RTG: A Recursive Realistic Graph Generator Using Random Typing

Leman Akoglu and Christos Faloutsos

Carnegie Mellon University
School of Computer Science
{lakoglu,christos}@cs.cmu.edu

Abstract. We propose a new, recursive model to generate realistic graphs, evolving over time. Our model has the following properties: it is (a) flexible, capable of generating the cross product of weighted/ unweighted, directed/undirected, uni/bipartite graphs; (b) realistic, giving graphs that obey *eleven* static and dynamic laws that real graphs follow (we formally prove that for several of the (power) laws and we estimate their exponents as a function of the model parameters); (c) parsimonious, requiring only *four* parameters. (d) fast, being linear on the number of edges; (e) simple, intuitively leading to the generation of macroscopic patterns. We empirically show that our model mimics two real-world graphs very well: Blognet (unipartite, undirected, unweighted) with 27K nodes and 125K edges; and Committee-to-Candidate campaign donations (bipartite, directed, weighted) with 23K nodes and 880K edges. We also show how to handle time so that edge/weight additions are bursty and self-similar.

1 Introduction

Study of complex graphs such as computer and biological networks, the link structure of the WWW, the topology of the Internet, and recently with the widespread use of the Internet, large social networks, has been a vital research area. Many fascinating properties have been discovered, such as small and shrinking diameter [2,20], power-laws [5,11,16,24,22,28,29,20], and community structures [12,13,27]. As a result of such interesting patterns being discovered, and for many other reasons which we will discuss next, how to find a model that would produce synthetic but realistic graphs is a natural question to ask. There are several applications and advantages of modeling real-world graphs:

- *Simulation studies*: if we want to run tests for, say a spam detection algorithm, and want to observe how the algorithm behaves on graphs with different sizes and structural properties, we can use graph generators to produce such graphs by changing the parameters. This is also true when it is difficult to collect any kind of real data.
- *Sampling/Extrapolation*: we can generate a smaller graph for example for visualization purposes or in case the original graph is too big to run tests

W. Buntine et al. (Eds.): ECML PKDD 2009, Part I, LNAI 5781, pp. 13–28, 2009.

on it; or conversely to generate a larger graph for instance to make future prediction and answer what-if questions.

– *Summarization/Compression*: model parameters can be used to summarize and compress a given graph as well as to measure similarity to other graphs.
– *Motivation to understand pattern generating processes*: graph generators give intuition and shed light upon what kind of processes can (or cannot) yield the emergence of certain patterns. Moreover, modeling addresses the question of what patterns real networks exhibit that needs to be matched and provides motivation to figure out such properties.

Graph generator models are surveyed in [4]. Ideally, we would like a graph generator that is:

1. *simple*: it would be easy to understand and it would intuitively lead to the emergence of macroscopic patterns.
2. *realistic*: it would produce graphs that obey all the discovered "laws" of real-world graphs with appropriate values.
3. *parsimonious*: it would require only a few number of parameters.
4. *flexible*: it would be able to generate the cross product of weighted/unweighted, directed/undirected and unipartite/bipartite graphs.
5. *fast*: the generation process would ideally take linear time with respect to the number of edges in the output graph.

In this paper we propose RTG, for *Random Typing Generator*. Our model uses a process of 'random typing', to generate source and destination node identifiers, and it meets all the above requirements. In fact, we show that it can generate graphs that obey all *eleven* patterns that real graphs typically exhibit.

Next, we provide a survey on related work. Section 3 describes our RTG generator in detail. Section 4 provides experimental results and discussion. We conclude in Section 5. Appendix gives proofs showing some of the power-laws that the model generates.

2 Related Work

Graph patterns: Many interesting patterns that real graphs obey have been found, which we give a detailed list of in the next section. Ideally, a generator should be able to produce all of such properties.

Graph generators: The vast majority of earlier graph generators have focused on modeling a small number of common properties, but fail to mimic others. Such models include the *Erdos & Renyi* model [8], the *preferential attachment* model [3] and numerous more, like the *'small-world'*, *'winners don't take all'*, *'forest fire'* and *'butterfly'* models [31,26,20,22]. See [4] for a recent survey and discussion. In general, these methods are limited in trying to model some static graph property while neglecting others as well as dynamic properties or cannot be generalized to produce *weighted* graphs.

Random dot product graphs [17,32] assign each vertex a random vector in some d-dimensional space and an edge is put between two vertexes with probability equal to the dot product of the endpoints. This model does not generate weighted graphs and by definition only produces undirected graphs. It also seems to require the computation of the dot product for each pair of nodes which takes *quadratic* time.

A different family of models is utility-based, where agents try to optimize a predefined utility function and the network structure takes shape from their collective strategic behavior [10,9,18]. This class of models, however, is usually hard to analyze.

Kronecker graph generators [19] and their tensor followups [1] are successful in the sense that they match several of the properties of real graphs and they have proved useful for generating self-similar properties of graphs. However, they have two disadvantages: The first is that they generate multinomial/lognormal distributions for their degree and eigenvalue distribution, instead of a power-law one. The second disadvantage is that it is not easy to grow the graph incrementally: They have a fixed, predetermined number of nodes (say, N^k, where N is the number of nodes of the generator graph, and k is the number of iterations); where adding more edges than expected does *not* create additional nodes. In contrast, in our model, nodes emerge naturally.

3 Proposed Model

We first give a concise list of the *static* and *dynamic* 'laws' that real graphs obey, which a graph generator should be able to match.

L01. *Power-law degree distribution*: the degree distribution should follow a power-law in the form of $f(d) \propto d^{\gamma}$, with the exponent $\gamma < 0$ [5,11,16,24]

L02. *Densification Power Law (DPL)*: the number of nodes N and the number of edges E should follow a power-law in the form of $E(t) \propto N(t)^{\alpha}$, with $\alpha > 1$, over time [20].

