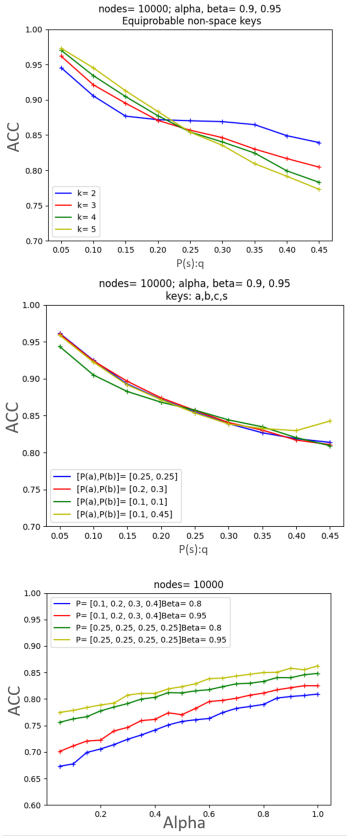


Fig. 9. Plots illustrating how ACC changes with different parameters

Figure 9 illustrates how ACC changes when we have different parameters in our model.

5 FITTING THE HYPERTYPE MODEL TO REAL DATA

In this section we demonstrate that the HyperType model is useful to be fit to real-world data.

Two real-world networks are chosen: *CondMat* (23k Nodes) is a list of co-authors from the ArXiv website who wrote a "Condensed Matter" paper [Leskovec et al., 2007], and *AutonomousSystem(AS)* (11k Nodes) is a graph that represents AS peering information inferred from Oregon route-views, Looking glass data, and Routing registry, all combined [Leskovec et al., 2005].

Since the graphs generated by our model always give a relatively high ACC, we combine the original typing model and the HyperType model to fit the real dataset. We hand-tune the parameters—including probability lists for each original and HyperType model, the number of triples for the HyperType model, and the number of pairs for the original Typing model—to find the best fit for each real dataset. We also use Random Typing Generator (RTG) to fit the real dataset for comparison. We find out that compared to RTG, HyperType requires less generated triples (or pairs) and has more flexibility for the datasets having high ACC or high ratio of edges to nodes. We expect to see similar problems of ACC for other models that are not high-dimensional. Table 2 show the results.

TABLE 2. Parameters for fitting HyperType and Random Typing Generator (RTG) to real data. We measure the number of nodes (Nodes), the number of edges (Edges), the Average Clustering Coefficient (ACC), the Global Clustering Coefficient (GCC), and the size of the largest connected component (lcc) for all dataset, HyperType fit, and RTG fit. The parameters used in the each HyperType and RTG fit are listed in the format of [the number of generated triples (or pairs), probability of key 1, probability of key 2, probability of key 3, probability of the space bar]. In both examples, $\alpha = 0.9$ and $\beta = 0.95$.

Dataset	Nodes	Edges	ACC	GCC	lcc
AS	10900	31180	0.5009	0.03855	10900
HyperType triples: $[110K, 0.1, 0.1, 0.4, 0.4]$ pairs: $[130K, 0.25, 0.25, 0.1, 0.4]$	10365	34511	0.515	0.018	10305
RTG pairs: $[1.1M, 0.15, 0.15, 0.2, 0.5]$	10276	31386	0.329	0.018	10244
CondMat	23133	93497	0.633	0.264	21363
HyperType triples=$600K$ $[0.1, 0.1, 0.4, 0.4]$ pairs=$250K$ $[0.25, 0.25, 0.1, 0.4]$	23337	94041	0.625	0.013	23213
RTG pairs: $[4M, 0.15, 0.15, 0.2, 0.5]$	23029	75974	0.328	0.013	23007

6 TYPING MODEL AND KRONECKER PRODUCT

6.1 KRONECKER PRODUCT

In this section, we consider building and evolving typing model with the Kronecker product. This perspective is inspired from Eikmeier et al. [2018]. The Kronecker product, often denoted by $\otimes$, is an operation on two matrices resulting in a block matrix. Given two matrices A and B, the Kronecker product of those two matrices

is calculated by multiplying every single entry of A by the entire matrix B. For example, if we have matrices $A = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}$ and $B = \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix}$, then [3]

$$A \otimes B = \begin{bmatrix} a_1 \cdot B & a_2 \cdot B \\ a_3 \cdot B & a_4 \cdot B \end{bmatrix} = \begin{bmatrix} a_1b_1 & a_1b_2 & a_2b_1 & a_2b_2 \\ a_1b_3 & a_1b_4 & a_2b_3 & a_2b_4 \\ a_3b_1 & a_3b_2 & a_4b_1 & a_4b_2 \\ a_3b_3 & a_3b_4 & a_4b_3 & a_4b_4 \end{bmatrix}.$$

[3] The dot sign in $a_1 \cdot B$ is to represent scalar multiplication where a_1 is the scalar and B is the matrix.

