A compact routing scheme for power-law networks using empirical discoveries in power-law graph topology

A thesis submitted by

Arthur R. Brady

In partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science

Tufts University

Date

May 2005

Advisor

Lenore J. Cowen

Abstract

The Internet is unique. It is the most complex quasi-physical entity created by humans: we can't completely model it, and we don't fully understand its structure or its growth patterns [21]. As a result, hot research topics in the past several years include the empirical study of Internet topology, the construction of mathematical models of "Internet-like" topology [20, 1], problems in network growth and evolution [2] and problems in network routing [25, 4, 17]. We propose a new compact routing scheme which builds on previous work in compact routing and Internet-like graph topology, augmented by empirical results on graph topology obtained using a new dynamic graph simulator.

Acknowledgements

My advisor Lenore Cowen exhibited superhuman patience and energy during the supervision of this thesis. Without her rescue back to academia, I'd be in a cubicle somewhere, wearing business-casual and wondering what to do with my life.

Patrick Schmid performed the unenviable task of drafting much of the DIGG code, and did so cheerfully, competently and tirelessly. Lev Makhlis contributed invaluable suggestions which helped streamline the codebase.

Dad and Mom go almost (but not quite) without saying.

Jay: Light. Love. Life. Bun.

Contents

Chapter 1

Background and Introduction

It's assumed that the reader is familiar with basic concepts of applied combinatorics (like graphs, multigraphs, cycles, graph diameter, trees, shortest-paths, etc.) and statistics (like probability distributions and sampling). If not, see [6] for an elegant and thorough introduction to graph theory, and [14] for a remarkably clearly-written introduction to probability and statistics.

This chapter will provide an introduction to the concepts of random graphs, power-law graphs and the compact routing problem.

1.1 Random graphs

The concept of *random graphs* first arose in 1959 [10] from the context of the *probabilistic method* [11, 3]. As a general example of this method, suppose we want to investigate the existence of a graph with a specific structural property P. We define a probability space in which the appearance of such a graph corresponds to an event E. If we can then prove that the probability of observing E is greater than zero, then some graph with property P must exist. Probability spaces like this, and their associated distributions, form the essence of the random graph idea.

The simplest (and until relatively recently, the most commonly studied) random graph model is one in which each edge appears with a fixed probability, independently of all other edges. This model is frequently referred to as $G(n, p)$. Formally, given a vertex set V with labels $\{1, \ldots, n\}$ and a fixed probability p, we define a random variable $G(n, p)$ to be a graph constructed from V, with each possible edge appearing with probability p, independently. Thus each subset of the complete graph on n vertices is selected as an instance of $G(n, p)$ with probability $p^m(1-p)^{\binom{n}{2}-m}$, where m is the number of edges in that subset. Another common model fixes the number of edges in the graph to define the probability space: given n and $m = m(n)$, select each graph with n vertices and m edges with (uniform) probability $\binom{\binom{n}{2}}{m}^{-1}$. This model is sometimes referred to as $G(n, m)$.

With these models we encounter the first type of heuristic commonly employed when analyzing or constructing instances of random graphs. Ideally, we'd like to look directly at *isomorphism classes* of graphs with a fixed size and a given structural property (such as "each edge appears in the graph with probability p independently"). These classes, however, are extremely difficult to compute for all but the simplest graphs, and they're hard to generate at random, because their sizes differ in ways difficult to determine using closed-form rules. So we consider graphs with a fixed vertex set instead, simplifying computation and analysis [26].

Other random graph models present their own particular difficulties with respect to analysis and construction, as we'll explore in the next section.

1.2 Power-law graphs

A *power-law distribution* D over some domain X is a distribution described (for fixed parameters α and β) by a rule $D(x) = \frac{e^\alpha}{x^\beta}$, for each $x \in X$. This rule is referred to (not surprisingly) as a power law. D is a probability distribution if values are normalized so that $\sum_{x \in X} D(x) = 1$; note that in this case, α is *uniquely determined by* $|X|$ *and* β. The term "power-law distribution" is also sometimes used to describe observed frequency distributions which are (approximately) proportional to their normalized counterparts.

Hence appears the idea of a "power-law graph," wherein the graph's *distribution of vertex degrees* follows (or closely approximates) some power law, generally described by $y \approx \frac{e^\alpha}{x^\beta}$, where y is the number of vertices in the graph of degree x. The *power-law random graph* model (Cf. e.g. [26], [7], [1], [20]), then, using parameters n and β (which unambiguously determine α), assigns uniform probability to all n-node graphs whose degree distributions match the corresponding power law as closely as possible; given that graphs represent discrete datasets and given the fact that β is an arbitrary real number, a minimal remainder-error term is allowed.

1.2.1 Generating models for power-law graphs

Several different models have been deployed recently to analyze and construct instances of power-law random graphs, due in part to their empirically-observed relationship to substructures of the Internet, about which we will have more to say in the next section.

The two such generators of importance to the present work are due to Aiello, Chung and Lu [1] and to Lu [20]. While the set of distributions they attempt to model are similar, the Aiello, Chung and Lu generator was designed for ease of sampling from the distribution, while the Lu generator was designed to make the mathematical analysis tractable.

The model of Aiello et al. fixes a degree sequence (based on input parameters) and constructs a multigraph matching that sequence. The procedure is as follows:

1. Form a set L containing $deg(v)$ distinct copies of each vertex v.

2. Choose a random matching of the elements of L.

3. For two vertices u and v, the number of edges joining u to v [in the target graph] is the number of edges in the matching of L joining copies of u to copies of v.

This is essentially the model we use to construct instances of random power-law graphs for our empirical study of power-law graph topology, using degree sequences produced as described below.

Lu's model, rather than fixing a degree sequence and applying it to a set of nodes, assigns *weights* to nodes corresponding to probabilities (again determined by input parameters). Edges are created to match successes on weighted coin flips, each combining a baseline probability with the weights on every pair of nodes. In this way the resulting graph has an *expected* power-law degree distribution. Lu's model is:

> Given n weighted vertices with weights $\{w_1, \ldots, w_n\}$, a pair of vertices (i, j) appears as an edge with probability $w_i w_j p$ independently.
> Here these parameters $\{w_1, \ldots, w_n\}$ and p satisfy:

1. The number of vertices i with weight w_i such that $1 \leq w_i < 2$ is $\lfloor e^{\alpha} \rfloor - r$.
2. The number of vertices i with weight w_i such that $k \leq w_i < k+1$ is $\lfloor \frac{e^{\alpha}}{k^{\beta}} \rfloor$ for all $k \in \{2, 3, \ldots, \lfloor e^{\frac{\alpha}{\beta}} \rfloor\}$.
3. The remainder term r is $n - \sum_{k=1}^{\lfloor e^{\frac{\alpha}{\beta}} \rfloor} \lfloor \frac{e^{\alpha}}{k^{\beta}} \rfloor$.
4. α is computed to minimize $|r|$.
5. $p := \frac{1}{\sum_{i=1}^{n} w_i}$.

Note that each weight w_i represents the *expected degree* of vertex i; we use a discretized version of steps 1–4 to create a fixed degree sequence for our simulator, which then produces a graph using the method of Aiello et al. Please see Chapter 3 for a description of our results.

Aiello et al. point out that the two models are "basically asymptotically equivalent, subject to bounding error estimates of the variances;" having combined elements of the two generators in the present work, we have not verified this empirically, but such a comparison should be made relatively easy by the DIGG code in its current incarnation (See Chapter 2).