L03. *Weight Power Law (WPL)*: the total weight of the edges W and the number of edges E should follow a power-law in the form of $W(t) \propto E(t)^{\beta}$, with $\beta > 1$, over time [22].

L04. *Snapshot Power Law (SPL)*: the total weight of the edges W_n attached to each node and the number of such edges, that is, the degree d_n should follow a power-law in the form of $W_n \propto d_n^{\theta}$, with $\theta > 1$ [22].

L05. *Triangle Power Law (TPL)*: the number of triangles Δ and the number of nodes that participate in Δ number of triangles should follow a power-law in the form of $f(\Delta) \propto \Delta^{\sigma}$, with $\sigma < 0$ [29].

L06. *Eigenvalue Power Law (EPL)*: the eigenvalues of the adjacency matrix of the graph should be power-law distributed [28].

L07. *Principal Eigenvalue Power Law (λ_1PL)*: the largest eigenvalue λ_1 of the adjacency matrix of the graph and the number of edges E should follow a power-law in the form of $\lambda_1(t) \propto E(t)^{\delta}$, with $\delta < 0.5$, over time [1].

L08. *small and shrinking diameter*: the (effective) diameter of the graph should be small [2] with a possible spike at the 'gelling point' [22]. It should also shrink over time [20].

L09. *constant size secondary and tertiary connected components*: while the 'giant connected component' keeps growing, the secondary and tertiary connected components tend to remain constant in size with small oscillations [22].

L10. *community structure*: the graph should exhibit a modular structure, with nodes forming groups, and possibly groups within groups [12,13,27].

L11. *bursty/self-similar edge/weight additions*: Edge (weight) additions to the graph over time should be self-similar and bursty rather than uniform with possible spikes [7,14,15,22].

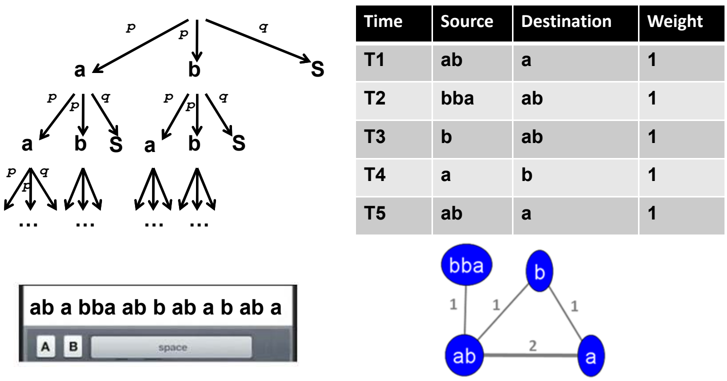
Zipf introduced probably the earliest power law [33], stating that, in many natural languages, the rank r and the frequency f_r of vocabulary words follow a power-law $f_r \propto 1/r$. Mandelbrot [21] argued that Zipf's law is the result of optimizing the average amount of information per unit transmission cost. Miller [23] showed that a random process also leads to Zipf-like power laws. He suggested the following experiment: "A monkey types randomly on a keyboard with k characters and a space bar. A space is hit with probability q; all other characters are hit with equal probability, $\frac{(1-q)}{k}$. A space is used to separate words". The distribution of the resulting words of this random typing process follow a power-law. Conrad and Mitzenmacher [6] showed that this relation still holds when the keys are hit with *unequal* probability.

Our model generalizes the above model of natural human behavior, using 'random typing'. We build our model RTG (*Random Typing Generator*) in *three* steps, incrementally. In the next two steps, we introduce the base version of the proposed model to give an insight. However, as will become clear, it has *two* shortcomings. In particular, the base model does not capture (1) homophily, the tendency to associate and bond with similar others- people tend to be acquainted with others similar in age, class, geographical area, etc. and (2) community structure, the existence of groups of nodes that are more densely connected internally than with the rest of the graph.

3.1 RTG-IE: RTG with Independent Equiprobable Keys

As in Miller's experimental setting, we propose each unique word typed by the monkey to represent a node in the output graph (one can think of each unique word as the label of the corresponding node). To form links between nodes, we mark the sequence of words as 'source' and 'destination', alternatingly. That is, we divide the sequence of words into groups of two and link the first node to the second node in each pair. If two nodes are already linked, the weight of the edge is simply increased by 1. Therefore, if W words are typed, the total weight of the output graph is $W/2$. See Figure 1 for an example illustration. Intuitively, random typing introduces new nodes to the graph as more words are typed, because the possibility of generating longer words increases with increasing number of words typed.

Due to its simple structure, this model is very easy to implement and is indeed mathematically tractable. If W words are typed on a keyboard with k keys and



Time	Source	Destination	Weight
T1	ab	a	1
T2	bba	ab	1
T3	b	ab	1
T4	a	b	1
T5	ab	a	1

ab a bba ab b ab a b ab a

Fig. 1. Illustration of the RTG-IE. Upper left: how words are (recursively) generated on a keyboard with two equiprobable keys, ‘a’ and ‘b’, and a space bar; lower left: a keyboard is used to randomly type words, separated by the space character; upper right: how words are organized in pairs to create source and destination nodes in the graph over time; lower right: the output graph; each node label corresponds to a unique word, while labels on edges denote weights

a space bar, the probability p of hitting a key being the same for all keys and the probability of hitting the space bar being denoted as $q{=}(1-kp)$:

Lemma 1. *The expected number of nodes N in the output graph G of the RTG-IE model is*

$$N \propto W^{-log_p k}.$$

Proof: In the Appendix. □

Lemma 2. *The expected number of edges E in the output graph G of the RTG-IE model is*

$$E \approx W^{-log_p k} * (1 + c' log W), \quad for\ c' = \frac{q^{-log_p k}}{-log p} > 0.$$

Proof: In the Appendix. □

Lemma 3. *The in(out)-degree d_n of a node in the output graph G of the RTG-IE model is power law related to its total in(out)-weight W_n, that is,*

$$W_n \propto d_n^{-log_k p}$$

with expected exponent $-log_k p > 1$.