We observe that the recursive structure of the typing model could be represented by a Kronecker product.

For instance, if there are three keys a, b, s, we can define an initiator matrix $P = \begin{bmatrix} aa & ab & as \\ ba & bb & bs \\ sa & sb & ss \end{bmatrix}$. Each entry of P represents the probability of the edge between two nodes. For example, on $(1, 2)$, we have 'ab', which means on this entry, we can find the probability of an edge between the node 'a' and the node 'b'. For each key, we assign a probability to it, and the probability of an entry is the product of the probability of all the keys the entry has. In our example, the probability on $(1, 2)$ is $P_a \times P_b$.

Currently, P is a 3 by 3 matrix, and there is only one letter for each node. We can conduct the Kronecker product of P by itself to add more letters to each node.

$$P \otimes P = \begin{bmatrix} aa \otimes P & ab \otimes P & as \otimes P \\ ba \otimes P & bb \otimes P & bs \otimes P \\ sa \otimes P & sb \otimes P & ss \otimes P \end{bmatrix} = \begin{bmatrix} aaaa & aaab & aaas & abaa & abab & abas & asaa & asab & asas \\ aaba & aabb & aabs & abba & abbb & abbs & asba & asbb & asbs \\ aasa & aasb & aass & absa & absb & abss & assa & assb & asss \\ baaa & baab & baas & bbaa & bbab & bbas & bsaa & bsab & bsas \\ baba & babb & babs & bbba & bbbb & bbbs & bsba & bsbb & bsbs \\ basa & basb & bass & bbsa & bbsb & bbss & bssa & bssb & bsss \\ saaa & saab & saas & sbaa & sbab & sbas & ssaa & ssab & ssas \\ saba & sabb & sabs & sbba & sbbb & sbbs & ssba & ssbb & ssbs \\ sasa & sasb & sass & sbsa & sbsb & sbss & sssa & sssb & ssss \end{bmatrix}$$

After the Kronecker product, we get $P \otimes P$, a 9 by 9 matrix, which means there are at most 9 nodes in the graph, and each node contains two letters. For example, 'aasa' is the first entry on the second row, and this entry represents the edge between the node 'aa' and the node 'sa'. However, since every word in the typing model needs to end with a space character, we have to append an 's' to 'aa'. Plus, if a word contains 's' in the middle, the typing model will ignore all the letters after the first 's'. Thus, the pair of words become 'aas' and 's'. We can keep calculating the Kronecker product of the current matrix by the initiator to get more nodes as well as more letters on each node. This method is only based on the Kronecker product, but not the Kronecker model. This is because of the feature that letters after the space character will be ignored in the typing model, which means there will be multiple entries correspond to the same pair of nodes. For instance, both 'aasa' and 'aasb' correspond to the pair of nodes 'aas' and 's'. As the model evolves, those entries may represent different pairs again. In the previous example, after appending 'aa' to the end, 'aasaaa' and 'aasbaa' represent to two different pairs. This instability of correspondence makes it impossible to fully implement the typing model with the Kronecker model. Given the probability matrix, we use a ball dropping approach Ramani et al. [2017] to decide which edge to include in the graph. Imagine a rectangle divided into parts where each part corresponds to one entry in P, and the area of each part is proportional to the probability of the entry. Then we randomly drop a ball into the rectangle. According to which part the ball hits, we put the corresponding edge into the graph.

