Large Graph Sampling and Visualization

Ryan Kelly
Dept. of Computer Science
Carnegie Mellon University
rkelly@cs.cmu.edu

Amit Manjhi
Dept. of Computer Science
Carnegie Mellon University
manjhi@cs.cmu.edu

H. Brendan McMahan
Dept. of Computer Science
Carnegie Mellon University
mcmahan@cs.cmu.edu

April 24, 2003

1 Introduction

Many large graphs exist in today's digitized world: the hyper-text graph of the world wide web contains several hundred million nodes and about a billion edges; the Internet topology can be thought of as another large graph with about a million nodes at the router level; the phone network can be thought of as a huge graph showing who is calling whom; numerous other examples of social networks also exist. These graphs are so big that standard techniques provide little insight into the data, yet these graphs all seem to share some structural properties [23]. Discovering and analyzing these properties and finding ways to work with such large graphs will be an important research goal for some time.

There are many technologies that could make large graphs easier to work with: improved algorithms, improved hardware, better visualization tools, etc. In this project, we hope to show that in some cases the need to work directly with very large graphs can be avoided by instead finding a representative sample and working with this smaller sample graph instead. We propose five sampling algorithms and evaluate them in terms of how well they preserve certain properties of the original graph. We also subjectively evaluate them via the use of several visualization tools.

The ability to effectively work with large real-world graphs has numerous applications: in social networks, for criminology and law enforcement, in analysis of virus propagation patterns, and in helping us understand the structure and evolution of the Web. Finding structure in the hyper-text graph can result in efficient and effective algorithms for searching [21], understanding the Internet topology can help us design better topology generators which can lead to better network algorithms and configurations, and studying social networks can be useful in detecting money-laundering rackets and terrorist organizations. We hope to take a small step toward enabling this kind of study in this project.

The rest of the paper is organized as follows. In the next section, we survey the related work. In section 3, we elaborate our proposed solution, and the intuition behind it. Section 4 has the results for the experiments we conducted, and finally, we conclude in section 5.

2 Survey

There has been a lot of work in the related areas of modeling large real-world graphs and finding their essential characteristics; see [23] for an introduction to this topic. In particular, we have focused on similar work in the area of social network graphs, the Internet topology graph, and the graph induced by the hyper-link connections among web-pages. We also cite some work in the related area of *Property Testing.* In a later section we address the algorithmic and data storage issues that arise when computing with these graphs, and discuss some methods for addressing those issues.

In social networks, the earliest work dates back to 1960's, when Stanley Milgram was able to demonstrate that short chains of acquaintances link people in the United States who do not know each other. This led to the now familiar folk-lore of *six degrees of separation* in social networks, and the follow-up work on the *small-world phenomenon* [26] by Jon Kleinberg, which took an algorithmic view of things. Kleinberg raised the following question: even if a short chain of acquaintances existed between people, how are arbitrary pairs of strangers able to find it using just local information? He, in turn, came up with a network model that improved upon existing models, in which it was possible to find good paths using just local information.

In a seminal paper, Faloutsos et al. observed that the degree distribution of Internet topology graphs follow a power-law [17]. This held for both the autonomous-system level topology as well as the router graph of the Internet. Moreover, they found that the property was preserved, despite the growth of the Internet. This paper led to a complete upheaval in the area of network topology generators. The existing hierarchical generators (which modeled the seemingly strict hierarchy in the Internet) were replaced by Inet [42] and similar topology generators, which modeled the Internet as power-law random-graphs. More recently, in 2002, Tangmunarunkit et al. [38] concluded that the loose hierarchy present in the Internet more closely resembles the loose hierarchy present in power-law random graphs rather than the strict hierarchy modeled by structural generators of yore.

Tauro et al. [39] came up with a simple intuitive model of the Internet topology, and argued by way of empirical validation that a conceptual way to think of the Internet is as a jellyfish-like structure.

The hyper-link structure of web documents has also seen a slew of work recently. Broder et al. [11] consider the directed graph whose nodes correspond to static pages, and whose arcs correspond to links between pages. They confirm the previous reports that the in-degree and out-degree distribution of links follow the power-law. More importantly, their studies reveal that the Web graph is like a bowtie with four components, all of which are roughly the same size. The center of the bowtie is a strongly connected component SCC (given any two nodes u and v, it is possible to find a directed path from u to v), and on the either flank, there are the IN and the OUT components. They also find that the diameter of SCC is at least 28. Dill et al. [16] discovered self-similarity in the Web graph. They made measurements over various subsets of the web and found that many structural properties of the Web apply to significant subsets as well. Kleinberg et al. [27] and Brin et al. [9] give algorithms that operate on the Web graph from a search engine point of view. These algorithms, HITS and Page Rank, have formed the basis for Teoma [40] and Google [21], two of the most effective search engines currently available.

There has been some recent work on graph sampling that has an objective similar to our own. In [28] the authors employ several methods for graph reduction and apply them iteratively as long as the characteristic properties of the graph are preserved. Based on this empirical approach, they are successfully able to reduce the size of real graphs by 70%. Our work has considered a wider range of algorithms, and instead of the emperical iterative approach, we have tried to exploit the structural properties of the real world graphs. On the particular dataset that we consider, our results improve upon their limit of 70% improvement. In another recent work, Chakrabarti et al.[12] try to characterize real graphs and propose a *recursive matrix* model that can quickly generate realistic graphs.

Finally, the area of *Property Testing* [20] tries to answer the following question: given a large graph G, and a property P (for example, whether G is bipartite or not), is it possible to find out whether G satisfies P or not by running the tester (some algorithm which tests whether a graph satisfies some property P or not) on a carefully chosen subgraph of G? The naive solution of running the tester on G is presumed to be computationally unfeasible. Previous work in the area has dealt with deciding whether a large graph is bipartite or not. This property has the helpful characteristic that if the original graph is bipartite then all sub-graphs are bipartite, which means only one-sided error is possible.