1.2.2 Internet-like graphs and the present work

Much work has been done both in pursuit of rules describing the topology of the Internet at various levels of granularity, and toward the generation of reasonable simulations of Internet-like networks [18, 19, 23, 24]. It has been claimed[1] that from several different perspectives (e.g. router-level, inter-domain level, web document linkage level) that the structure of the Internet,

[1] Dr. Mark Crovella and others have recently called into question the empirical validity of Internet topology-discovery mechanisms (see e.g. [9]), but we'll assume this claim to be true for the purposes of this thesis.

modeled naturally as a graph, follows a precise power-law degree distribution which appears to be stable against the network's growth in size over time [12].

This thesis proposes to extend this research through software simulation of power-law random graphs of varying sizes and parameters, in an attempt to discover topological themes. It is hoped that these toplogical observations might in turn suggest useful perspectives on and insights into the design of physical networks and/or the creation of efficient routing protocols.

It's worth mentioning that both the power-law random graph models discussed so far are contrived so as to preserve edge independence, for ease of analysis. Links in the Internet however, are certainly not constructed or altered independently of one another. Another Internet-like model developed by Barabási and Albert [5] and extended by Bu and Towsley [7] grows a tunable power-law random graph using an iterative process, whereby new edges are preferentially connected to higher-degree nodes. This kind of model presents interesting possibilities for future analytic simulations of dynamic graphs.

1.3 Compact routing

Routing is an essential component of network function. A *routing scheme* is some mechanism by which nodes in a network can reliably distribute information packets to one another. Specifically, a routing scheme is a distributed algorithm which specifies the nature and (presumably successful) systemic interaction of three components: local (node-level) decision-making processes (called *daemons*), local memory which helps each daemon make routing decisions (referred to as *routing tables*), and the transmitted information itself (discretized into chunks or *packets*). Each packet is further divided into two parts: the first of these is its *message*, which is the information to be passed along to the packet's destination node, generally specified to be of a fixed or limited length. The other part is the packet's *header* or *metadata*. The packet header contains routing information which – along with local routing tables – helps nodes forward the packet properly along the path between its source and destination nodes, so that it can arrive safely in some guaranteed fashion. For a given routing scheme R, let the worst-case distance traveled by any packet from its source to its destination be D. Let the optimal (i.e., shortest) distance which that same packet could conceivably have taken (possibly given some better routing scheme) be d. The ratio $\frac{D}{d}$, then, is known as the *stretch* of R.

Consider a natural tradeoff between the size of local routing tables and the size of packet headers: on one extreme of this dimension, we have a network in which each node stores, in its routing table, destination information for every other node, leading to $\Omega(n \log n)$-sized routing tables, where n is the number of nodes in the network. In this model, packet headers need only contain the network label of their destination node (which requires $\lceil \log_2(n) \rceil$ bits to encode). On the other extreme of this tradeoff dimension, nodes store no data except their own network label, and each packet header contains complete routing information, telling each node where to send it next. In the worst case of this latter extreme, packet headers are $\Omega(n \log n)$ in size. Neither of these extremes scales well as a routing scheme for large networks.

A *compact routing scheme* is one which attempts to effect a useful tradeoff between the worst-case size of local routing tables and of packet headers. This problem has been well-studied (Cf. e.g. [4, 25]). For an overview and extensive list of references in this area, we refer the reader to Peleg [22]. The development of compact routing schemes which are *optimized to Internet-like* *topologies*, then, constitutes the ultimate focus of this thesis. (For a sample of previous work in this area, see Krioukov et al. [17].)

1.4 Thesis outline

The rest of this thesis is structured as follows:

Chaper 2 describes the DIGG (Dynamic Internet Graph Generator) project. DIGG serves as our codebase for graph simulation and analysis. Originally motivated by the need for expanded empirical analysis of dynamic graph algorithms, its mission was extended somewhat to suit the needs of the current work.

Chapter 3 describes our discoveries in power-law random graph topology, and their implications for power-law graphs in general.

Chapter 4 outlines a newly-proposed compact routing algorithm for Internet-like (i.e. power-law) graphs, combining previous compact routing results with our new observations in topology.

Chapter 5 suggests directions for future related research, including the greater mission of the DIGG project and some theoretical directions for topological studies of power-law random graphs.

Chapter 2

The DIGG project

The motivation for the Dynamic Internet Graph Generator project was to create software to simulate, evaluate and develop dynamic graph algorithms. Inputs to these algorithms are streams representing graphs which change over time; generally, these changes are visualized as the actions of some unknown agent outside the algorithm's foreknowledge or control. Dynamic algorithms maintain invariants like "can quickly report the diameter of the input graph upon request" or "the input graph is always biconnected." Data about the graph are learned and stored, then updated over time according to structural changes to the graph indicated in the input stream. Structural properties like k-connectedness might explicitly be preserved over time by the algorithm, through the deliberate manipulation of the graph in addition to the mutations indicated in the input stream.

For example, the concept of "currently active phone calls in Saskatchewan" could be modeled as a dynamic graph, with nodes representing active telephones, switches, and routers, appearing and disappearing in the graph as people place and terminate calls; graph edges representing point-to-point connections would evolve into and out of existence as physical connections are created and dropped.

Dynamic graph algorithms are typically memory-based, with the goal of performing some task using fewer computational resources than would be needed to repeatedly perform[1] that task (without any prior knowledge, i.e. memory) at successive instants in time. If, for instance, we want to discover (or estimate) the longest person-to-person network connection our Saskatchewanian friends have invoked at a given point in time, we could deploy a dynamic algorithm to keep watch over critical information about distances inside the network, updating the information regularly as the global structure of the connections mutates. Hopefully, using such a strategy, the task of reporting the longest connection length on-demand would consume fewer resources than recomputing all the necessary distance information from scratch to serve each request.

A large number of clever algorithms to maintain information (or structural properties) in dynamic graphs has been produced (see e.g. [15, 16] for examples). The time and space resources consumed by these algorithms are significantly smaller than those that would be needed to repeatedly invoke their static counterparts. However, since resource requirements are usually

[1] The rule prohibiting "split infinitives" is a hypercorrection dating from the 19th century, whence Victorian grammarians – having studied Latin and having rather ridiculously deemed it to be "the most perfect language" – arbitrarily decided that English word order and sentence structure should conform as closely as possible to that of Latin. Given that Latin infinitives are single words, these "proscriptive" (as opposed to "descriptive") grammarians created several entirely artificial rules and analytic structures – including the ban on so-called "split infinitives" – which have unfortunately survived to the present day, despite having absolutely nothing to do with the native Germanic morphology and syntax of the English language.

reported in asymptotic terms with hidden or unknown constants, the job of fully evaluating these algorithms isn't complete: empirical performance analysis is a prerequisite to any complete understanding of the benefits that these algorithms could provide in practice. Some work has been done in the experimental evaluation of selected algorithms [8, 13], but as far as we know, a generalized software platform for the simulation and analysis of dynamic graph algorithms has not been made available until DIGG.

There has been some limited research in experimental analysis of dynamic graph algorithms (cf. for example [8]), but it has generally relied on experiment-specific software, both for the implementation of algorithms and for associated data structures. We sought to decouple the two and provide a flexible, general codebase containing data structures designed specifically for such work. Following is a description of the current state of the DIGG package. Planned and possible extensions to the codebase are discussed in Chapter 5. Implementation details can be examined by retrieving the release packages, available at the DIGG project website http://digg.cs.tufts.edu/. All future stable releases will be available at this website as well.