Proof: In the Appendix. □

Even though most of the properties listed at the beginning of this section are matched, there are two problems with this model: (1) the degree distribution follows a power-law only for small degrees and then shows multinomial characteristics (See Figure 2), and (2) it does not generate homophily and community structure, because it is possible for every node to get connected to every other node, rather than to ‘similar’ nodes in the graph.

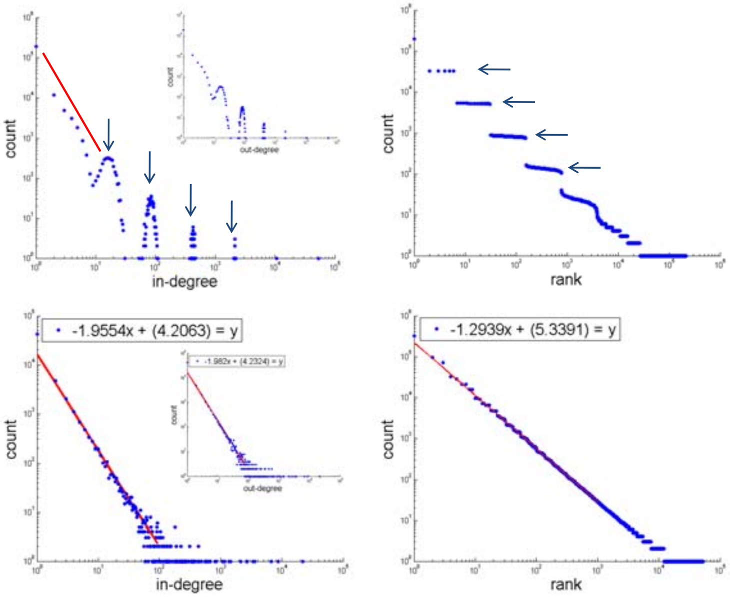


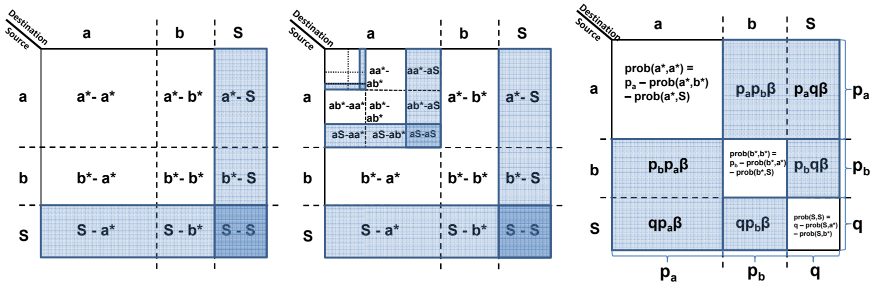
Fig. 2. Top row: Results of RTG-IE ($k = 5$, $p = 0.16$, $W = 1M$). The problem with this model is that in(out)-degrees form multinomial clusters (left). This is because nodes with labels of the same length are expected to have the same degree. This can be observed on the rank-frequency plot (right) where we see many words with the same frequency. Notice the 'staircase effect'. Bottom row: Results of RTG-IU ($k = 5$, $p = [0.03, 0.05, 0.1, 0.22, 0.30]$, $W = 1M$). Unequal probabilities introduce smoothing on the frequency of words that are of the same length (right). As a result, the degree distribution follows a power-law with expected heavy tails (left).

3.2 RTG-IU: RTG with Independent Un-Equiprobable Keys

We can spread the degrees so that nodes with the same-length but otherwise distinct labels would have different degrees by making keys have *unequal* probabilities. This procedure introduces smoothing in the distribution of degrees, which remedies the first problem introduced by the RTG-IE model. In addition, thanks to [6], we are still guaranteed to obtain the desired power-law characteristics as before. See Figure 2.

3.3 RTG: Random Typing Graphs

What the previous model fails to capture is the homophily and community structure. In a real network, we would expect nodes to get connected to similar nodes (homophily), and form groups and possibly groups within groups (modular structure). In our model, for example on a keyboard with two keys 'a' and 'b', we would like nodes with many 'a's in their labels to be connected to similar nodes, as opposed to nodes labeled with many 'b's. However, in both RTG-IE and



(a) first level (b) recursion (c) communities

Fig. 3. The **RTG** model: random typing on a 2-d keyboard, generating edges (source-destination pairs). See Algorithm 1. (a) an example 2-d keyboard (nine keys), hitting a key generates the row(column) character for source(destination), shaded keys terminate source and/or destination words. (b) illustrates recursive nature. (c) the imbalance factor β favors diagonal keys and leads to homophily.

RTG-IU it is possible for every node to connect to every other node. In fact, this yields a tightly connected core of nodes with rather short labels.

Our proposal to fix this is to envision a two-dimensional keyboard that generates source and destination labels in one shot, as shown in Figure 3. The previous model generates a word for source, and, completely independently, another word for destination. In the example with two keys, we can envision this process as picking one of the nine keys in Figure 3(a), using the independence assumption: the probability for each key is the product of the probability of the corresponding row times the probability of the corresponding column: p_l for letter l, and q for space ('S'). After a key is selected, its row character is appended to the source label, and the column character to the destination label. This process repeats recursively as in Figure 3(b), until the space character is hit on the first dimension in which case the source label is terminated and also on the second dimension in which case the destination label is terminated.