There has been a good deal of work on general graph visualization, including a yearly symposium (Graph Drawing) and various publications [15]. More specifically, some of the work has been devoted to the visualization of large graphs. Pajek [8] is software intended for large graph analysis, and it also can draw large graphs quickly by utilizing recursive factorization. This is the process of clustering a large graph as a collection of smaller, simpler graphs which may be treated separately or in combination by other algorithms. Tamara Munzner describes three systems[34] for graph visualization, each designed for a different purpose. The H3 system can draw hyperlink graphs of over 300,000 nodes while providing interactivity with the user, the Constellation system can draw large semantic networks to aid linguists, and the Planet Multicast system can draw a 3D map of the internet backbone around the globe. These systems have all been tailored for specific purposes, but can all render the relevant moderately large graphs in a useful manner for those purposes. The H3 system has a close cousin Walrus, developed under CAIDA, which has a different interface, but is also built on Munzner's published H3 algorithms.

CAIDA's early work in graph visualization produced Otter, which is a general purpose graph visualization tool lacking many of the features of more current systems. However, it is still used in interesting ways on large graphs. Other visualization tools focus on illustrating dynamic systems as animated graphs. This includes NAM and Javis, the C and Java implementations of graph visualization tools mainly used for packet flow analysis. A newer Java system which is built to support large graphs is the Graph Visualization Framework[31], and its included application Royere.

Also for drawing general large graphs, Gajer et al. have created an algorithm[19] "GRIP" which first embeds the graph in a Euclidean space E of any dimension and then projects this space on a two or three dimensional subspace of E for rendering. The result is a smooth and symmetric visualization which is produced quickly, even for graphs with tens of thousands of vertices. Other systems include Graphviz, which is a suite of tools from Att Research, and NicheWorks[41], which is a large graph visualization tool which can handle weighted graphs and interactivity.

2.1 Algorithmic and Data Storage Issues

Problems that are easy on small graphs or even on any graph that fits entirely into main memory may become much more expensive when applied to very large graphs that will not fit in main memory. This problem has led to a large body of research addressing the need for external graph algorithms: graph algorithms that do not require the whole graph be represented in main memory. As expected, I/O costs for such algorithms dominate other costs, and so minimizing disk I/O is the primary focus. This issue was first addressed in a seminal paper by Chiang, et al[13]. This paper introduced a number of new methods for developing and analyzing algorithms on very large graphs. Many others have extended and improved on the initial results of this paper. The buffer tree algorithm [4] for external memory problems led to some straightforward algorithms for several graph problems. A functional programming approach to external graph algorithms is described in a new paper, [1]. This approach has some theoretical and performance advantages.

A paper be Kuman and Schwabe [29] gives two new efficient external-memory data structure implementations, for the heap and tournament tree, and uses these two structures to design algorithms for minimum spanning trees, breadth-first search, and single-source shortest path. Improved results for minimum-spanning tree and single source shortest path appear in [6].

Efficient algorithms for depth-first search and breadth-first search on sparse graphs proved particularly elusive, but recently progress has been made. In [7], algorithms for I/O efficient depth first search in externally stored undirected embedded planar graphs are explored, and an external algorithm for BFS requiring only sublinear I/O for sparse undirected graphs [32] has also been developed. An improved external priority queue implementation is presented in [5], which the authors then apply to the BFS, DFS, and minimum spanning forest problems.

Another approach to dealing with very large graphs is to try to increase the relative amount of information about the graph that can be stored in main memory via compression schemes. Such approaches for the web graph are explored in [37] and [10]. A straightforward implementation of a web graph compression scheme which takes advantage of the standard UNIX tools **sort**, **gzip**, and **bzip** tools is described in [22]. The Connectivity Server is a special-purpose database developed at Compaq's Systems Research Center that provides fast access to web graph data. It relies on very large main memory and compression schemes to try to keep the entire web graph in memory. The Link Database [36] component of this system provides efficient access to link structure. It's implementation takes advantage of both general purpose compression techniques and techniques particular to the structure of the web graph.

The research on external graph algorithms outlined above, while directed at very large graphs, makes relatively few assumptions about the structure of such graphs. Most of the algorithms depend on the edge density of the graph in obvious ways, but no assumptions are made about the degree distribution of the graphs. One exception to this trend is [30], which investigates some I/O efficient algorithms for bounded treewidth graph. It remains an open question to determine if the power-law structure of many real-world graphs can be exploited to created better external algorithms.

3 Proposed Methods

In this section, we propose five classes of algorithms for effectively sampling large graphs. Building on the related work (Section 2), we want to explore the problem of effectively data-mining large graphs. We first propose some techniques for graph sampling and outline several properties that we believe characterize real graphs and should be preserved. Next, we give details of our work in the area of visualization. For easy comparison and quick reference, we tabulate the features of all existing tools that we could find, and discuss which tool is most relevant for our purposes. Finally, we consider issues relating to the representation of graphs; though we did not implement any of the ideas discussed in this section, they remain a promising direction for future work.

3.1 Sample Algorithms

Given a large graph $G = (V, E)$ (we assume G is undirected unless mentioned otherwise), we want to come up with a sub-graph $G' = (V', E')$, such that G' is small (possibly not greater than log of the size of G), and it preserves the essential properties of G. Then, instead of working with G, we can work with G' (data-mining techniques for small graphs would apply) and hope to get similar results. Moreover, we want that the computation involved in arriving at G', given G, should not be large.

A natural way to get G' from G is to employ a sampling algorithm S to sample G, in a way that preserves most of the properties that characterize G, or at least the properties of interest for some application. A post-processing step would be to verify if G and G' are indeed similar with respect to the properties that characterize G. Therefore, we need to specify properties that we think characterize real-world graphs. First, we discuss the sampling algorithms that we have implemented and tested. We use abreviated names for these algorithms in our result plots and elsewhere in the text. These abbreviated names are summarized in Table (1).

3.1.1 Forest Fire (ff)

This algorithm was motivated by propigation methods in natural graphs. In particular, consider two different cases: how fires in the forest spread, or how infections spread in human networks.