2.1 Data structure layer

The DIGG code is written in ANSI C++. Our most elementary objects represent nodes and edges in an undirected multigraph. These can be arbitrarily weighted and colored, and are designed to store local topological information such as adjacency lists, degrees, etc.

The second level of object complexity, and the main level of data structure functionality, is an object representing the undirected multigraph itself. This structure creates, destroys and generally manages its constituent edge and node objects, as well as maintaining general structural integrity over time as updates are performed. It supports elementary update operations such as adding, deleting or altering nodes or edges. We can report elementary properties on demand, such as a graph's maximum degree, a graph's average degree, or the current number of edges in a given graph.

A time counter is built into each graph which can be controlled at the algorithm level (see next section) for arbitrarily granular analysis of dynamic algorithms over a simulated time scale. One of the main advantages of this feature is that it allows, when desired, the abstraction of "time" analysis away from indirect reporting based on CPU clock cycles.

Graphs can also perform more complicated update operations on themselves, such as the addition and deletion of random edges, or the addition of new edges to nodes selected at random which match specified criteria. Examples of this kind of operation include "attach an edge to a random node whose weight ≥ 3.8" and "link two random nodes whose degree ≤ 5." The availability of more complex updates allows for the flexible definition at the algorithm level of precisely what constitutes an "elementary operation." Extremely complex or significantly time-consuming operations – such as the computation of a given graph's diameter – are deferred to the algorithm level. Efficient implementations of some of the more useful complex operations will be published later in a separate library.

Wrapped around the graph object is what we refer to as a graph "manager" which represents the interface between the data structure layer and that of the experimental algorithm. In addition to providing transparent access to all graph functionality, a graph manager quietly supervises the creation and maintenance of XML files which record the evolution of a graph over time, as a dynamic algorithm is executed. The manager can be configured to record sequences of graph operations corresponding to the steps of a dynamic algorithm taken over time, with each operation recorded at configurable levels of abstraction. Examples of information which could be stored as operations in such a sequence include:

- add 5 random edges
- add 5 edges, specifically between node pairs $(1, 4), (3, 2), \ldots$
- received instruction to add 5 random edges; the actual edges which were added were between node pairs $(42, 108), (7, 11), \ldots$

In this way two goals are met. First, any results can be replicated exactly by repeatedly generating precisely identical graphs in a well-documented manner. Second, statistical analysis can be conducted by repeatedly executing randomized algorithms, and then reporting on aggregate results.

Each graph manager is also capable of recording a snapshot of its entire graph, at any point in time. These snapshots could for example be used to represent analytic breakpoints, e.g. "record a copy of the graph after each 100 steps of the algorithm have been completed." Lastly, graph managers can read XML files as input, reproducing predetermined command sequences or re-creating graphs which were constructed earlier.

2.2 Algorithm layer

On top of the structural level sits what we have chosen to call the algorithm layer, consisting simply of code which invokes and manages the evolution of a graph over time, using the graph manager interface. It is at this level that parameters are specified which govern the representation of a graph in XML for later analysis, publication or archiving. This is a primarily user-defined layer, and is intended to encapsulate code which is largely specific to each experiment. Also included at this layer is code common to more than one experiment, but too complex to be reasonably conceptually encapsulated inside the graph objects themselves. The code which we will continue to publish at this layer includes, for example, methods to calculate the diameter of a given graph, methods which determine whether a given graph is a forest, some power-law random graph "generators," and controller mechanisms for statistical analysis of pre-generated groups of graphs. More detail on the higher-level code which is relevant to this thesis will be provided in Chapter 3.

Chapter 3

Topology experiment

Despite the focus of our initial motivation – and our continuing high hopes for future applications – the first experiment conducted using DIGG was essentially non-dynamic.

3.1 Experimental goal

In his study of massive power-law random graphs, Lu [20] observed that under his model (cf. Chapter 1), four things relevant to our current pursuit were true with high probability[1]:

1. For all ranges of the power-law parameter β, a random power-law graph G_β has a unique giant component (meaning all other components have size at most $O(\log n)$).
2. When $0 < \beta < 2$, the size of the second-largest component of G_β — and hence of every non-giant component — is $\Theta(1)$.

[1] When we say a property R is true "with high probability" for a particular random n-node graph model, we mean that $[\mathrm{P}(R \text{ is false}) = o(n^{-1})]$.

3. When $0 < \beta < 2$, the diameter of the giant component of G_β is $\Theta(1)$. For this range of β, the giant component consists of a heavily interconnected, constant-diameter core and sparse "tree-like tails" of constant length hanging off of the core. The diameter of the core is at most 3, and the distance from any vertex in the giant component to the core is at most $\lfloor \frac{1}{2-\beta} \rfloor + 1$.

4. The highest-degree nodes in G_β are all inside the core of the giant component.

Considering the fact that we know how to perform optimal compact routing on trees [25], if these asymptotic statements prove relevant to graphs of practical size, then perhaps we can employ power-law topology to our advantage as a heuristic in tailoring a compact tree-routing scheme to power-law networks.

The focus of our initial experiment, then, is to generate sample sets of random power-law graphs and compare their structure to the asymptotic predictions outlined above.

3.2 Random graph model

Our simulations used a model which was essentially that of Aiello et al. [1], with some preprocessing based on ideas in Lu [20]. The full generator algorithm is as follows; compare with the two models in Chapter 1.

```
Input:  $n$ vertices from which to construct the target graph,
and $\beta \in \mathbb{R}^+$, a power-law parameter.
Output:  A random graph whose degree sequence is determined
by a power law with exponent $\beta$.
```

- Create a degree sequence of length n by associating degrees d_v with vertices v such that
  1. The number of vertices of degree 1 is $\lfloor e^{\alpha} \rfloor - r$.
  2. The number of vertices of degree k is $\lfloor \frac{e^{\alpha}}{k^{\beta}} \rfloor$ for all $k \in \{2, 3, \ldots, \lfloor e^{\frac{\alpha}{\beta}} \rfloor\}$.
  3. The remainder term r is $n - \sum_{k=1}^{\lfloor e^{\frac{\alpha}{\beta}} \rfloor} \lfloor \frac{e^{\alpha}}{k^{\beta}} \rfloor$.
  4. α is computed to minimize $|r|$.
- Construct the output graph from the degree sequence thus created:
  1. Form a set L containing $deg(v)$ distinct copies of each vertex v.
  2. Choose a random matching of the elements of L.
  3. Create the output graph, where the number of edges joining two vertices u and v is the number of edges in the matching of L joining copies of u to copies of v.

To view the complete code for our generator, please visit http://digg.cs.tufts.edu and examine the "AielloTopologyGenerator" object.

3.3 Data

We generated an XML library, available online[2], consisting of 800 instances of the above model. Parameters were varied so that 20 instance graphs were created using each of the members of the parameter domain $\{(n, \beta) | n \in \{1000, 2500, 5000, 7500, 10000\}$ and $\beta \in \{1.2, 1.3, \ldots, 1.9\}\}$.

[2] Please visit http://digg.cs.tufts.edu for a gzipped copy.