In order to model homophily and communities, rather than assigning cross-product probabilities to keys on the 2-d keyboard, we introduce an imbalance factor β, which will decrease the chance of a-to-b edges, and increase the chance for a-to-a and b-to-b edges, as shown in Figure 3(c). Thus, for the example that we have, the formulas for the probabilities of the nine keys become:

$$\begin{aligned}
prob(a,b) = prob(b,a) = p_a p_b \beta \ &, \ prob(a,a) = p_a - (prob(a,b) + prob(a,S)),\\
prob(S,a) = prob(a,S) = q p_a \beta \ &, \ prob(b,b) = p_b - (prob(b,a) + prob(b,S)),\\
prob(S,b) = prob(b,S) = q p_b \beta \ &, \ prob(S,S) = q - (prob(S,a) + prob(S,b)).
\end{aligned}$$

By boosting the probabilities of the diagonal keys and down-rating the probabilities of the off-diagonal keys, we are guaranteed that nodes with similar labels will have higher chance to get connected. The pseudo-code of generating edges as described above is shown in Algorithm 1.

Next, before showing the experimental results of RTG, we take a detour to describe how we handle time so that edge/weight additions are bursty and self-similar. We also discuss the generalizations of the model in order to produce all types of uni/bipartite, (un)weighted, and (un)directed graphs.

Algorithm 1. RTG

```
Input: $k$, $q$, $W$, $\beta$
Output: edge-list L for output graph $\mathcal{G}$
Initialize $(k+1)$-by-$(k+1)$ matrix $P$ with cross-product probabilities
// in order to ensure homophily and community structure
Multiply off-diagonal probabilities by $\beta$, $0 < \beta < 1$
Boost diagonal probabilities s.t. sum of row(column) probabilities remain the same.
Initialize edge list L
for 1 to $W$ do
    L1, L2 $\leftarrow$ SelectNodeLabels($P$)
    Append L1, L2 to L
end for

function SelectNodeLabels ($P$) : L1, L2
Initialize L1 and L2 to empty string
while not terminated L1 and not terminated L2 do
    Draw $i$,$j$ with probability $P(i,j)$
    if $i \leq k$, $j \leq k$ then
        Append character ‘i’ to L1 and ‘j’ to L2 if not terminated
    else if $i \leq k$, $j$=$k+1$ then
        Append character ‘i’ to L1 if not terminated
        Terminate L2
    else if $i$=$k+1$, $j \leq k$ then
        Append character ‘j’ to L2 if not terminated
        Terminate L1
    else
        Terminate L1 and L2
    end if
end while
Return L1 and L2
end function
```

3.4 Burstiness and Self-Similarity

Most real-world traffic as well as edge/weight additions to real-world graphs have been found to be self-similar and bursty [7,14,15,22]. Therefore, in this section we give a brief overview of how to aggregate time so that edge and weight additions, that is ΔE and ΔW, are bursty and self-similar.

Notice that when we link two nodes at each step, we add 1 to the total weight W. So, if every step is represented as a single time-tick, the weight additions are uniform. However, to generate bursty traffic, we need to have a *bias factor* b> 0.5, such that b-fraction of the additions happen in one half and the remaining in the other half. We will use the b-model [30], which generates such self-similar and bursty traffic. Specifically, starting with a uniform interval, we will recursively subdivide weight additions to each half, quarter, and so on, according to the bias b. To create randomness, at each step we will randomly swap the order of fractions b and $(1-b)$.

Among many methods that measure self-similarity we use the entropy plot [30], which plots the entropy $H(r)$ versus the resolution r. The resolution is the scale, that is, at resolution r, we divide our time interval into 2^r equal sub-intervals, compute ΔE in each sub-interval $k(k = 1 \ldots 2^r)$, normalize into fractions $p_k = \frac{\Delta E}{E}$, and compute the Shannon entropy $H(r)$ of the sequence p_k. If the plot $H(r)$ is linear, the corresponding time sequence is said to be *self-similar*, and the slope of the plot is defined as the fractal dimension f_d of the time sequence. Notice that a uniform Δ distribution yields f_d=1; a lower value of f_d corresponds to a more bursty time sequence, with a single burst having the lowest f_d=0: the fractal dimension of a point.