In case of forest fires, first a single tree starts burning. Subsequently, its neighbors may catch fire, and finally, most of the combustible material in the forest is engulfed. As long as a tree burns, it has some probability p of spreading the fire to its neighbors. The extent to which the fire spreads is very much dependent on the starting node - if the starting node has very few neighbors, or if all the neighbors are far-off, the fire would die automatically after some time. The number of trees burnt down by the fire is also a function of how bushy the whole forest is. In addition, there are other phenomena at work. A tree is more likely to catch fire if it is surrounded by well-connected trees (trees that have many neighbors), and the probability goes up as more and more of these neighbors start burning.

Similarly, an infection in a human network starts in the same innocuous random way. An infected person is likely to infect her friends, neighbors, or acquaintances. If any of the infected persons meets lots of people, and travels a lot (i.e., is well connected), the infection is more likely to affect many people. This phenomena is also different from the forest fire in some important ways.

With infections, there can be carriers, that is, people who have a high probability of infecting their neighbors but suffer none of the negative effects of the disease directly, and second, people can become immune to disease after some exposure to the germs.

Based on the above, we have implemented a sampling algorithm that works similarly. In this subsubsection, we use sampling and infection analogously. For ease of explanation, let our graph consist of *blue*, *white*, *black* and *green* nodes, respectively denoting nodes that are susceptible to being sampled, nodes that already have been sampled and no longer affect their neighbors, nodes that have been chosen in the sample and are actively affecting their neighbors, and nodes that have developed resistance to sampling. Similarly, the edges are either *blue*, *green* or *white*, respectively denoting edges through which sampling can proceed, edges that are resistant to sampling, and edges that are in the sample but the infection can no longer spread through them. The inputs to our sampling algorithm are p (the probability of a node infecting its neighbors), and the number of steps *dur* after which the propagation stops. Initially, all nodes and edges in the graph are *blue*. Any node remains *black* for only one time unit. After that, it converts to *white*. (The analogy to think of is a tree which can spread fire only when it is burning). Similarly, the propagation can propagate at most once through an edge. Our sampling algorithm can be easily described in time steps. At time 0, a node, say u randomly turns black. In the next time unit, it tries to convert its neighbors (connected by *blue* links) to *black* with probability p. If it succeeds, the edge becomes *white* and the node, v, becomes *black* and remains so for one time step. In addition, all other edges incident to v immediately turn *green*. (Thus, if our sampling method can be thought of as generating a directed graph, all nodes have in-degree 1). If it fails, only the edge becomes *green*. In the next time unit, all current *black* nodes try to spread the infection. This process goes on for *dur* time-steps. Finally, the sample includes all *white* and *black* nodes, and all *white* edges.

In our simple implementation, we did not have *green* nodes. One simple way to have *green* nodes, based on how humans respond to infections, would be to have a *black* node turn green, when a certain number of its incident edges have turned green. Another aspect missing from our current implementation is that there is neither an addition of probabilities (i.e., if two nodes are infected, then the chances that their common *black* neighbor is infected is the sum of the two infection probabilities), nor are the infections probabilistic (i.e., all nodes have probabilities between 0 and 1 of being infected and later a thresholding function is used to pick the nodes in the sample).

Moreover, this way of sampling can make it computationally easier to implement some simple sampling algorithms. For example, if in the above case a node and its link turns *green* after just one unsuccessful try, then it is equivalent to *random node sampling*, discussed in a later subsubsection. This method just imposes an ordering on the sampling, and hence it is easier to keep track of the sampling method. We also feel that it is more intuitive.

3.1.2 SVD sampling (svd)

Eigenvalues and eigen vectors of adjacency matrices of graphs are believed to represent their underlying structure. The largest eigen values and eigen vector model the principal components of a matrix. In this sampling method, we capitalize on this idea. Recently, distribution of eigenvalues of random power law graphs have been examined by Chung et al. [14].

Singular value decomposition of a matrix is defined as decomposing a matrix A into

$$[A]_{m\times n} = [U]_{m\times r}[S]_{r\times r}[V']_{r\times n}$$

such that U and V are comprised of orthonormal vectors, and S is a diagonal matrix consisting of the eigen values of A. A graph G=(n, m) can naturally be represented as an $n \times n$ matrix with only m non-zero entries. If the graph is not weighted, normally all the non-zero entries are 1.

In this implementation, we first obtain the SVD for the adjacency matrix of the given graph. Then, we transform S to S_1 by making all but its first k diagonal elements 0 (for the samples described later we used k=6, 12 and 25). Using S_1, we compute A_1 as $U \times S_1 \times V'$. Finally, we keep only those entries of A_1 that are greater than or equal to 1. We implemented this algorithm in Matlab on top of RMAT.

3.1.3 Random node sampling (rns)

We implemented the naive random node sampling algorithm as a baseline for comparison against our other methods. We sample V uniformly and at random with probability p ($0 \leq p \leq 1$) to produce a smaller subset V'. E' consists of all edges, both the end-points of which are in V'. More formally, $(u, v) \in E'$, iff $(u, v) \in E$, $u \in V'$, and $v \in V'$.

Lemma 1 *RNS preserves the power-law on the degree distribution.*

Proof 1 *The power-law on the degree distribution states that (***Power-Law 2** *of [17]) the frequency, f_d, of an outdegree, d, is proportional to the outdegree to the power of a constant, α. Let, $f_d = kd^\alpha$ $\forall$ degrees d in G. Let us denote the number of nodes that have degree d in G' by f'_d. If the sampling probability is p, then in G', $E(f'_d) = f_{d/p} = kp^{-\alpha}d^\alpha = k'd^\alpha$. This simply means that on a log-log scale, the curve of f_d against d is shifted to the left.*

Thus, RNS preserves the power-law on the degree distribution in the expected case. Krishnamurthy et al. [28] have also independently tried this technique and found it to be the best among all the sampling techniques they have considered.

Abbreviation	Description
mnc	Metis Node Collapsed Sample
mp	Metis Single Partition Sample
svd	Singular Value Decomposition based Sample
ff	Forest Fire Model Sample
ps	Path Sampling based Sample
rns	Random Node Sample

Table 1: Summary of Graph Sampling Algorithms Tested

3.1.4 Path Sampling (ps)

An important node is likely to be on many paths, at least for some applications. Thus, a good way to sample a graph such that its important properties are preserved is to sample paths of the graph.