3.4 Method

Since our goal is to explore the advantages of using a tree-routing algorithm, it would be convenient if we could reduce the giant component to a forest by some simple, computationally cheap procedure. Recall that the highest-degree vertices in G_β are all located in the core of the giant component. Given that the diameter of this core is of length 3, we conjecture that the removal of the single highest-degree vertex from the graph, along with its 3-neighborhood[3], should leave only the "tree-like tails" mentioned above, which we can then directly examine to see just how "tree-like" they are. Formally, our experimental method is as follows:

- For each graph G_β in the dataset
  - Identify the giant component of G_β and delete the rest of the graph[4] to create a new graph G'_β.
  - Remove the highest-degree vertex from G'_β along with its 3-neighborhood.
  - Determine whether or not that which remains of G'_β is exactly a forest.

3.5 Results

We summarize here the outcome of the above procedure. Auxiliary results detailing some variations on this experiment can be found in Appendix A;

[3] By "the 3-neighborhood of a vertex x," we mean all vertices y such that $d(x, y) \leq 3$.

[4] At this range of β, all other components are of constant size, so the compact routing problem can be solved trivially for each non-giant component.

the tables there follow the same format as Tables 3.1 and 3.2, which describe our key results. Tables 3.1 and 3.2 can be decoded using the following information. Note that each sample set — representing one combination of n and β — consists of 20 graphs generated using the model described in Section 3.2.

Preprocessing statistics:

n is the graph-size parameter for the current sample set.
β is the power-law parameter for the current sample set.
μ_{GCsize} is the mean size of the giant component across the current sample set.
σ_{GCsize} is the standard deviation of the size of the giant component for the current sample set.
μ_{comp} is the mean number of graph components across all graphs in the current sample set.
σ_{comp} is the standard deviation of the number of graph components across all graphs in the current sample set.
$\%forest$ is the percentage of graphs in the current sample set which were exact forests before any changes were made to them.

Postprocessing statistics (measures taken after the giant component has been isolated and the highest-degree vertex has been extracted with its 3-neighborhood):

$\mu_{N'}$ is the mean number of nodes left in the graph; that is, the mean number of nodes which were originally in the giant component and which were not in the extracted neighborhood.
$\sigma_{N'}$ is the standard deviation of the number of nodes left in the graph.
$\mu_{comp'}$ is the mean number of components remaining after the isolation and dismemberment of the giant component.
$\sigma_{comp'}$ is the standard deviation of the number of components remaining after the processing of the giant component.
$\%forest$ is the percentage of graphs in the current sample set whose giant

components devolve into exact forests after processing.

It should be stressed that these results are certainly preliminary with respect to a complete empirical analysis of this sort of graph topology. Nevertheless, for these values of n, we can already observe that G_β appears to devolve very quickly into an exact forest after removing just one low-diameter neighborhood; that is to say, G_β as a class of graphs appears to converge rapidly to its predicted asymptotic topology. Please see Chapter 5 for a description of future plans in the area of detailed statistical/topological analysis.

Table 3.1: Experimental results for $1.2 \leq \beta \leq 1.9,\ n \in \{1000, 2500, 5000\}$.

n	β	$\mu_{GCsize} \pm \sigma_{GCsize}$	$\mu_{comp} \pm \sigma_{comp}$	$\%forest$	$\mu_{N'} \pm \sigma_{N'}$	$\mu_{comp'} \pm \sigma_{comp'}$	$\%forest$
1000	1.2	993.50 ± 3.27	4.20 ± 1.67	0.00%	10.85 ± 4.00	10.60 ± 3.80	100.00%
1000	1.3	985.50 ± 5.18	8.20 ± 2.65	0.00%	23.90 ± 4.68	22.95 ± 4.11	100.00%
1000	1.4	974.80 ± 5.00	13.20 ± 2.26	0.00%	48.85 ± 7.68	45.85 ± 6.64	100.00%
1000	1.5	963.40 ± 9.56	18.70 ± 4.52	0.00%	88.45 ± 14.58	81.20 ± 13.42	100.00%
1000	1.6	938.20 ± 11.08	30.70 ± 5.08	0.00%	141.00 ± 16.53	124.30 ± 16.20	100.00%
1000	1.7	907.00 ± 8.25	44.45 ± 3.69	0.00%	210.10 ± 26.04	173.30 ± 19.50	100.00%
1000	1.8	864.80 ± 15.78	63.45 ± 6.79	0.00%	295.80 ± 36.23	223.70 ± 18.87	100.00%
1000	1.9	821.55 ± 16.44	82.30 ± 7.05	0.00%	380.90 ± 62.24	239.00 ± 23.90	95.00%
2500	1.2	2489.75 ± 2.97	6.10 ± 1.45	0.00%	15.55 ± 3.99	15.50 ± 4.01	100.00%
2500	1.3	2478.50 ± 5.20	11.60 ± 2.50	0.00%	36.10 ± 6.60	35.30 ± 6.58	100.00%
2500	1.4	2463.60 ± 6.77	19.10 ± 3.39	0.00%	75.75 ± 11.00	73.35 ± 10.42	100.00%
2500	1.5	2434.60 ± 12.96	32.85 ± 6.34	0.00%	155.05 ± 16.09	146.85 ± 13.94	100.00%
2500	1.6	2389.05 ± 18.30	54.55 ± 8.65	0.00%	272.10 ± 21.83	249.00 ± 19.16	100.00%
2500	1.7	2316.15 ± 20.31	89.10 ± 9.42	0.00%	442.85 ± 49.78	387.40 ± 39.56	100.00%
2500	1.8	2235.95 ± 16.83	125.60 ± 8.11	0.00%	640.80 ± 61.39	522.10 ± 46.56	100.00%
2500	1.9	2133.75 ± 21.93	171.15 ± 9.96	0.00%	871.85 ± 80.03	632.25 ± 41.77	95.00%
5000	1.2	4984.10 ± 5.04	8.90 ± 2.51	0.00%	18.90 ± 5.33	18.80 ± 5.36	100.00%
5000	1.3	4975.90 ± 7.09	13.00 ± 3.46	0.00%	49.05 ± 6.57	48.45 ± 6.21	100.00%
5000	1.4	4999.90 ± 0.31	1.10 ± 0.31	0.00%	1.90 ± 1.29	1.90 ± 1.29	100.00%
5000	1.5	4993.30 ± 5.42	4.30 ± 2.58	0.00%	76.50 ± 11.08	74.30 ± 10.32	100.00%
5000	1.6	4948.35 ± 9.39	26.20 ± 4.38	0.00%	275.15 ± 19.77	258.40 ± 18.36	100.00%
5000	1.7	4836.40 ± 20.34	79.90 ± 9.93	0.00%	601.90 ± 38.50	548.00 ± 36.31	100.00%
5000	1.8	4657.00 ± 27.70	163.85 ± 12.49	0.00%	1036.75 ± 76.28	888.05 ± 57.34	100.00%
5000	1.9	4388.25 ± 29.70	285.05 ± 12.93	0.00%	1593.65 ± 128.43	1239.90 ± 92.21	100.00%

Table 3.2: Experimental results for $1.2 \leq \beta \leq 1.9,\ n \in \{7500, 10000\}$.