3.5 Generalizations

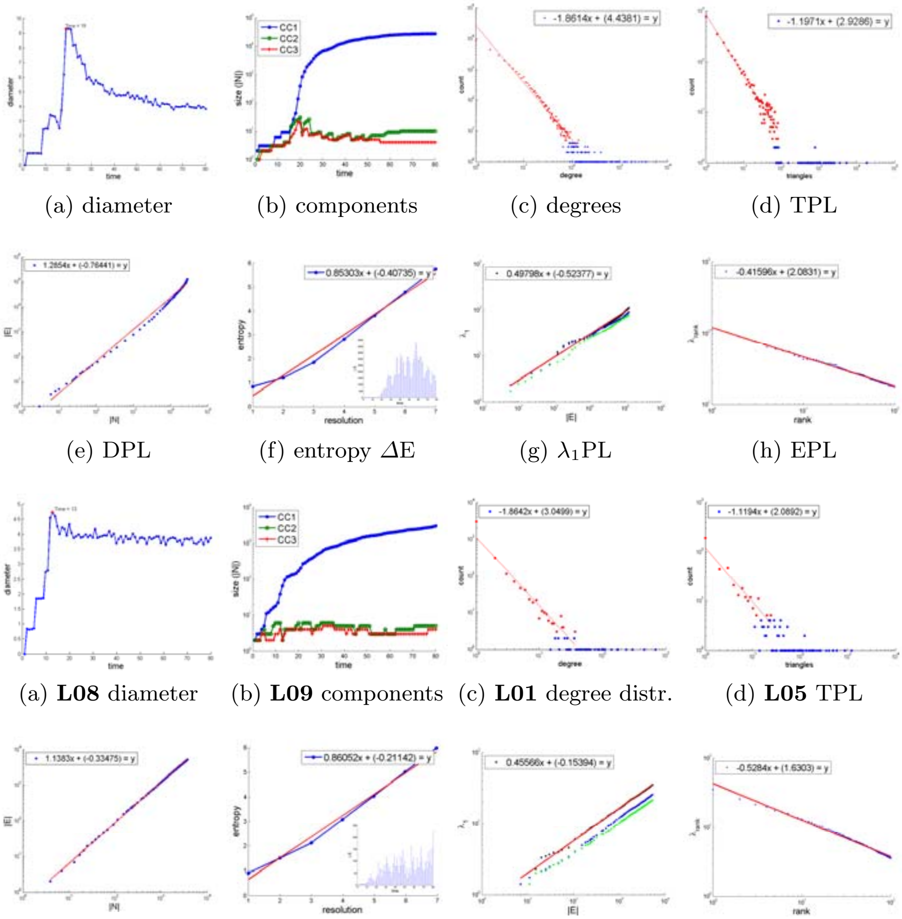
We can easily generalize RTG to model all type of graphs. To generate undirected graphs, we can simply assume edges from source to destination to be undirected as the formation of source and destination labels is the same and symmetric. For unweighted graphs, we can simply ignore *duplicate* edges, that is, edges that connect already linked nodes. Finally, for bipartite graphs, we can use *two* different sets of keys such that on the 2-d keyboard, source dimension contains keys from the first set, and the destination dimension from the other set. This assures source and destination labels to be completely different, as desired.

4 Experimental Results

The question we wish to answer here is how RTG is able to model real-world graphs. The datasets we used are:
Blognet: a social network of blogs based on citations (undirected, unipartite and unweighted with N=27, 726; E=126, 227; over 80 time ticks).
Com2Cand: the U.S. electoral campaign donations network from organizations to candidates (directed, bipartite and weighted with N=23, 191; E=877, 721; and W=4, 383, 105, 580 over 29 time ticks). Weights on edges indicate donated dollar amounts.

In Figures 4 and 5, we show the related patterns for *Blognet* and *Com2Cand* as well as synthetic results, respectively. In order to model these networks, we ran experiments for different parameter values k, q, W, and β. Here, we show the closest results that RTG generated, though fitting the parameters is a challenging future direction. We observe that RTG is able to match the *long* wish-list of static and dynamic properties we presented earlier for the two real graphs.

In order to evaluate community structure, we use the modularity measure in [25]. Figure 6(left) shows that modularity increases with smaller imbalance factor β. Without any imbalance, β=1, modularity is as low as 0.35, which indicates that no significant modularity exists. In Figure 6(right), we also show



(a) diameter (b) components (c) degrees (d) TPL

(e) DPL (f) entropy ΔE (g) λ_1PL (h) EPL

(a) **L08** diameter (b) **L09** components (c) **L01** degree distr. (d) **L05** TPL

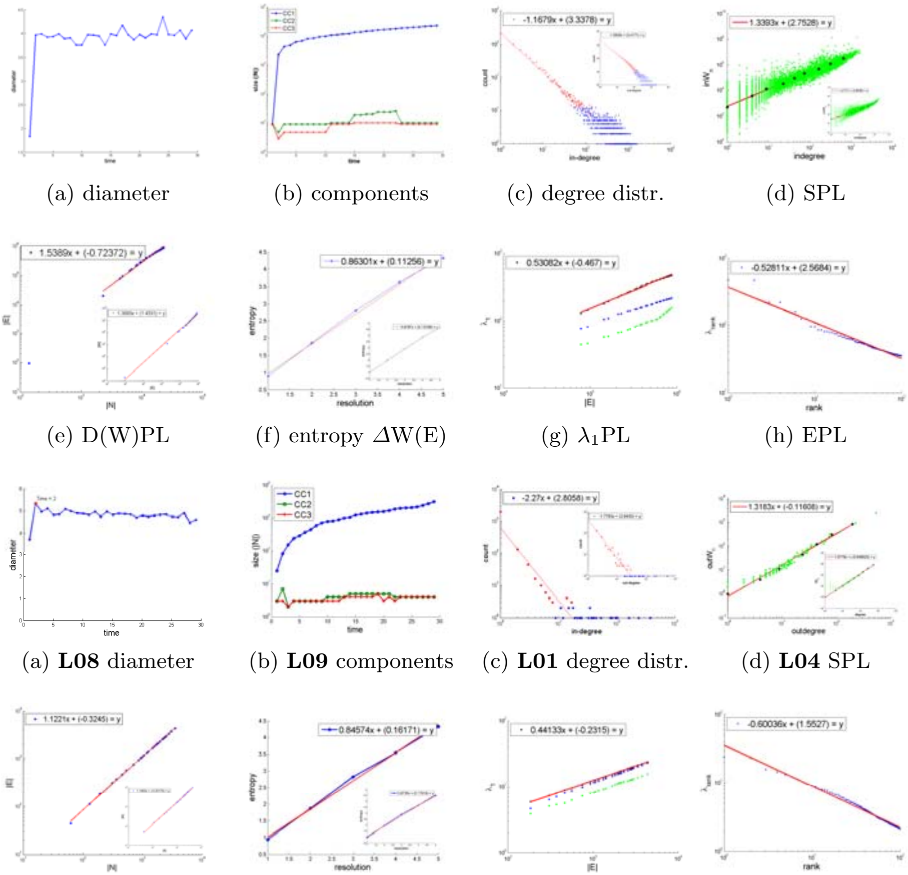
(e) **L02** DPL (f) **L11** entropy ΔE (g) **L07** λ_1PL (h) **L06** EPL