Following this intuition, we randomly picked a node and tried to generate a path from it by randomly following one of its out-going nodes. The sample includes all the edges picked. The algorithm samples paths of a fixed length, though some paths may terminate after shorter lengths at nodes with out degree zero. Paths are sampled repeatedly until we attain a sample size close to the required sample size. Later, we post-process the output so that it does not contain any duplicate edges.

3.1.5 Graph Partitioning and Metis

A *cut* in a graph G is a set of edges whose removal results in G being broken into two or more components (disconnected). We originally proposed to determine empirically if one or more of these resulting components preserve important characteristic of real-world graphs. We planned to investigate using min-cuts and then recursively cutting the resulting components until we produced a sample of the desired size. The problem of finding a minimum cut in a graph, that is, the smallest set of edges that disconnects a graph, is a well-studied tractable problem. However, the min-cut problem makes no restrictions on the sizes of the resulting connected components; it might result in only a single vertex being disconnected. Such a cut would not be useful for our purposes. The real question is how to determine a better the cut to use.

A more general problem is the k-way graph partitioning problem. Here the goal is to partition the nodes of a graph into sets of roughly equal size, while minimizing the number of edges between nodes of different partitions. While this problem is NP-Complete, efficient approximation algorithms have been developed that give good partitions for many real word graphs. We have adopted the algorithms from [24, 25] as they are implemented in the software package Metis. We develop several graph sampling algorithms based on having access to an approximation algorithm for the k-way graph partitioning problem. The k-way cuts provided by such an algorithm give rise to more intuitive sampling methods than min-cut techniques.

3.1.6 Single Partitions as Samples (mp)

Suppose we are given a graph G with n nodes, and want to produce a sample on about n/k nodes. We can produce such a sample by partitioning G into k sets, and then taking the subgraph induced by the nodes in one of those sets as our sample. The intuition for this is that in a graph with a recursive structure we will produce a similar graph. The partitioning algorithm will try to make the partitions as dense as possible, and this will make sense in certain domains. For example, consider a graph representing a network topology containing a hierarchy of servers and other machines, where there is a central server with very high degree. Partitions in such a graph might logically correspond to densely-connected sub-networks of the overall network, where the sub-networks have a similar hierarchical structure. On such a graph this algorithm should produce a useful sample. In general, this sampling mechanism should produce graphs that capture the local structure of the overall graph, though some global structure may be lost.

In our experiments we just pick one of the k-partitions at random to be our sample. However, if it is known that a particular measurable property should be maintained by the sample, then all k partitions could easily be tested and the most suitable sample chosen. In particular, the edge density of the different samples may very greatly.

We implemented this algorithm as a Java wrapper around Metis that handles pre- and post-processing. In particular, the input is made undirected and stripped of any self-loops. Metis then generates a partition which it outputs as a file assigning a partition number to each vertex. We then use this partition file to construct the actual subgraph in a representation that can then be processed by any of our other sampling algorithms and similarity measure calculations.

3.1.7 Vertex-Collapsed Partitions as Samples (mnc)

The next sampling algorithm based on graph partitioning can be though of as a dual of the single-partition algorithm just described. Again, suppose we have a graph G on n nodes and we want a sample of size n/k. We can run our graph partitioning algorithm on G and divide it into n/k partitions. We then collapse all the nodes in each of these n/k partitions; these collapsed nodes are the nodes in our sampled graph. Two nodes in the sampled graph G' are connected if any two original nodes in the corresponding partitions are connected in the original graph.

This graph partitioning algorithm may not capture the local structure of the original graph, but hopefully it will do a better job of preserving the overall topology of the graph than other methods. This might be useful for simulating network algorithms, for example, where a simpler model can be used to describe the behavior of the algorithm on the dense sub-networks represented by a single node in the sampled graph.

If the sample size is small, the naive method of generating edges for this graph will produce very dense graphs. This can be remedied by implementing the following scheme: generate edges as described, but weigh each edge by the number of edges in the original graph going between the two corresponding partitions, and then perturb each of these weights by a small random amount. Remove low-weight edges until the desired edge-density is achieved. Other edge removal techniques could be considered, such as a purely random scheme, but we believe that this technique would do the best job of preserving the overall topological structure of the graph, which is the primary goal of this sampling method.

As with the single vertex method, we implemented this algorithm as a wrapper around Metis. The post-processing is slightly more complicated as it requires performing the actual vertex-collapses to produce the sample graph. We implemented the edge weighting and thresholding scheme as well, but did not have time to test it's performance.

3.1.8 Other Ideas

Here, we list ideas that we could not implement due to lack of time, but believe to be promising.

- Reverse engineer the generative model: Several researchers argue that the structure of real-world graphs are a cumulative result of a sequence of local changes and offer evolutionary models to explain the different observed properties of real-world graphs. If based on the real data-sets, we are able to find the probability of various local events that give rise to the final data-set, we can undo the local changes one by one and hope to get a smaller sub-graph, which is representative of the original. We also hope that, similar to the findings of [17], where they conclude that the power-laws hold despite a 45% growth in the size of the Internet, the smaller sub-graph obtained by performing a sequence of local undo events will retain the structural properties of the original graph.

  We plan to use the model by Bar*á*b*á*si et al. [3], which improves on its predecessors on two main counts. First, it accounts for the fact that most real-world graphs continue to evolve by addition of new links, new nodes and rewiring of links. Second, it incorporates the rich-get-richer phenomena, evident by the fact that Web pages that already have many links are more likely to acquire more links. The model starts with n vertices and no edges - at every step, the graph probabilistically decides either to add a single new vertex and m edges; all the new edges link the new vertex to some of the vertices already present, or rewire m-edges, or to add m new edges. In all cases, the probability that a given vertex will receive a new edge is proportional to the share of the total set of edges that the vertex already owns; hence the well-connected become still better-connected. Growing according to these rules, the graph attains a statistical steady state - the shape of the distribution of node degrees does not change over time. Depending on the frequencies of the three local processes mentioned above, the distribution is either a power-law with the exponent a function of the frequencies, or, it approaches an exponential. In addition, some information can be obtained if we know where the origin of the data-sets. For example, there can be no rewiring in the actor graph [3].