n	β	$\mu_{GCsize} \pm \sigma_{GCsize}$	$\mu_{comp} \pm \sigma_{comp}$	$\%forest$	$\mu_{N'} \pm \sigma_{N'}$	$\mu_{comp'} \pm \sigma_{comp'}$	$\%forest$
7500	1.2	7500.00 ± 0.00	1.00 ± 0.00	0.00%	0.00 ± 0.00	0.00 ± 0.00	100.00%
7500	1.3	7498.90 ± 1.77	1.55 ± 0.89	0.00%	9.90 ± 2.53	9.80 ± 2.53	100.00%
7500	1.4	7465.40 ± 6.39	18.05 ± 3.19	0.00%	107.15 ± 12.71	105.65 ± 12.74	100.00%
7500	1.5	7368.25 ± 17.88	66.00 ± 8.53	0.00%	305.75 ± 23.69	295.80 ± 22.73	100.00%
7500	1.6	7253.10 ± 19.04	121.25 ± 9.30	0.00%	620.15 ± 36.91	587.40 ± 35.37	100.00%
7500	1.7	7105.80 ± 25.76	191.50 ± 11.48	0.00%	1033.25 ± 69.45	945.80 ± 60.12	100.00%
7500	1.8	6893.90 ± 30.77	291.15 ± 14.52	0.00%	1638.35 ± 131.95	1419.60 ± 113.27	100.00%
7500	1.9	6608.65 ± 35.39	419.15 ± 15.15	0.00%	2317.85 ± 162.57	1849.85 ± 116.03	95.00%
10000	1.2	9999.30 ± 1.03	1.35 ± 0.49	0.00%	4.45 ± 2.35	4.45 ± 2.35	100.00%
10000	1.3	9972.45 ± 7.58	14.75 ± 3.75	0.00%	71.10 ± 10.17	70.45 ± 9.90	100.00%
10000	1.4	9921.75 ± 18.19	39.95 ± 8.95	0.00%	165.05 ± 18.66	162.65 ± 18.38	100.00%
10000	1.5	9848.15 ± 16.94	75.95 ± 8.49	0.00%	381.00 ± 29.43	369.35 ± 27.31	100.00%
10000	1.6	9705.50 ± 25.26	145.35 ± 12.48	0.00%	723.00 ± 69.89	687.20 ± 65.86	100.00%
10000	1.7	9500.75 ± 30.41	242.35 ± 14.14	0.00%	1311.40 ± 59.59	1205.05 ± 56.14	100.00%
10000	1.8	9226.50 ± 28.02	371.95 ± 12.20	0.00%	2122.55 ± 150.36	1864.50 ± 121.25	100.00%
10000	1.9	8857.40 ± 46.31	540.45 ± 21.14	0.00%	3089.90 ± 152.93	2500.90 ± 96.67	100.00%

Chapter 4

A compact routing scheme for power-law graphs

For a power-law graph with $0 < \beta < 2$ which has converged to its asymptotic topology, we present a simple compact routing scheme which achieves an additive stretch[1] of $(OPT + 5)$. As we mentioned before, all non-giant components in such graphs are of constant size, so for simplicity, we will *only discuss the giant component* in the following scheme; that is, for the purposes of this discussion, we are equating "G_β" with "the giant component of G_β". Routing on the smaller components is clearly trivial.

Our proposed scheme can be easily summarized:

[1] Some disambiguation is required here. On page 7, we defined the stretch of a routing scheme R to be the ratio $\frac{D}{d}$, where D was the worst-case distance a packet would travel from a node u to a node v using R's rules, and $d = d(u, v)$ is the distance between u and v in the input graph. (This is also known as "multiplicative stretch.") Here we are speaking of *additive* stretch, in which the worst-case transmission path between u and v *adds no more than a constant number of edges* to the actual graph distance $d(u, v)$ (which is here denoted "OPT").

I. Preprocessing phase:

1. `Select the highest-degree node` h `in` G_β `and grow a breadth-first-search tree rooted at` h. `Preprocess this BFS-tree according to the compact tree-routing scheme of Thorup and Zwick (see section 2.1 of [25]), using` h `as the root of the tree.`

   Please see Appendix B for a summary of this scheme. When the preprocessing is finished, each vertex in the giant component will have an $O(\log^2 n)$-bit BFS-tree label and an $O(\log n)$-bit routing table.

II. Routing phase:

1. `Route according to Thorup and Zwick [25].`

   See Appendix B for details.

Claim: *Given a power-law random graph $G_\beta = (V, E)$, with $|V| = n$, which has converged to its asymptotic topology, this compact routing scheme achieves an additive stretch of $(OPT + 5)$. That is, given $u, v \in V$ such that $d(u, v) = k$, a message from u to v which is routed using this scheme will travel along a path of length no more than $k + 5$.*

Support for claim:

We are employing a routing scheme which has optimal stretch (that is, stretch 1) for trees. The nodes in the giant component which aren't contained in the core are contained in tree-like subgraphs, which our evidence suggests are exactly trees. Assuming this is true, any non-optimal routing distance contributed by our scheme will be confined to the core of the giant component.

The diameter of the core of the giant component is no more than 3 [20]. The highest-degree vertex in G_β is within the core of the giant component [20]. Since we are rooting our BFS-tree at this vertex, the entire core of the giant component is contained within the first four levels of the BFS-tree (counting the root as the first level). The worst conceivable increase in routing distance over graph distance, then, occurs when the optimal path between two vertices contains two vertices u and v, both occurring on the fourth level of the BFS-tree, which are connected by an edge in G_β (see Figure 4.1). In this case, $d(u, v) = 1$, but our routing scheme takes a message from u to v through the root, along a path of length 6, for a total additive increase of 5 hops.

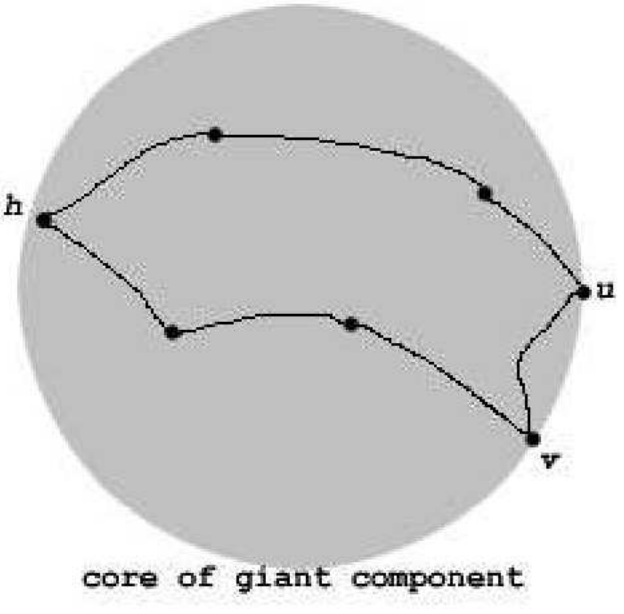

Figure 4.1: Worst-case constribution of our scheme to additive stretch: the highest-degree vertex in the giant component, h, which is the root of the BFS-tree used to route messages, is on the periphery of the core, and $d(u, h) = d(v, h) = 3$ while $d(u, v) = 1$.

Chapter 5

Future work

There's a massive amount of work remaining to be done even within the narrow boundaries circumscribing this work.

It would be useful to have some broad, large-scale, detailed empirical results carefully tracing the progression (as n increases) towards predicted asymptotic topological properties in random power-law graphs. Apart from identifying more precisely the context in which algorithms become useful when taking advantage of these properties, it seems likely that intermediate substructures or patterns of development would be discovered, fertilizing further theoretical investigation.

New theoretical results are of course extremely appealing in their own right, especially those focusing on the particular *mechanisms and phases of convergence* from chaos to reliable structure in random graphs as their size grows.