Fig. 4. Top two rows: properties of *Blognet*: (a) small and shrinking diameter; (b) largest 3 connected components; (c) degree distribution; (d) triangles Δ vs number of nodes with Δ triangles; (e) densification; (f) bursty edge additions; (g) largest 3 eigenvalues wrt E; (h) rank spectrum of the adjacency matrix. Bottom two rows: results of RTG. Notice the similar qualitative behavior for all *eight* laws.

the running time of RTG wrt the number of duplicate edges (that is, number of iterations W). Notice the *linear* growth with increasing W.

5 Conclusion

We have designed a generator that meets all the five desirable properties in the introduction. Particularly, our model is



(a) diameter (b) components (c) degree distr. (d) SPL

(e) D(W)PL (f) entropy ΔW(E) (g) λ_1PL (h) EPL

(a) **L08** diameter (b) **L09** components (c) **L01** degree distr. (d) **L04** SPL

(e) **L02, L03** D(W)PL (f) **L11** entropy ΔW(E) (g) **L07** λ_1PL (h) **L06** EPL

Fig. 5. Top two rows: properties of *Com2Cand*; as opposed to *Blognet*, *Com2Cand* is *weighted*. So, different from above we show: (d) node weight vs in(inset: out)degree; (e) total weight vs number of edges(inset); (f) bursty weight additions(inset); Bottom two rows: results of RTG. Notice the similar qualitative behavior for all *nine* laws.

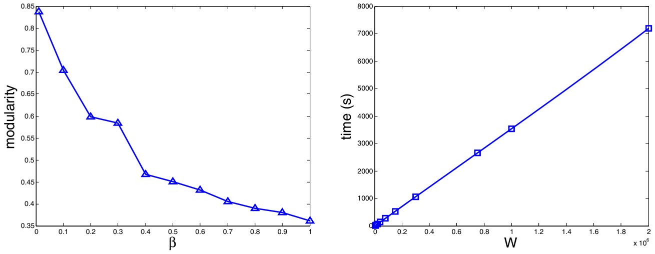


Fig. 6. Left: modularity score vs. imbalance factor β, modularity increases with decreasing β. For β=1, the score is very low indicating no significant modularity. Right: computation time vs. W, time grows *linearly* with increasing number of iterations W.

1. simple and intuitive, yet it generates the emergent, macroscopic patterns that we see in real graphs.
2. realistic, generating graphs that obey all *eleven* properties that real graphs obey - no other generator has been shown to achieve that.
3. parsimonious, requiring only a handful of parameters.
4. flexible, capable of generating weighted/unweighted, directed/undirected, and unipartite/bipartite graphs, and any combination of the above.
5. fast, being linear on the number of iterations (on a par with the number of duplicate edges in the output graph).

Moreover, we showed how well RTG can mimic some large, real graphs. We have also proven that an early version of RTG generates several of the desired (power) laws, formulated in terms of model parameters.

Acknowledgments. This material is based upon work supported by the National Science Foundation under Grants No. IIS-0705359, IIS-0705215 and IIS-0808661, also by the iCAST project sponsored by the National Science Council, Taiwan, under Grants No. NSC97-2745-P-001-001 and by the Ministry of Economic Affairs, Taiwan, under Grants No. 97-EC-17-A-02-R7-0823 and under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 (LLNL-CONF-404625), subcontracts B579447, B580840. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any of the funding parties.

References

1. Akoglu, L., McGlohon, M., Faloutsos, C.: Rtm: Laws and a recursive generator for weighted time-evolving graphs. In: ICDM (2008)
2. Albert, R., Jeong, H., Barabasi, A.-L.: Diameter of the World Wide Web. Nature 401, 130–131 (1999)
3. Barabasi, A.L., Albert, R.: Emergence of scaling in random networks. Science 286(5439), 509–512 (1999)
4. Chakrabarti, D., Faloutsos, C.: Graph mining: Laws, generators, and algorithms. ACM Comput. Surv. 38(1) (2006)
5. Chakrabarti, D., Zhan, Y., Faloutsos, C.: R-MAT: A recursive model for graph mining. In: SIAM Int. Conf. on Data Mining (April 2004)
6. Conrad, B., Mitzenmacher, M.: Power laws for monkeys typing randomly: the case of unequal probabilities. IEEE Transactions on Information Theory 50(7), 1403–1414 (2004)
7. Crovella, M., Bestavros, A.: Self-similarity in world wide web traffic, evidence and possible causes. Sigmetrics, 160–169 (1996)
8. Erdos, P., Renyi, A.: On the evolution of random graphs. Publ. Math. Inst. Hungary. Acad. Sci. 5, 17–61 (1960)
9. Even-Bar, E., Kearns, M., Suri, S.: A network formation game for bipartite exchange economies. In: SODA (2007)
10. Fabrikant, A., Luthra, A., Maneva, E.N., Papadimitriou, C.H., Shenker, S.: On a network creation game. In: PODC (2003)