Based on the above observations, we should be accurately able to work out the probabilities of each of the local events, and get a good sample of the real-world graphs by undoing the steps one by one.

- A modification of the RMS algorithm might also be worth investigating: pick a random subset of nodes $V` \subseteq V$, and then connect two nodes u and v in the new graph G' if they are within a distance d in G'. Choosing d appropriately is obviously an important issue.

- If the graph is self-similar [16], our task is slightly easier. One possible option is to remove the central core of the Web graph, and look at one or more of the resulting connected components. The issue then boils down to finding which connected components most resemble the original graph G.

3.2 Similarity Measures

The following properties seem to characterize real-world graphs and we have used them to measure the similarity of G with its sub-graph G'.

- **Power-law on the degree distribution**: Real graphs have a distribution of degrees that follows power-law. The number of vertices of degree d is proportional to $d^{-\alpha}$. Faluotsos et al. [17] observed power-law relationships at both the router level as well at the AS level in the Internet topology. Kleinberg et al. [27] show that the in and out-degrees of nodes in the Web graph follow a similar distribution. The values of α in these graphs range between 2 to 3.

- **Hop exponent**: Many real graphs [17] have a neighborhood function, $N(d)$, that follows power law ($N(d) \propto d^H$). $N(d)$ is defined as the number of pairs of nodes in G within distance

d. Palmer et al. [35] argue that the exponent, H, called the hop-exponent is a very good metric for measuring the similarity of two graphs.

- **Scree Plot**: This is the plot of the first eigenvalues $\lambda_1, \ldots, \lambda_i, \ldots$ of the adjacency matrix, versus their rank i in log-log scales. Chakrabarti et al. argue for their effectiveness in [12].

- **Network values of nodes**: The elements $v_{1,i}$ in the first eigen vector $\bar{v_1}$ of the adjacency matrix roughly correspond to the *network values* of nodes in an undirected graph. This metric is used in the R-MAT model, developed by Chakrabarti et al. [12].

- **Quality of k-way partitioning**: We have already described how k-way graph partitioning algorithms can be used to derive graph samples that should preserve useful characteristics. We also can view k-way partitioning as a means of evaluating a sample. In particular, the quality of a possible k-way partition is a graph characteristic we might wish to preserve. We define the quality of a k-way partition to be the number of edges contained in partitions (as opposed to edges between partitions) divided by the total number of edges in the graph. Why is this a useful characteristic? The availability of a good k-way partition can be used in many domains to speed computations on large graphs. In particular, k-way partitioning problems arise naturally in solving sparse systems of linear equations via iterative methods and in organizing the storage of spatially related items on disk in large geographical information systems. Since this is both a computationally and theoretically interesting characteristic of a graph it is useful to consider whether or not it can be preserved in samples. For the input graphs and their samples we used Metis to compute k-way partitions for $k = 2$ through 9. These values are plotted in the results section.

We had hoped to use the following similarity measures but could not do so because of lack of time:

- **Link value distribution**: Traversal set of a link is defined as the set of source-destination node pairs whose shortest path includes the link. The size of a traversal set seems to be a good metric for measuring the importance of the link. An even better metric [38] is *link value*. Link value is defined as the vertex cover of the traversal set of a link, i.e., the minimum number of nodes that need to be removed to eliminate at least one node from each pair in the traversal set. This metric was used in [38] and a distribution of link value gave a very good idea of hierarchy present in real graphs.

- **Effective eccentricity**: Effective eccentricity of a node v is the minimum number of hops required to reach at least 90% of the nodes that are reachable from that node. It has been used in [39] for measuring similarity of graphs.

	Open Source	Language	Interactive	Size Limitations	2D/3D	Generality	Comments
Royere	yes	Java 1.3	Pick, Zoom, Pan	interactive: 10000 nodes	2D	General	Extensive Packages
Walrus	no	Java 1.3, 3D	Zoom, Pan, Distort, Prune	interactive: 500,000 nodes	3D	Digraphs	Requires Spanning Tree
H3Viewer	yes	C++	Zoom, Pan, Distort, Prune	interactive: 300,000 nodes	3D	Digraphs	Requires Spanning Tree
Otter	yes	Java	Zoom, Pan, Filter, Grow	auto-layout: 100 nodes	2D	General	Useful for Large Graphs as Well
NAM/Javis	yes	C/Java	Animated Simulations	hundreds of nodes	2D	Networks	Good for viewing packet traces, etc
GraphViz	yes	C	Label	hundreds of nodes	2D	General	Disorderly Display of Large Graphs
NicheWorks	no	N/A	Zoom, Focus, Filter	millions of nodes	2D	General	Good Interaction Facilities
GRIP	yes	C	Rotate, Zoom	10000 nodes	3D	General	Force Directed Graph Drawing
PAJEK	no	N/A	Partition	N/A	2D/3D	General	Analysis in Additional to Visualization

Table 2: Summary of visualization tools

3.3 Visualization

The visualization systems described above all strike a balance between performance and generality. By making assumptions about the graphs or utilizing specific properties of the graphs to be drawn, a more practical visualization may be obtained faster. For general large non-sparse graphs, none of these systems can perform reasonably in time or space. In conjunction with sampling methods that preserve the essential qualities of large graphs, we will be able to arrive at visualizations of the samples. These visualizations eliminate the need to visualization of the large graph in a naive way while using only modest resources.

The main focus of our visualization component is the demonstration and assessment of our sampling methods. The actual graphical plotting is handled by currently available graphing packages. The sampling tools output graphs in a form that is accepted by these packages. A list of visualization packages closely examined for this is given as Table (2).