6.2 PRODUCING HYPERTYPE MODEL WITH KRONECKER PRODUCT

We can also construct and evolve the probability tensor for HyperType model with a Kronecker product. To start, instead of building an initiator matrix, we build an initiator tensor, T. For a model with k non-space keys and a space character, T will be a $(k+1) \times (k+1) \times (k+1)$ tensor with three keys on each entry. Entry (i, j, k) will contain the i^{th}, the j^{th}, and the k^{th} letter from the key list. Every time we conduct Kronecker product to the current tensor by initiator, and the size of the current tensor will grow by $(k+1)^3$ times, and the number of keys on each entry will grow by three. Hence, the number of keys on each entry will always be divisible by three. We thus regard the first one-third of letters as word one, the second one-third of letters as word two, and the last one-third of letters as word three. For example, if an entry is 'asbbab', then the three words represented by this entry are 'as', 'bbs', and 'abs'. When an entry is chosen, we put the three words represented by that entry into the graph as a clique of three nodes.

6.3 ADVANTAGES AND LIMITATIONS OF KRONECKER PERSPECTIVE

In a Kronecker typing model, all the edges and nodes come from the probability matrix. That is to say, we have a range of nodes and edges. Therefore, if one wants to build a network with only a certain number of vertices, the Kronecker typing model is a good choice. Furthermore, since the length of words in a Kronecker typing model is bounded above, the resulting network is more interconnected compared with a regular typing model. Notice that if a word contains more letters, then its probability of being connected is significantly lower. Thus, in a regular typing model, we usually see some small node clusters at some corner and never be connected again. However, because of the bounded length, the probability of edges is bounded below as well. As a result, the graph is more interconnected. Nonetheless, having a certain range of nodes can also bring some limitations. For instance, it is not possible to use the Kronecker typing model to simulate a network in which the order increases over time. Also, since the probability of edges is bounded below, the graph will have a higher chance to be connected as more edges are placed. Therefore, the Kronecker typing model is not suitable for simulating networks with many sparse clusters. Because we can only use the Kronecker product, but not the Kronecker model to implement, we can not use properties of the Kronecker model. Compared with the regular Kronecker model raised by Jure et al. [2010], and its 3-dimensional extension HyperKron model designed by Eikmeier et al. [2018], our typing model developed with Kronecker product has more limitations.

7 DISCUSSION

7.1 FUTURE WORK

Our current approach to test if our model is suitable for simulating a real data set is to calculate the global clustering coefficient (GCC) and the average clustering coefficient (ACC). There are indeed more aspects that we can examine to check similarity, such as average degree and degree distribution. Furthermore, similar to the KRONFIT algorithm designed by Jure et al. [2010], an algorithm could be designed to check those aspects and return a quantifiable value indicating similarity. In this paper, we proved two lemmas about expected number of nodes and placed triangles generated by our model. In the future, we would like to give more theoretical results for the properties that we empirically observe. In this paper we only explored directed, weighted networks, however HyperType can easily be extended to undirected and unweighted versions. More exploration of these versions is needed.

7.2 ADVANTAGES

Our HyperType model has the advantages of a) complying with important laws that real networks have, such as power-law relationship in the degree distribution; b) having significant average clustering coefficient (ACC), which the regular typing model Akoglu and Faloutsos [2009] cannot have but some real data would; c) being flexible to adjust the ratio of nodes to edges to fit different real data. For example, for the two real data we test in section 5, **AS** has the ratio of nodes to edges of 2.86 but **CondMat** has 4.04. Our model can simulate both the number of nodes and edges in the real data by tuning parameters, while the Random Typing Generator (RTG) model could not match the high ratio of nodes to edges of **CondMat** dataset even with 0.5 used as its space bar probability.

7.3 LIMITATIONS

Admittedly, there are some limitations of our model. After a series of real word data simulation testing, we found that the ACC and GCC of graphs generated by our model is relatively fixed. Hence, the model is best suited for data with very large ACC and relatively small GCC (see discussion in Section 4.2). Of course there are other models with significant clustering such as those mentioned in the introduction.

Even though we can relate the typing model to a Kronecker product, the typing is more limited than the true Kronecker graph model, due to the role of the space bar (see explanation in Section 6.3).

8 APPENDIX

8.1 LEMMA 1

Let k be the number of keys (not including the space bar) in the HyperType model. Let W be the number of words which are typed (an integer). Let p be the probability of typing one of the keys. That implies the probability of typing the space bar is $q = 1 - kp$.

Lemma 1. *Let N be the number of unique nodes that are generated in the HyperType model. The expected value of $\mathbb{E}[N]$ is*

$$\mathbb{E}[N] \propto W^{-\log_p k},$$

where W is the number of words.