We've presented a compact routing scheme for power-law random graphs with $1.2 \leq \beta \leq 1.9$, but current estimates for the power-law parameter of the actual Internet are between 2.1 and 2.3. The diameter of the giant component of G_β increases to $\Theta(\log n)$ when $\beta \geq 2.0$, but we're hopeful that minor augmentations might eventually produce a compact routing scheme useful for ranges of β greater than 2. In addition, experimental analysis of the actual stretch of this algorithm on such graphs may yield good results without requiring any alterations to the algorithm itself.

The empirical analysis of *dynamic* graph algorithms clearly retains top mission priority.

The DIGG codebase needs to be expanded to include explicit support for simple undirected graphs, simple directed graphs, and directed multigraphs. Once functionality's been generalized, a revision pass will be needed to clean up issues with tight OO encapsulation and the functional integrity of the object hierarchy.

As an adjunct to the core DIGG code, we'd like to see the ongoing development and distribution of a library of more complex graph-processing algorithms or algorithmic components. Beyond the fairly elementary functionality of the graph objects themselves, we need reusable components which perform useful analytic tasks like the fast computation of graph diameter.

Appendix A

Auxiliary experimental data

In addition to removing a 3-neighborhood from around the highest-degree vertex in various instances of G_β, we experimented with removing 2-neighborhoods from around the 1, 2, 3, 4, and 5 highest-degree vertices in instances of G_β with $n = 1000$ and $n = 5000$. Results are summarized below.

Table A.1: Results for $1.2 \leq \beta \leq 1.9$, $n = 1000$, and the single highest-degree vertex extracted along with its 2-neighborhood.

n	β	μ_{GCsize}	σ_{GCsize}	μ_{comp}	σ_{comp}	$\%forest$	$\mu_{N'}$	$\sigma_{N'}$	$\mu_{comp'}$	$\sigma_{comp'}$	$\%forest$
1000	1.2	993.50	3.27	4.20	1.67	0.00%	307.80	21.33	288.45	19.22	100.00%
1000	1.3	985.50	5.18	8.20	2.65	0.00%	381.95	16.72	338.80	14.78	100.00%
1000	1.4	974.80	5.00	13.20	2.26	0.00%	474.05	25.55	383.00	14.29	95.00%
1000	1.5	963.40	9.56	18.70	4.52	0.00%	549.20	22.72	387.40	20.76	60.00%
1000	1.6	938.20	11.08	30.70	5.08	0.00%	598.95	21.39	351.45	22.81	30.00%
1000	1.7	907.00	8.25	44.45	3.69	0.00%	642.30	30.19	302.50	23.56	5.00%
1000	1.8	864.80	15.78	63.45	6.79	0.00%	668.65	16.40	238.65	38.19	0.00%
1000	1.9	821.55	16.44	82.30	7.05	0.00%	674.55	34.28	174.60	38.62	0.00%

Table A.2: Results for $1.2 \leq \beta \leq 1.9$, $n = 1000$, and the two highest-degree vertices extracted along with their 2-neighborhoods.

n	β	μ_{GCsize}	σ_{GCsize}	μ_{comp}	σ_{comp}	$\%forest$	$\mu_{N'}$	$\sigma_{N'}$	$\mu_{comp'}$	$\sigma_{comp'}$	$\%forest$
1000	1.2	993.50	3.27	4.20	1.67	0.00%	192.00	16.45	185.25	15.56	100.00%
1000	1.3	985.50	5.18	8.20	2.65	0.00%	255.70	15.29	238.60	14.19	100.00%
1000	1.4	974.80	5.00	13.20	2.26	0.00%	330.00	19.64	293.90	16.62	100.00%
1000	1.5	963.40	9.56	18.70	4.52	0.00%	400.85	21.54	330.40	17.06	100.00%
1000	1.6	938.20	11.08	30.70	5.08	0.00%	464.15	17.95	343.35	15.21	90.00%
1000	1.7	907.00	8.25	44.45	3.69	0.00%	518.50	22.37	329.90	16.08	55.00%
1000	1.8	864.80	15.78	63.45	6.79	0.00%	557.30	22.59	291.45	23.60	45.00%
1000	1.9	821.55	16.44	82.30	7.05	0.00%	580.90	33.75	243.60	30.00	10.00%

Table A.3: Results for $1.2 \leq \beta \leq 1.9$, $n = 1000$, and the three highest-degree vertices extracted along with their 2-neighborhoods.

n	β	μ_{GCsize}	σ_{GCsize}	μ_{comp}	σ_{comp}	$\%forest$	$\mu_{N'}$	$\sigma_{N'}$	$\mu_{comp'}$	$\sigma_{comp'}$	$\%forest$
1000	1.2	993.50	3.27	4.20	1.67	0.00%	140.95	12.88	136.75	12.29	100.00%
1000	1.3	985.50	5.18	8.20	2.65	0.00%	194.70	14.36	183.45	12.38	100.00%
1000	1.4	974.80	5.00	13.20	2.26	0.00%	255.25	15.95	231.80	15.21	100.00%
1000	1.5	963.40	9.56	18.70	4.52	0.00%	318.35	20.45	274.20	13.08	100.00%
1000	1.6	938.20	11.08	30.70	5.08	0.00%	383.60	18.57	305.85	13.10	95.00%
1000	1.7	907.00	8.25	44.45	3.69	0.00%	445.70	20.53	316.45	14.67	100.00%
1000	1.8	864.80	15.78	63.45	6.79	0.00%	477.00	23.50	296.80	14.66	80.00%
1000	1.9	821.55	16.44	82.30	7.05	0.00%	510.45	27.90	262.30	13.20	40.00%

Table A.4: Results for $1.2 \leq \beta \leq 1.9$, $n = 1000$, and the four highest-degree vertices extracted along with their 2-neighborhoods.

n	β	μ_{GCsize}	σ_{GCsize}	μ_{comp}	σ_{comp}	$\%forest$	$\mu_{N'}$	$\sigma_{N'}$	$\mu_{comp'}$	$\sigma_{comp'}$	$\%forest$
1000	1.2	993.50	3.27	4.20	1.67	0.00%	112.20	9.64	109.40	9.20	100.00%
1000	1.3	985.50	5.18	8.20	2.65	0.00%	157.90	13.44	149.95	11.79	100.00%
1000	1.4	974.80	5.00	13.20	2.26	0.00%	212.55	14.82	195.60	13.69	100.00%
1000	1.5	963.40	9.56	18.70	4.52	0.00%	269.35	20.25	236.90	15.04	100.00%
1000	1.6	938.20	11.08	30.70	5.08	0.00%	331.40	19.20	271.15	12.89	100.00%
1000	1.7	907.00	8.25	44.45	3.69	0.00%	390.05	13.83	292.20	14.37	100.00%
1000	1.8	864.80	15.78	63.45	6.79	0.00%	424.15	24.46	282.70	14.29	100.00%
1000	1.9	821.55	16.44	82.30	7.05	0.00%	458.15	34.99	262.85	16.76	85.00%

Table A.5: Results for $1.2 \leq \beta \leq 1.9$, $n = 1000$, and the five highest-degree vertices extracted along with their 2-neighborhoods.