11. Faloutsos, M., Faloutsos, P., Faloutsos, C.: On power-law relationships of the internet topology. In: SIGCOMM, August-September 1999, pp. 251–262 (1999)
12. Flake, G.W., Lawrence, S., Giles, C.L., Coetzee, F.M.: Self-organization and identification of web communities. IEEE Computer 35, 66–71 (2002)
13. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. PNAS 99, 7821 (2002)
14. Gomez, M.E., Santonja, V.: Self-similarity in i/o workload: Analysis and modeling. In: WWC (1998)
15. Gribble, S.D., Manku, G.S., Roselli, D., Brewer, E.A., Gibson, T.J., Miller, E.L.: Self-similarity in file systems. In: SIGMETRICS 1998 (1998)
16. Kleinberg, J.M., Kumar, R., Raghavan, P., Rajagopalan, S., Tomkins, A.S.: The Web as a graph: Measurements, models and methods. In: Asano, T., Imai, H., Lee, D.T., Nakano, S.-i., Tokuyama, T. (eds.) COCOON 1999. LNCS, vol. 1627, pp. 1–17. Springer, Heidelberg (1999)
17. Scheinerman, E., Kraetzl, M., Nickel, C.: Random dot product graphs: a model for social networks (Preliminary Manuscript) (2005)
18. Laoutaris, N., Poplawski, L.J., Rajaraman, R., Sundaram, R., Teng, S.-H.: Bounded budget connection (bbc) games or how to make friends and influence people, on a budget. In: PODC (2008)
19. Leskovec, J., Chakrabarti, D., Kleinberg, J.M., Faloutsos, C.: Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In: Jorge, A.M., Torgo, L., Brazdil, P.B., Camacho, R., Gama, J. (eds.) PKDD 2005. LNCS (LNAI), vol. 3721, pp. 133–145. Springer, Heidelberg (2005)
20. Leskovec, J., Kleinberg, J., Faloutsos, C.: Graphs over time: densification laws, shrinking diameters and possible explanations. In: ACM SIGKDD (2005)
21. Mandelbrot, B.: An informational theory of the statistical structure of language. Communication Theory (1953)
22. McGlohon, M., Akoglu, L., Faloutsos, C.: Weighted graphs and disconnected components: Patterns and a generator. In: ACM SIGKDD, Las Vegas (August 2008)
23. Miller, G.A.: Some effects of intermittent silence. American Journal of Psychology 70, 311–314 (1957)
24. Newman, M.E.J.: Power laws, Pareto distributions and Zipf's law (December 2004)
25. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Physical Review E 69, 026113 (2004)
26. Pennock, D.M., Flake, G.W., Lawrence, S., Glover, E.J., Giles, C.L.: Winners don't take all: Characterizing the competition for links on the web. Proceedings of the National Academy of Sciences, 5207–5211 (2002)
27. Schwartz, M.F., Wood, D.C.M.: Discovering shared interests among people using graph analysis of global electronic mail traffic. Communications of the ACM 36, 78–89 (1992)
28. Siganos, G., Faloutsos, M., Faloutsos, P., Faloutsos, C.: Power laws and the AS-level internet topology (2003)
29. Tsourakakis, C.E.: Fast counting of triangles in large real networks without counting: Algorithms and laws. In: ICDM (2008)
30. Wang, M., Madhyastha, T., Chan, N.H., Papadimitriou, S., Faloutsos, C.: Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic. In: ICDE, pp. 507–516 (2002)
31. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world' networks. Nature 393(6684), 440–442 (1998)

32. Young, S.J., Scheinerman, E.R.: Random dot product graph models for social networks. In: Bonato, A., Chung, F.R.K. (eds.) WAW 2007. LNCS, vol. 4863, pp. 138–149. Springer, Heidelberg (2007)
33. Zipf, G.K.: Selective Studies and the Principle of Relative Frequency in Language. Harvard University Press (1932)

Appendix

Consider the following setting: W words are typed on a keyboard with k keys and a space bar, the probability of hitting a key p being the same for all keys and probability of hitting the space bar being denoted as q=$(1 - kp)$, in the output graph G of the RTG-IE model:

Lemma 1. *The expected number of nodes N is*

$$N \propto W^{-log_p k}.$$

Proof. Given the number of words W, we want to find the expected number of nodes N that the RTG-IE graph consists of. This question can be reformulated as follows: "Given W words typed by a monkey on a keyboard with k keys and a space bar, what is the size of the vocabulary V?" The number of unique words V is basically equal to the number of nodes N in the output graph.

Let w denote a single word generated by the defined random process. Then, w can recursively be written as follows: "$w : c_i w|S$", where c_i is the character that corresponds to key i, $1 \leq i \leq k$, and S is the space character. So, V as a function of model parameters can be formulated as:

$$\begin{aligned} V(W) &= V(c_1, Wp) + V(c_2, Wp) + \ldots + V(c_k, Wp) + V(S) \\ &= k * V(Wp) + V(S) = \quad k * V(Wp) + \begin{cases} 1, 1 - (1-q)^W \\ 0, \ (1-q)^W \end{cases} \end{aligned}$$

where q denotes the probability of hitting the space bar, i.e. $q = 1 - kp$. Given the fact that W is often large, and $(1 - q) < 1$, it is almost always the case that w=S is generated; but since this adds only a constant factor, we can ignore it in the rest of the computation. That is,

$$V(W) \approx k * V(Wp) = k * (k * V(Wp^2)) = k^n * V(1)$$

where $n = log_p(1/W) = -log_p W$. By definition, when W=1, that is, in case only one word is generated, the vocabulary size is 1, i.e. V(1)=1. Therefore,

$$V(W) = N \propto k^n = k^{-log_p W} = W^{-log_p k}.$$

□

The above proof shown using recursion is in agreement with the early result of Miller [23], who showed that in the monkey-typing experiment with k equiprobable keys (with probability p) and a space bar (with probability q), the rank-frequency distribution of words follow a power law. In particular,