Proof. Let w be a single word generated by the typing model process. That is, w is a sequence of m keys ending with a space bar.

Let N_{c_i} be the number of unique words generated that begin with c_i. Then, the expected number of all uniquely generated words is

$$\mathbb{E}[N] = \mathbb{E}[N_{c_1}] + \mathbb{E}[N_{c_2}] + \cdots + \mathbb{E}[N_{c_k}] + \mathbb{E}[N_S].$$

Since the probability of all of the keys except the space bar is the same, we can simplify the expression to

$$\mathbb{E}[N] = k\mathbb{E}[N_{c_1}] + \mathbb{E}[N_S]$$

Now, words that start with S contribute either 1 unique word or 0 unique words (since the space bar ends the word). The probability that a word starting with a space does not appear is $(1-q)^W$, which is very small. Therefore it is safe to assume that this part of the equation contributes 1 new word with high probability.

$$\mathbb{E}[N] = k\mathbb{E}[N_{c_1}] + 1.$$

Now, we will recurse into $\mathbb{E}[N_{c_1}]$ in a similar manner, as we can write $\mathbb{E}[N_{c_1}] = k\mathbb{E}[N_{c_1c_1}] + \mathbb{E}[N_{c_1S}]$. Here, our notation $N_{c_1c_1}$ means a word that starts with c_1

as the first two characters, and $N_{c_1 S}$ is a word that starts with a c_1 as the first character and a Space as the second character. So,

$$\begin{aligned}\mathbb{E}[N] &= k\mathbb{E}[N_{c_1}] + 1 \\ &= k(k\mathbb{E}[N_{c_1 c_1}] + 1) + 1 \\ &= \vdots \\ &= k^n \mathbb{E}[N_{c_1 \ldots c_1}] + k^{n-1} + k^{n-2} + \ldots + k\end{aligned}$$

where in the first term, c_1 appears n times, representing the length of the longest generated word. Note that this implies that there is a word of length n, hence $\mathbb{E}[N_{c_1 \ldots c_1}] = Wp^n = 1$. We will show momentarily that $n = \log_p(1/W)$ suffices. Now we claim that $\mathbb{E}(N) \propto k^n \mathbb{E}[N_{c_1 \ldots c_1}]$. In order for that to be true, we must have that

$$\frac{\mathbb{E}[N]}{k^n \mathbb{E}[N_{c_1 \ldots c_1}]} = c$$

for some constant c.

$$\begin{aligned}\frac{\mathbb{E}[N]}{k^n \mathbb{E}[N_{c_1 \ldots c_1}]} &= \frac{k^n + k^{n-1} + \ldots + k}{k^n} \\ &= 1 + 1/k + 1/k^2 + \ldots + 1/k^{n-1} \\ &= c,\end{aligned}$$

where the last equality holds since k is fixed, and n is fixed. In particular, to solve for n,

$$\begin{aligned}Wp^n &= 1 \\ p^n &= 1/W \\ n &= \log_p(1/W).\end{aligned}$$

Finally, $\mathbb{E}[N] \propto k^n = k^{\log_p(1/W)} = W^{\log_p(1/k)} = W^{-\log_p(k)}$. □

8.2 LEMMA 2

Lemma 2. *Let T be the number of placed triangles in the HyperType model, where a placed triangle is a unique triple with consideration of order and without consideration of repetition. Then as W grows large, the expected value of T is*

$$T = W^{\alpha}\left(1 + \big(c - c^2 \log_p q\big)n - \frac{n(n^2 + 2n - 1)}{2}c^2\right),$$

for $\alpha = -\log_p k, c = q^{-\log_p k}, n = \log_p W$.

Proof. The number of placed triangles T is the same as the unique number of triples of words. A placed triangle is a unique triple with consideration of order and without consideration of repetitions. For example, $[as, aas, as]$ is a placed triangle, and $[as, bs, cs]$ and $[bs, as, cs]$ are two unique placed triangles. Therefore, we can think of a triple of words as a single word, the generation of which is stopped after the third hit to the space bar. So it always contains two space characters.