To explain some of the terms above, Zoom and Pan describe the ability for the software to navigate the view of the graph by changing the detail levels and scrolling around the graph. Pick is the ability to select nodes to view detail of them. Prune is the ability to remove branches from the view of the graph. Distort is a fish eye view of the graph, providing relative detail around the mouse pointer. Filter is the ability to remove certain types of nodes from the graph. Grow is the ability to create a graph iteratively from the node level. Partition is the ability to cluster nodes into groups. Force directed graph drawing is the algorithm which minimizes the energy of edges in the final representation of the graph.

3.4 Algorithmic Issues

In our milestone report, we discussed the possiblity of using data compression to increase the size of graphs that can be represented in main memory. While not strictly in the spirit of databases research, from a systems perspective such techniques are valuable in greatly increasing the practical range of graphs that can be considered. Due to time constraints and the lack of direct relevance to databases research these ideas were not implement, but as the group spent some effort developing these ideas and believe them to provide good possiblities for future research, we include some discussion of the topic here.

There are two primary techniques for dealing with graphs too large to fit into main memory directly: using external algorithms on graph structures stored on disk, and using data compression to fit the graph into main memory. While it is possible that these techniques could be combined, to our knowledge the literature has not addressed this possibility and we see no straightforward way to merge the two approaches, so we discuss their strengths and weaknesses as distinct methods.

External algorithms take advantage of the fact that disk space is abundantly available, and so in theory there is no upper bound on the size of problems that can be solved (though running times may become very large). Compared to disk, memory is relatively expensive and further there are hard bounds on the amount of memory a given architecture can address, and so the compression approach will in general only apply to smaller graphs. Even though efficient external algorithms can greatly improve over the performance of naive algorithms, they are still significantly slower than internal memory algorithms. Further, the external graph algorithms we are interested in using are complex, and they in turn depend on other external algorithms (for example sorting) and data structures (heaps, tournament trees [29]). We are aware of no standard libraries for these algorithms, and concluded that the complexity of implementing the necessary structures was beyond the reasonable scope of this project. Thus, for this project we adopt algorithms that either have straighforward external implementations, or we work on graphs that fit in main memory.

Most of the literature ([37, 22, 2, 36]) on the subject of graph compression applies techniques specific to the web graph. In particular, many authors are concerned with storing URL information as well as link structure, and they also take advantage of the structure information that the URLs provide: for example, the techniques of [37] treats links between pages on one host (local links) differently from links between pages on different hosts (global links). We believe it is possible to generalize these techniques to apply to arbitrary graphs (or at least arbitrary power law graphs). In particular, we considered a number of techniques for compressing adjacency list representations in main memory.

3.4.1 Node numbering Techniques

Suppose we are considering a web graph on 500,000,000 vertices. Then, in general we will need 29 bits to represent a vertex number, and so using a 32-bit Java `integer` is the correct choice. However, we can do better. The following idea has proved useful in compressing the link structure of web graphs: order the URLs (vertices) lexicographically, and then number vertices according to this ordering. The observation is that many links from a vertex x will be to pages y with a "nearby" number, that is $|x-y|$ is small. We refer to $x-y$ as the delta of the edge. In our case, suppose that a large number of links are to nodes y such that $|x-y| \leq 127$. Then, we can store each such edge as a single Java `byte`.

This optimization is mentioned in a number of papers ([37, 22, 36]) in slightly different forms, but they all only consider numbering nodes based on ordering URLs lexicographically (that is, by host). This is certainly a good heuristic, but there is no reason that this technique cannot be applied to general graphs. However, the difficulty here is to select a good ordering of vertices so that most links are to nodes of nearby numbers.[1]

Finding an optimal such node-ordering is almost certainly NP-complete, but it seems likely that heuristic algorithms could be used so that large numbers of edges have small deltas.

Another approach to node numbering is to try to number nodes so that adjacency lists consist of consecutively numbered nodes; the adjacency list can then represent intervals rather than individual nodes. For example, an adjacency list $\{3,4,5,8,9,10,11\}$ can be represented as $\{3-5, 8-11\}$. This seems particularly applicable in power law graphs, where some nodes have very high degree. For example, suppose we pick two nodes x and y of very high degree. Let S be the set of nodes adjacent to either x or y. We can partition S into (at most) three sets, X, Y, and I, where X contains nodes only adjacent to X, Y contains nodes only adjacent to Y, and I contains nodes adjacent to both. If we number the nodes in X, Y, and I consecutively, then we can represent the adjacency lists of X and Y each with a single numeric range. If x and y are the nodes of highest out-degree in a web graph, they might have degrees on the order of 10^6. Thus, in the naive representation representing these two adjacency lists would have required 8M of memory, which has been replaced by a small constant amount.

This technique can be generalized: If we select k high degree nodes, then the set of nodes

[1] This has a very similar flavor to finding good space-filling curves such as z-order.

adjacent to one or more of these nodes can be partitioned into $2^k - 1$ sets. We can construct these sets, and assign each set a "score" equal to the number of the k nodes that are all adjacent to this set of nodes times the size of the set. We then number these nodes sequentially, thus compressing their representation for all nodes that link to this set. We believe that picking a node numbering so as to maximize the size of these consecutive ranges is a new contribution to the theory of graph compression that seems particularly applicable to power law graphs; we hope to demonstrate the usefulness of this technique experimentally.

Note that these two different node-number techniques can potentially be used together: some nodes are numbered so that high-degree nodes are adjacent to large consecutive sequences of nodes. Then, nodes that have not been numbered yet could be numbered so as to maximize the number of short-label edges: this could be done via some kind of randomized traversal, for example. It remains to determine experimentally which of these methods is most effective on actual web graphs, and to determine how the parameters should be adjusted.

3.4.2 Standard Compression Techniques

In [37] the authors applies several standard data compression techniques. They consider the set of k nodes with highest in-degree and use Huffman-Coding to assign aliases to these nodes; then, these short aliases are used to represent links to these nodes. A similar technique is applied to represent local frequent links, that is, frequent links on a specific host. On a general graph we do not have the notion of a host, but it may be possible to identify (or allow the user to identify) sets of highly interconnected nodes that are good candidates for internal compression. Further, [37] uses an encoding based on a Golomb code to compress global adjacency lists of length greater than four.