n	β	μ_{GCsize}	σ_{GCsize}	μ_{comp}	σ_{comp}	$\%forest$	$\mu_{N'}$	$\sigma_{N'}$	$\mu_{comp'}$	$\sigma_{comp'}$	$\%forest$
1000	1.2	993.50	3.27	4.20	1.67	0.00%	92.50	8.99	90.05	8.58	100.00%
1000	1.3	985.50	5.18	8.20	2.65	0.00%	131.50	12.31	125.05	10.74	100.00%
1000	1.4	974.80	5.00	13.20	2.26	0.00%	182.00	14.85	167.95	12.91	100.00%
1000	1.5	963.40	9.56	18.70	4.52	0.00%	231.95	15.68	206.60	10.43	100.00%
1000	1.6	938.20	11.08	30.70	5.08	0.00%	287.80	19.01	240.30	13.45	100.00%
1000	1.7	907.00	8.25	44.45	3.69	0.00%	343.95	13.62	267.30	13.64	100.00%
1000	1.8	864.80	15.78	63.45	6.79	0.00%	381.20	20.08	266.45	11.98	100.00%
1000	1.9	821.55	16.44	82.30	7.05	0.00%	421.70	31.06	255.50	19.32	85.00%

Table A.6: Results for $1.2 \leq \beta \leq 1.9$, $n = 5000$, and the single highest-degree vertex extracted along with its 2-neighborhood.

n	β	μ_{GCsize}	σ_{GCsize}	μ_{comp}	σ_{comp}	$\%forest$	$\mu_{N'}$	$\sigma_{N'}$	$\mu_{comp'}$	$\sigma_{comp'}$	$\%forest$
5000	1.2	4984.10	5.04	8.90	2.51	0.00%	1421.95	45.31	1387.80	44.15	100.00%
5000	1.3	4975.90	7.09	13.00	3.46	0.00%	1838.85	66.99	1742.35	57.89	100.00%
5000	1.4	4999.90	0.31	1.10	0.31	0.00%	1213.45	63.67	1090.50	50.71	100.00%
5000	1.5	4993.30	5.42	4.30	2.58	0.00%	2129.80	83.76	1744.05	48.57	95.00%
5000	1.6	4948.35	9.39	26.20	4.38	0.00%	2848.30	80.98	2032.20	36.81	45.00%
5000	1.7	4836.40	20.34	79.90	9.93	0.00%	3324.60	67.14	1957.05	47.15	5.00%
5000	1.8	4657.00	27.70	163.85	12.49	0.00%	3580.75	84.96	1686.55	59.07	0.00%
5000	1.9	4388.25	29.70	285.05	12.93	0.00%	3683.25	67.54	1250.70	95.06	0.00%

Table A.7: Results for $1.2 \leq \beta \leq 1.9$, $n = 5000$, and the two highest-degree vertices extracted along with their 2-neighborhoods.

n	β	μ_{GCsize}	σ_{GCsize}	μ_{comp}	σ_{comp}	$\%forest$	$\mu_{N'}$	$\sigma_{N'}$	$\mu_{comp'}$	$\sigma_{comp'}$	$\%forest$
5000	1.2	4984.10	5.04	8.90	2.51	0.00%	898.90	36.18	886.60	35.03	100.00%
5000	1.3	4975.90	7.09	13.00	3.46	0.00%	1236.15	41.60	1198.65	40.52	100.00%
5000	1.4	4999.90	0.31	1.10	0.31	0.00%	636.15	33.98	603.55	28.98	100.00%
5000	1.5	4993.30	5.42	4.30	2.58	0.00%	1411.70	43.78	1270.10	38.25	100.00%
5000	1.6	4948.35	9.39	26.20	4.38	0.00%	2150.00	64.84	1790.10	46.36	95.00%
5000	1.7	4836.40	20.34	79.90	9.93	0.00%	2705.15	66.24	1988.70	40.38	75.00%
5000	1.8	4657.00	27.70	163.85	12.49	0.00%	3083.15	84.15	1899.85	40.65	15.00%
5000	1.9	4388.25	29.70	285.05	12.93	0.00%	3276.35	77.99	1595.80	57.91	0.00%

Table A.8: Results for $1.2 \leq \beta \leq 1.9$, $n = 5000$, and the three highest-degree vertices extracted along with their 2-neighborhoods.

n	β	μ_{GCsize}	σ_{GCsize}	μ_{comp}	σ_{comp}	$\%forest$	$\mu_{N'}$	$\sigma_{N'}$	$\mu_{comp'}$	$\sigma_{comp'}$	$\%forest$
5000	1.2	4984.10	5.04	8.90	2.51	0.00%	657.20	38.74	649.65	37.78	100.00%
5000	1.3	4975.90	7.09	13.00	3.46	0.00%	944.05	31.36	921.40	30.46	100.00%
5000	1.4	4999.90	0.31	1.10	0.31	0.00%	385.00	22.76	370.90	20.70	100.00%
5000	1.5	4993.30	5.42	4.30	2.58	0.00%	1075.70	37.26	994.35	35.51	100.00%
5000	1.6	4948.35	9.39	26.20	4.38	0.00%	1746.90	62.77	1524.90	47.17	95.00%
5000	1.7	4836.40	20.34	79.90	9.93	0.00%	2314.40	61.36	1833.90	44.89	90.00%
5000	1.8	4657.00	27.70	163.85	12.49	0.00%	2744.50	86.43	1894.30	38.74	60.00%
5000	1.9	4388.25	29.70	285.05	12.93	0.00%	2971.35	69.55	1715.80	35.61	30.00%

Table A.9: Results for $1.2 \leq \beta \leq 1.9$, $n = 5000$, and the four highest-degree vertices extracted along with their 2-neighborhoods.

n	β	μ_{GCsize}	σ_{GCsize}	μ_{comp}	σ_{comp}	$\%forest$	$\mu_{N'}$	$\sigma_{N'}$	$\mu_{comp'}$	$\sigma_{comp'}$	$\%forest$
5000	1.2	4984.10	5.04	8.90	2.51	0.00%	520.30	30.92	514.70	30.26	100.00%
5000	1.3	4975.90	7.09	13.00	3.46	0.00%	764.05	24.97	748.25	25.06	100.00%
5000	1.4	4999.90	0.31	1.10	0.31	0.00%	264.95	20.18	257.25	18.60	100.00%
5000	1.5	4993.30	5.42	4.30	2.58	0.00%	854.35	31.08	800.15	27.50	100.00%
5000	1.6	4948.35	9.39	26.20	4.38	0.00%	1472.00	59.11	1315.45	48.73	100.00%
5000	1.7	4836.40	20.34	79.90	9.93	0.00%	2042.85	60.61	1678.50	46.32	95.00%
5000	1.8	4657.00	27.70	163.85	12.49	0.00%	2456.10	79.74	1807.95	42.90	90.00%
5000	1.9	4388.25	29.70	285.05	12.93	0.00%	2710.70	80.11	1720.50	31.87	65.00%

Table A.10: Results for $1.2 \leq \beta \leq 1.9$, $n = 5000$, and the five highest-degree vertices extracted along with their 2-neighborhoods.

n	β	μ_{GCsize}	σ_{GCsize}	μ_{comp}	σ_{comp}	$\%forest$	$\mu_{N'}$	$\sigma_{N'}$	$\mu_{comp'}$	$\sigma_{comp'}$	$\%forest$
5000	1.2	4984.10	5.04	8.90	2.51	0.00%	427.70	27.41	424.10	27.66	100.00%
5000	1.3	4975.90	7.09	13.00	3.46	0.00%	643.50	27.60	630.75	27.82	100.00%
5000	1.4	4999.90	0.31	1.10	0.31	0.00%	193.05	16.11	188.05	15.25	100.00%
5000	1.5	4993.30	5.42	4.30	2.58	0.00%	712.30	27.13	671.70	22.66	100.00%
5000	1.6	4948.35	9.39	26.20	4.38	0.00%	1281.60	54.44	1159.50	46.16	100.00%
5000	1.7	4836.40	20.34	79.90	9.93	0.00%	1827.90	55.72	1535.95	49.61	95.00%
5000	1.8	4657.00	27.70	163.85	12.49	0.00%	2250.20	79.06	1711.60	45.46	95.00%
5000	1.9	4388.25	29.70	285.05	12.93	0.00%	2517.60	72.12	1684.00	33.37	80.00%

Appendix B

A compact routing scheme for trees

Please note that the only changes made to the following text from its appearance in Section 2.1 of [25] have been minor contextual and text-flow alterations. The conceptual content in its entirety was constructed and published by — and belongs to — Mikkel Thorup and Uri Zwick.