Similar to the lemma 2 proof of the typing model [Akoglu and Faloutsos, 2009], we first pull out the first letter of the word (with 2 space bars in this case). In other words, $t : w_1 - s - w_2 - s - w_3$, where w_1, w_2, w_3 are three words without s at the end and s is a space bar. We pull out the first letter, then $t : c_i w_1' - s - w_2 - s - w_3$, where w_1' is the word w_1 without its first letter.

We consider the following two cases:

- If the first letter is not a space bar, we recursively repeat the process here.

· If the first letter is a space bar, the remaining part of the word only contains one space bar. Therefore, we can apply the conclusion from the typing model Lemma 2[4].

For large values of W, $T(W) = k * T(Wp) + E(Wq)$, where $E(Wq)$ is the number of edges of Wq words with a space. From the typing model Lemma 2, we know

$$E(Wq) = (Wq)^{-\log_p k} * (1 + c' \log Wq),$$

where $c' = \frac{q^{-\log_p k}}{-\log p}$ (a fixed number).

Now set

$$\alpha = -\log_p k, c = q^{-\log_p k}, c' = \frac{q^{-\log_p k}}{-\log p}, s = c'c \log q,$$

where

$$E(W) = W^\alpha + c' W^\alpha \log W.$$

We can extend this above equation to

$$\begin{aligned} E(Wq) &= cW^\alpha + c'cW^\alpha \log W + c'c \log qW^\alpha \\ &= cW^\alpha + c'cW^\alpha \log W + sW^\alpha, \end{aligned}$$

$$E(Wqp) = cWp^\alpha + c'cWp^\alpha \log Wp + sWp^\alpha,$$

and

$$E(Wqp^2) = c(Wp^2)^\alpha + c'c(Wp^2)^\alpha \log Wp^2 + s(Wp^2)^\alpha.$$

For some of the following steps, we extract the first letter of all the words, and each word here has two space characters as we described above. For example, in the first step, after we extract the first letter of all the words, the number of placed triangles is the sum of the following two cases:

1) We have $k * T(Wp)$ placed triangles when the first character is not the space bar.

2) We have $E(Wq)$ placed triangles when the first character is the space bar.

Now we have:

$$\begin{aligned} T(W) &= k * T(Wp) + E(Wq) \\ &= k * T(Wp) + cW^\alpha + c'cW^\alpha \log W + sW^\alpha && \text{(expand } E(Wq)\text{)} \\ &= k * \big(k * T(Wp^2) + E(Wqp)\big) + cW^\alpha + c'cW^\alpha \log W + sW^\alpha && \text{(extract the first character from } Wp \text{ words)} \\ &= k * \Big(k * \big(k * T(Wp^3) + E(Wqp^2)\big) + E(Wqp)\Big) + E(Wq) && \text{(extract the first character from } Wp^2 \text{ words)} \\ &= k^n * T(1) + (c+s)W^\alpha \big((kp^\alpha)^{n-1} + (kp^\alpha)^{n-2} + (kp^\alpha)^{n-3} + \ldots + 1\big) \\ &+ c'cW^\alpha \big((kp^\alpha)^{n-1} \log Wp^{n-1} + (kp^\alpha)^{n-2} \log Wp^{n-3} + \ldots + \log W\big) && \text{(recursively extract the first character from words)} \\ &= k^n * T(1) + (c+s)W^\alpha \big((kp^\alpha)^{n-1} + (kp^\alpha)^{n-2} + (kp^\alpha)^{n-3} + \ldots + 1\big) \\ &+ c'cW^\alpha \log Wp\big((n-1)(kp^\alpha)^{n-1} + (n-2)(kp^\alpha)^{n-2} + \ldots + (kp^\alpha)^1\big) \\ &+ c'cW^\alpha \log W. \end{aligned}$$

Since $T(1) = 1, n = \log_p(\frac{1}{W}) = \log_p W$ and $kp^\alpha = kp^{-\log_p k} = 1$,

$$\begin{aligned} T(W) &= k^n + (c+s)W^\alpha n + c'cW^\alpha \log Wp \frac{n(n-1)}{2} + c'cW^\alpha \log W \\ &= W^\alpha (1 + (c+s)n + c'c \log Wp \frac{n(n-1)}{2} + c'c \log W) \\ &= W^\alpha \left(1 + (c+s)n + c'c \log W \frac{n(n+1)}{2} + c'c \log p \frac{n(n-1)}{2}\right). \end{aligned}$$

[4] Although the algorithm for generating edges is different in the typing model and the HyperType model, which means the relationship between E and W in the Lemma 2 of the typing model cannot be applied to the HyperType one, the original proof of Lemma 2 can be applied to all the cases where two words form a pair with a space between them. It's worth noticing that we can only borrow Lemma 2 when calculating the number of triangles in the HyperType model. The number of edges of the HyperType model requires a new proof and is very likely to generate a different result.