$$f(r) \propto r^{-1+log_k(1-q)-1} = r^{log_k p}.$$

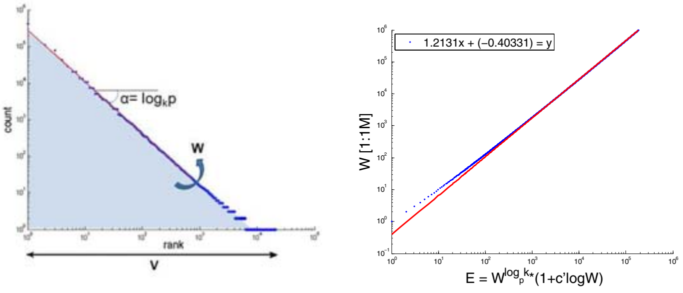


Fig. 7. (a) Rank vs count of vocabulary words typed randomly on a keyboard with k equiprobable keys (with probability p) and a space bar (with probability q), follow a power law with exponent $\alpha = log_k p$. Approximately, the area under the curve gives the total number of words typed. (b) The relationship between number of edges E and total weight W behaves like a power-law (k=2, p=0.4).

In this case, the number of ranks corresponds to the number of unique words, that is, the vocabulary size V. And, the sum of the counts of occurrences of all words in the vocabulary should give W, the number of words typed. The total count can be approximated by the area under the curve on the rank-count plot. See Figure 7(a). Next, we give a second proof of Lemma 1 using Miller's result.

Proof. Let $\alpha = log_k p$ and $C(r)$ denote the number of times that the word with rank r is typed. Then, $C(r) = cr^{\alpha}$, where $C(r)_{min} = C(V) = cV^{\alpha}$ and the constant $c = C(V)V^{-\alpha}$. Then we can write W as

$$W = C(V)V^{-\alpha}\left(\sum_{r=1}^{V} r^{\alpha}\right) \approx C(V)V^{-\alpha}\left(\int_{r=1}^{V} r^{\alpha}dr\right) = C(V)V^{-\alpha}\left(\frac{r^{\alpha+1}}{\alpha+1}\Big|_{r=1}^{V}\right)$$
$$= C(V)V^{-\alpha}\left(\frac{1}{-\alpha-1} - \frac{1}{(-\alpha-1)V^{-\alpha-1}}\right) \approx c'V^{-\alpha}.$$

where $c' = \frac{C(V)}{-\alpha-1}$, where $\alpha < -1$ and $C(V)$ is very small (usually 1). Therefore,

$$V = N \propto W^{-\frac{1}{\alpha}} = W^{-log_p k}.$$

□

Lemma 2. *The expected number of edges E is*

$$E \approx W^{-log_p k} * (1 + c' logW), \quad for\ c' = \frac{q^{-log_p k}}{-logp} > 0.$$

Proof. Given the number of words W, we want to find the expected number of edges E that the RTG-IE graph consists of. The number of edges E is the same as the unique number of *pairs* of words. We can think of a pair of words as a single word e, the generation of which is stopped after the *second* hit to the space bar. So, e always contains a single space character. Recursively, "$e : c_i e|Sw$", where "$w : c_i w|S$". So, E can be formulated as:

$$E(W) = k * E(Wp) + V(Wq) \tag{1}$$

$$V(Wq) = \;\; k * V(Wqp) + \begin{cases} 1, \, 1-(1-q)^{Wq} \\ 0, \, (1-q)^{Wq} \end{cases} \tag{2}$$

From Lemma 1, Equ.(2) can be approximately written as $V(Wq) = (Wq)^{-log_p k}$. Then, Equ.(1) becomes $E(W) = k * E(Wp) + cW^{\alpha}$, where $c = q^{-log_p k}$ and $\alpha = -log_p k$. Given that E(W=1)=1, we can solve the recursion as follows:

$$\begin{aligned} E(W) &\approx k * (k * E(Wp^2) + c(Wp)^{\alpha}) + cW^{\alpha} \\ &= k * (k * (k * V(Wp^3) + c(Wp^2)^{\alpha}) + c(Wp)^{\alpha}) + cW^{\alpha} \\ &= k^n * V(1) + k^{n-1} * c(Wp^{n-1}) + k^{n-2} * c(Wp^{n-2})^{\alpha} + \ldots + cW^{\alpha} \\ &= k^n * V(1) + cW^{\alpha}((kp^{\alpha})^{n-1} + (kp^{\alpha})^{n-2} + \ldots + 1) \end{aligned}$$

where $n = log_p(1/W) = -log_p W$. Since $kp^{\alpha} = kp^{-log_p k} = 1$,

$$E(W) \approx k^n * V(1) + n * cW^{\alpha} = k^{-log_p W} + c\frac{-log\frac{1}{W}}{-logp} W^{-log_p k} = W^{-log_p k}(1 + c' logW)$$

where $c' = \frac{c}{-logp} = \frac{q^{-log_p k}}{-logp} > 0$.

□

The above function of E in terms of W and other model parameters looks like a power-law for a wide range of W. See Figure 7(b).

Lemma 3. *The in/out-degree d_n of a node is power law related to its total in/out-weight W_n, that is,*

$$W_n \propto d_n^{-log_k p}$$

with expected exponent $-log_k p > 1$.

Proof. We will show that $W_n \propto d_n^{-log_k p}$ for out-edges, and a similar argument holds for in-edges. Given that the experiment is repeated W times, let W_n denote the number of times a unique word is typed as a source. Each such unique word corresponds to a node in the final graph and W_n is basically its out-weight, since the node appears as a source node. Then, the out-degree d_n of a node is simply the number of unique words typed as a destination. From Lemma 1,

$$W_n \propto d_n^{-log_k p}, \;\; for \; -log_k p > 1.$$

□