The text describes a compact routing scheme for trees. A breadth parameter for the scheme, b, can for our purposes be assumed to be 2.

——

We begin by describing, for every integer $b > 1$, a routing scheme that uses routing tables consisting of $O(b)$ words, and node labels (and therefore packet headers) consisting of $O(\log_b n)$ words. Each word here is $O(\log n)$ bits long, where n is the number of vertices in the tree. This scheme works with any assignment of port numbers [1].

[1] In this model, each endpoint of each edge is assigned a "port number" identifying it locally to its adjacent node.

The *weight* s_v of a vertex v is the number of its descendants in the tree. (A vertex is considered to be a descendant of itself.) A child v' of a vertex v is said to be *heavy* if $s_{v'} \leq \frac{s_v}{b}$, and *light* otherwise. In other words, the child v' is heavy if a fraction of at least $\frac{1}{b}$ of the descendants of v are also descendants of v'. Obviously, each vertex can have at most $b - 1$ heavy children. For convenience, we define r, the root of the tree, to be heavy. The *light level* l_v of a vertex v is defined as the number of light vertices on the path from r to v, including v if it is light.

We enumerate the vertices of the tree in depth first order, where all the light children of a vertex are visited before its heavy children, if any. We identify a vertex v with the number assigned to it by this enumeration, and let f_v be the largest descendant of v. We let h_v be the first heavy child of v, if it exists, or $f_v + 1$ otherwise. We let H_v be an array containing in its first element the number of heavy children that v has, and in its subsequent elements all the heavy children of v. Finally, we let P_v be an array containing in its first element $P_v[0]$ the port number corresponding the the edge from v to its parent, and then the port numbers corresponding to the edges from v to its heavy children. The routing information stored at v consists of (v, f_v, h_v, H_v, P_v), requiring a total of at most $O(b)$ words.

Each time an edge from a vertex to one of its light children is descended, the number of descendants in the corresponding subtree decreases by a factor of at least b. Thus, the light level l_v of every vertex v is at most $\log_b n$. Let $\langle v_0, v_1, \ldots, v_k \rangle$, where $r = v_0$ and $v_k = v$, be the path from the root of the tree to v, and let i_j , for $1 \leq j \leq l_v$ be the index of the j-th light vertex on the path. We let $L_v = (port(v_{(i_1-1)}, v_{i_1}), port(v_{(i_2-1)}, v_{i_2}), \ldots, port(v_{(i_{l_v}-1)}, v_{i_{l_v}}))$. In other words, L_v is an array of at most $\log_b n$ words containing the port numbers corresponding to the edges leading to the light vertices on the path from r to v. We then let $label(v) = (v, L_v)$ be the label of v. A packet addressed to v would carry $label(v)$ at its header.

The routing algorithm should now be obvious. Suppose that a packet with the header (v, L_v) arrives at w. If $w = v$, we are done. Otherwise, we check whether $v \in [w, f_w]$. If not, then v is not a descendant of w and the packet is forwarded to the parent of w using port $P_v[0]$. Next, we check whether $v \in [h_w, f_w]$. If so, then v is a descendant of a heavy child of w. We find the appropriate heavy child by searching H_w, and then obtain the corresponding port number from P_w. Otherwise, v is a descendant of a light child, in which case we use the port number given in $L_v[l_w]$. (We assume that the indices of L_v start from 0.)

Bibliography

[1] 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*, pages 171–180. ACM Press, 2000.

[2] A. Akella, S. Chawla, A. Kannan, and S. Seshan. Scaling properties of the internet graph. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 337–346. ACM Press, 2003.

[3] N. Alon and J. Spencer. *The Probabilistic Method*. Wiley Interscience, New York, 1992.

[4] M. Arias, L. Cowen, and A.K. Laing. Compact roundtrip routing with topology-independent node names. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 43–52. ACM Press, 2003.

[5] A. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[6] J.A. Bondy and U.S.R. Murty. *Graph theory with applications*. Elsevier North-Holland, New York, 1976.

[7] T. Bu and D. Towsley. On distinguishing between internet power law topology generators. In *Proceedings of IEEE INFOCOM*, pages 1587–1596, June 2002.

[8] Camil Demetrescu, Stefano Emiliozzi, and Giuseppe F. Italiano. Experimental analysis of dynamic all pairs shortest path algorithms. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 369–378. Society for Industrial and Applied Mathematics, 2004.

[9] Benoit Donnet, Philippe Raoult, Timur Friedman, and Mark Crovella. Efficient algorithms for large-scale topology discovery. In *Proceedings of ACM SIGMETRICS*, June 2005.

[10] P. Erdős and A. Rényi. On random graphs. In *Publicationes Mathematicae*, volume 6, pages 290–297, 1959.

[11] P. Erdős and J. Spencer. Probabilistic methods in combinatorics, 1974.

[12] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

[13] D. Frigioni, T. Miller, U. Nanni, and C. Zaroliagis. An experimental study of dynamic algorithms for transitive closure, 2001.

[14] W. Hays. *Statistics*. Harcourt Brace, Orlando, fifth edition, 1994.

[15] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge and biconnectivity. In *Proceedings of the 30th ACM Symposium on the Theory of Computing (STOC)*, pages 79–89, 1998.

[16] V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *IEEE Symposium on Foundations of Computer Science*, pages 81–91, 1999.

[17] D. Krioukov, K. Fall, and X. Yang. Compact routing on internet-like graphs, 2003.

[18] X. Liu and A. Chien. Realistic large-scale online network simulation. November 2004. Submitted to *Proceedings of the ACM Conference on High Performance Computing and Networking (SC2004)*.

[19] D. Lu and P. Dinda. Synthesizing realistic computational grids. In *Proceedings of the 15th ACM/IEEE SuperComputing (SC 2003)*, November 2003.

[20] L. Lu. The diameter of random massive graphs. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 912–921. Society for Industrial and Applied Mathematics, 2001.

[21] C. Papadimitriou. Algorithms, games, and the internet. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 749–753. ACM Press, 2001.

[22] David Peleg. *Distributed computing: a locality-sensitive approach.* Society for Industrial and Applied Mathematics, 2000.

[23] L. Subramanian, S. Agarwal, J. Rexford, and R. Katz. Characterizing the internet hierarchy from multiple vantage points. In *Proceedings of IEEE INFOCOM 2002*, June 2002.

[24] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network topologies, power laws, and hierarchy. Technical Report

01-746, Computer Science Department, University of Southern California, June 2001.

[25] M. Thorup and U. Zwick. Compact routing schemes. In *Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 1–10. ACM Press, 2001.

[26] D. West. *Introduction to Graph Theory.* Prentice Hall, Upper Saddle River, NJ, 1996.