From what we have shown above:

$$\alpha = -\log_p k, c = q^{-\log_p k}, c' = \frac{q^{-\log_p k}}{-\log p}, s = c'c\log q, n = \log_p W.$$

Now we need to simplify the above equation:

$$\begin{aligned}
T(W) &= W^\alpha\left(1 + \big(c + c\frac{c}{-\log p}\log q\big)n + \frac{n(n+1)}{2}c\frac{c}{-\log p}\log W + \frac{n(n-1)}{2}c\frac{c}{-\log p}\log p\right) \\
&= W^\alpha\left(1 + \big(c - c^2\log_p q\big)n - \frac{n(n+1)}{2}c^2\log_p W - \frac{n(n-1)}{2}c^2\right) \\
&= W^\alpha\left(1 + \big(c - c^2\log_p q\big)n - \frac{n^2(n+1)}{2}c^2 - \frac{n(n-1)}{2}c^2\right) \\
&= W^\alpha\left(1 + \big(c - c^2\log_p q\big)n - \frac{n(n^2+2n-1)}{2}c^2\right),
\end{aligned}$$

for

$$\alpha = -\log_p k, c = q^{-\log_p k}, n = \log_p W.$$

□

REFERENCES

[Abebe et al., 2018] R. Abebe, L. Adamic, and J. Kleinberg. *Mitigating overexposure in viral marketing in: Proceedings of the 32nd conference on artificial intelligence*. 2018. Cited on page 1.

[Aiello et al., 2000] 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*, p. 171–180. 2000. doi:10.1145/335305.335326. Cited on page 1.

[Akoglu and Faloutsos, 2009] L. Akoglu and C. Faloutsos. *Rtg: a recursive realistic graph generator using random typing*. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 13–28. 2009. Cited on pages 1, 2, 3, 7, 11, 15, and 16.

[Barabási and Albert, 1999] A.-L. Barabási and R. Albert. *Emergence of scaling in random networks*. science, 286 (5439), pp. 509–512, 1999. Cited on page 1.

[Barabasi and Oltvai, 2004] A.-L. Barabasi and Z. N. Oltvai. *Network biology: understanding the cell's functional organization*. Nature reviews genetics, 5 (2), pp. 101–113, 2004. Cited on page 1.

[Benson et al., 2016] A. R. Benson, D. F. Gleich, and J. Leskovec. *Higher-order organization of complex networks*. Science, 353 (6295), pp. 163–166, 2016. arXiv:https://science.sciencemag.org/content/353/6295/163.full.pdf, doi:10.1126/science.aad9029. Cited on page 1.

[Broido and Clauset, 2019] A. D. Broido and A. Clauset. *Scale-free networks are rare*. Nature communications, 10 (1), pp. 1–10, 2019. Cited on page 11.

[Chodrow, 2020] P. S. Chodrow. *Configuration models of random hypergraphs*. Journal of Complex Networks, 8 (3), p. cnaa018, 2020. Cited on page 1.

[Clauset et al., 2009] A. Clauset, C. R. Shalizi, and M. E. Newman. *Power-law distributions in empirical data*. SIAM review, 51 (4), pp. 661–703, 2009. Cited on page 11.

[Eikmeier and Gleich, 2019a] N. Eikmeier and D. F. Gleich. *Classes of preferential attachment and triangle preferential attachment models with power-law spectra*. Journal of Complex Networks, 2019a. Cited on page 1.

[Eikmeier and Gleich, 2019b] ———. *Classes of preferential attachment and triangle preferential attachment models with power-law spectra*. Journal of Complex Networks, 2019b. Cited on page 1.

[Eikmeier et al., 2018] N. Eikmeier, A. S. Ramani, and D. Gleich. *The hyperkron graph model for higher-order features*. In *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 941–946. 2018. Cited on pages 12 and 14.