3.4.3 Bit Matrices for Dense Subgraphs

Suppose we identify a subgraph of the webgraph that is dense. We can then represent the edges internal to this subgraph via an adjacency matrix, with significant savings over an edge-list or adjacency-list representation. This technique could also be applied to compress individual adjacency lists as bit vectors in the case where the adjacency list was dense. This is of course dependent on node-ordering. We could potentially get around this by creating alias name sets for dense subgraphs that could then provide indices into such bit vectors where appropriate.

	mp	mnc	ff	ps	svd	rns
small	(1264, 2208)	(1265, 78838)	(989, 988)	(5876, 5860)	(520, 6164)	(1237, 97)
medium	(6324, 54585)	(6234, 137646)	(8863, 8862)	(20079, 27149)	(940, 8931)	(6266, 3166)
large	(25295, 71310)	n.a.	(24155, 24154)	(45101, 87861)	(1755, 12906)	(25388, 44338)

Table 3: Size of Epinions Graph Samples (nodes, edges). Original Graph is on 75,888 nodes and 508,960 edges.

4 Experiments and Results

In this section we describe the performance of our various sampling algorithms, in how similar they are to the original graph. Specifically, we seek to answer the following:

- Most importantly, how good are our sampling algorithms?
- Do they preserve the *essential* characteristics of the original graph and how they do they stack against each other, regarding this property? Moreover, is it always the case that with regard to any property we care about (found in real-world graphs), one algorithm is always better than the other? How does this change with the sample size?
- How computationally expensive is each of these sampling algorithms?

In the rest of the section, we first present details about our test-bed, and the experiments we conducted. Next, we analyze the performance of each sampling algorithm against similarity measures proposed or observed elsewhere (The canonical example is the power-law distribution on the degree distribution). After that, we compared the performance of various graphs against the one new k-way partition quality statistic. Next, we discuss the results in general and try to answer the questions raised at the beginning of this section. We also summarize the important aspects of each sampling algorithm in Table (4). The final two subsections talk about visualization experiments and results.

4.1 Experiment and infrastructure details

We evaluated the algorithms primarily on 3 datasets. *Epinions* is a trust-graph from epinions.com; it has 75,879 nodes corresponding to users, and 508,960 directed edges corresponding to trust relationships. The *Clickstream* dataset is a bipartite graph representing web browsing behavior. One set of nodes (size 23,396) corresponds to users and the other set (size 199,308) corresponds to web pages. The graph has 952,580 edges. Finally, the *Oregon* dataset is an undirected graph of autonomous system connectivity. Analysing all these datasets produced a huge amount of raw data, and so in this discussion we will focus our attention on the performance of the sampling algorithms on the representative Epinions graph.

Since we wanted to study the impact of sample size on the accuracy of the algorithms, we ran each of our 6 sampling algorithms on the original datasets to create samples of 3 sizes. Our target sizes were a*small* at 1/60 the original number of nodes, a *medium* sample of 1/12 of the nodes, and a *large* sample with 1/3 the number of nodes of the original. Table (3) shows the sizes of the samples actually created by the various methods. Controlling the exact size of the sample generated is difficult for both the path sampling and SVD methods, and hence the sizes of samples generated by these methods are significantly different from the target values. In particular, when looking at the figures one should note that SVD is working with much smaller samples than the other methods, which makes its performance seem even more impressive.

For the mnc method, we were unable to generate a large sample because Metis ran out of memory on a machine with 1 gigabyte of RAM, the largest we had available.

We used RMAT extensively to analyze our data. In addition, we used Metis ([33]) to generate roughly equal sized partitions of the graph. Most of our sampling algorithms were written in Java, and we also provide an efficient way to store a large graph by using only two array objects. For the SVD sampling method, we used Matlab. In addition, different scripting languages like perl and bash were used to convert to and fro from different file formats.

4.2 Performance of Sampling Algorithms

Figures 4 to 21 show how different sized samples produced by the six sampling algorithms we compare, measure to the original graph with regard to the similarity measures enumerated in Section 3. Based on these, we draw the following conclusions.

4.2.1 Random Node Sampling

This method is more of a baseline for us to test our results. As shown in Section 3, this method is expected to preserve the degree distribution of real-world graphs. But, because of the high variability of the results, when the sample size is small, it ends up with a high number of disconnected nodes (as evident from the Hop plot of Figure 14 and of Figure 15. The number of nodes in these cases is 1237 and 6266, where as the maximum number of reachable pairs is only about 2000 (about 3%) and 2 million (about 10%)). Since the method works quite well in case of the large sample (it models all characteristics closely - Figure 16), it seems that this method works well only if the sample size is at least about a third of the original size. The obvious advantages of this method are that it is very easy to understand and implement. Moreover, it also runs very fast (expected running time: O(n' log n), where n' is the number of nodes sampled, and n is the number of nodes in the original graph, since the average degree in the real-world graphs is about log (n)).

4.2.2 Forest Fire

This method works quite well. In particular, it does a good job of modeling the out-degree distribution (the in-degree is always 1, an artifact of how we implemented this scheme), the hop plot (though it does not have a diameter as small as the original) and the eigen-value distribution. We believe that this class of algorithms present an effective way of sampling real graphs, and there is plenty of room for coming up with a set of parameters that perform significantly better. This algorithm, as we have implemented it, should have a similar running time as *random node sampling.*

4.2.3 Path Sampling

This method tends to generate graphs with disconnected components, but the largest connected component (which was used to generate the results) preserves *all* the graph properties. Based on this idea, we believe that a random walk on the graph might turn out to be a good sampling algorithm. Currently, we implement this in a randomized way and post-process the sample to eliminate duplicates. We haven't worked out how much resources it exactly takes to run this algorithm, but in practice, it has been fast. Its running complexity should be in between O(n' log n) and O(n^2).

4.2.4 Singular Value Decomposition

This is one method that has sound theoretical principles behind it. The samples produced were very small, and the results were by far the best in all tests we conducted (the *medium* sample is about 1/75th the size of the original). This method also has the advantage that it always gives the exact same results - there is no variability. Computing it though requires a lot of CPU cycles - it has a running complexity of at least 0(n^3). Moreover, it uses a lot of memory as it continually operates on the whole matrix (Blocking helps though and we provide an implementation of block matrix multiplication in Matlab).