[Erdös and Alfréd, 1959] P. Erdös and R. Alfréd. *A. rényi,"on random graphs"*. Publicationes Mathematicae, 6, pp. 290–297, 1959. Cited on page 1.

[Foucault Welles et al., 2010] B. Foucault Welles, A. Van Devender, and N. Contractor. *Is a friend a friend? investigating the structure of friendship networks in virtual worlds*. In *CHI'10 Extended Abstracts on Human Factors in Computing Systems*, pp. 4027–4032. 2010. Cited on page 1.

[Glen, 2017] S. Glen. *Power law and power law distribution*. 2017. Cited on page 11.

[Grilli et al., 2017] J. Grilli, G. Barabás, M. J. Michalska-Smith, and S. Allesina. *Higher-order interactions stabilize dynamics in competitive network models*. Nature, 548 (7666), pp. 210–213, 2017. Cited on page 1.

[Hobson and DeDeo, 2015] E. A. Hobson and S. DeDeo. *Social feedback and the emergence of rank in animal society*. PLoS Computational Biology, 11 (9), p. e1004411, 2015. Cited on page 1.

[Holland et al., 1983] P. W. Holland, K. B. Laskey, and S. Leinhardt. *Stochastic blockmodels: First steps*. Social networks, 5 (2), pp. 109–137, 1983. Cited on page 1.

[Jure et al., 2010] L. Jure, C. Deepayan, K. Jon, F. Christos, and G. Zoubin. *Rtg: a recursive realistic graph generator using random typing*. In *Journal of Machine Learning Research*, pp. 985–1042. 2010. Cited on page 14.

[Kolda et al., 2014] T. G. Kolda, A. Pinar, T. Plantenga, and C. Seshadhri. *A scalable generative graph model with community structure*. SIAM Journal on Scientific Computing, 36 (5), pp. C424–C452, 2014. Cited on page 1.

[Leskovec et al., 2010] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani. *Kronecker graphs: an approach to modeling networks*. Journal of Machine Learning Research, 11 (2), 2010. Cited on page 2.

[Leskovec et al., 2005] J. Leskovec, J. Kleinberg, and C. Faloutsos. *Graphs over time: densification laws, shrinking diameters and possible explanations*. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 177–187. 2005. Cited on page 12.

[Leskovec et al., 2007] ———. *Graph evolution: Densification and shrinking diameters*. ACM transactions on Knowledge Discovery from Data (TKDD), 1 (1), pp. 2–es, 2007. Cited on page 12.

[Ramani et al., 2017] S. A. Ramani, N. Eikmeier, and F. D. Gleich. *Coin-flipping, ball-dropping, and grass-hopping for generating random graphs from matrices of edge probabilities*. pp. 4,5,6, and 7, 2017. Cited on page 13.

[Ranshous et al., 2016] S. Ranshous, S. Harenberg, K. Sharma, and N. F. Samatova. *A scalable approach for outlier detection in edge streams using sketch-based approximations*. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pp. 189–197. 2016. Cited on page 1.

[Sala et al., 2010] A. Sala, H. Zheng, B. Y. Zhao, S. Gaito, and G. P. Rossi. *Brief announcement: revisiting the power-law degree distribution for social graph analysis*. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pp. 400–401. 2010. Cited on page 11.

[Savage, 2017] D. Savage. *Detection of illicit behaviours and mining for contrast patterns*. RMIT University, 2017. Cited on page 1.

[Scholtes, 2017] I. Scholtes. *When is a network a network? multi-order graphical model selection in pathways and temporal networks*. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1037–1046. 2017. Cited on page 1.

[Tamás, 2017] N. Tamás. *Fitting power-law distributions to empirical data*. https://github.com/ntamas/plfit, 2017. Cited on page 11.

[Xu et al., 2016] J. Xu, T. L. Wickramarathne, and N. V. Chawla. *Representing higher-order dependencies in networks*. Science Advances, 2 (5), 2016. arXiv:https://advances.sciencemag.org/content/2/5/e1600028.full.pdf, doi:10.1126/sciadv.1600028. Cited on page 1.

[Yin et al., 2018] H. Yin, A. R. Benson, and J. Leskovec. *Higher-order clustering in networks*. Phys. Rev. E, 97, p. 052306, 2018. doi:10.1103/PhysRevE.97.052306. Cited on page 1.