4.2.5 Metis partition

Using a Metis-generated partition produces good samples for all sample sizes. The method is expensive to compute computationally, but can give samples of a guaranteed size. The main open question for this method is that the usefulness of the sample selected depends to a large extent on the partition on which it is based. Thus, this sampling method can offer samples with many different degree densities, but we can offer no formal method for choosing among these other than trying them and determining which work best for the particular problem in question.

4.2.6 Metis Node Collapsing

This technique captures the overall topological structure of the graph, and should be useful in applications such as network modeling and shortest-path type computations. This preservation of structure is not well described the graph similarity measures we used. Examining the visualization graphs for this technique, however, shows that an overall radial structure of the graph is preserved; this should be compared to the more spread out result created by the mp algorithm. The version implemented produces somewhat dense graphs, but as mentioned previously a thresholding technique could be applied to remedy this problem. For small samples (Figure (20)) this causes the degree distribution to be somewhat noisy and to have more high-degree nodes than is ideal. However, it does a good job of modeling the in and out degree distributions for larger sample sizes (Figure (21)). This method also does a good job of maintaining the relative difficulty of performing k-partitioning; we discuss the measure in more detail in the next section. Unfortunately, this method is computationally expensive.

4.3 Evaluation of the k-way partitioning quality measure

As discussed previously, the quality of k-way partitioning achievable for a given graph may be an important measure that should be preserved through sampling. Figures (3), (6), and (6) show how well each of our partitioning methods does against this measure.

Since the forest fire model always produces a tree, these samples always have very high sample quality and hence do not match the input graph for this similarity measure. Thus, if preserving partition quality is a goal of sampling the forest fire method is a poor choice.

For the small sample, we see the svd method does very well, closely matching the quality of the the original graph over the range of k-values. The mnc quality values preserve the trend of the original set, but show a much lower overall quality. This is expected, as the smaller the sample, the denser a graph mnc will produce, and denser graphs will generally have lower-quality partitions (more edges will need to be cut). On the other hand, random node sampling, forest fire, and path sampling all produce graphs that are too sparse and thus result in easy (high quality) partitionings.

For the medium sample size, mnc is still produces a dense graph that tracks the overall trend of the original, but at a lower quality. The single partition Metis sample (mp) also has lower overall quality. This, however, probably is a consequence of selecting a partition with a large number of edges. We looked at a number of different partition choices, and depending on the edge density of the sample chosen the mp sample could have overall lower or higher quality than the original graph. The SVD sample method again does well, and the path sampling method is starting to do better. Random node sampling also does better, but clearly there is high variance in it's tracking of the quality.

Almost all the methods do better on the large sample. Random node sampling for the first time appears to be a reasonable sampling method, performing comparably to path sampling, svd, and mp.

Overall, the SVD method seems to do very well with respect to this metric, especially considering that it's samples are relatively smaller than those generated for the other methods. The mnc method also did well, but suffered because it produced denser graphs than the original. This problem could be remedied by using the edge-weight thresholding technique described in the proposed methods section.

4.4 Discussions

Our results show that the sampling algorithms we have proposed and implemented are quite effective in preserving the essential properties of real-world graphs. In particular, if the sampling needs to be done once, and ample computational and memory resources are available (at the time of sampling), then the *svd* techniques work well. If instead, samples are to be generated cheaply, *path sampling* is more suitable. Moreover, the *forest fire* class of algorithms need to be studied in greater detail and they also provide an effective sampling method. If the size of the sample is greater than about a third of the original graph, then *random node sample* works as good as anything else.

One sampling algorithm may be better in preserving one property, while some other might be better with regard to another property. For example, the small sample generated by *forest fire* is not able to model the distribution of network values (Figure 5) even though it models the degree distribution effective. The opposite is true of *Metis node collapsing* on a small sample (Figure 20). Similarly, for low sampling sizes, *forest fire* turns out to be better than *random node sampling*, but for large sample sizes, the converse is true.

Obviously, a lot of testing and formal reasoning needs to be done, before our results can be generalized. But, our results do demonstrate the feasibility of the process, and efficacy of our proposed solutions.

4.5 Visualization Setup

Our purposes do not require the display of graphs with millions of nodes, but our samples will be permitted 10,000 or more nodes. This requirement disallows some of the software packages described above, especially most of the 2D tools, as they become either too disordered or too slow in rendering and arranging a large number of vertices. These packages are NAM/Javis and

Method	Pros	Cons
rns	Simple; works for large sample sizes	disconnected graphs when sample size is small; high variability
ff	new class of algorithms; simple; effective	unable to model network values
ps	simple; models all parameters	not deterministic
svd	deterministic; accurate	high resource usage
mp	preserves local structure; good overall method	high resource usage difficult to select good partition
mnc	preserves global structure	high resource usage; only produces small samples

Table 4: Summary of different sampling algorithms

GraphViz. Walrus and H3Viewer require a meaningful spanning tree as input and accept only directed graphs, but they are easily the most beautiful displays, and effort was put into integrating Walrus, the better of the two, with the sampling methods. Nicheworks was not used merely because it will be difficult and costly to obtain and does not appear to have facilities some of the open source applications do have. Otter is quite old, and appears to have an inadequate view of most large graphs, unless they are specifically tailored for the application. GRIP was prone to bugs and crashing with large inputs, so was also not used. Pajek is not open source, but is freely available and was able to handle moderately large graphs. Royere, while modern and connected to an entire graph visualization library, was unable to perform adequately in laying out large graphs automatically.

Out of the tools Pajek and Walrus were chosen as the output packages for the generated samples. The five different sampling algorithms' outputs were attempted to be displayed with the visualization packages. The goals were to determine important characteristics for general large graph visualization software and also to provide insight into the actual graph data set used. Lastly, differences between the sampling algorithms are approached here statistically, but with the visualization may also be explored visually.

4.6 Visualization Details and Observations

Walrus was not used for visualization in the end, due to its major constraints. Walrus exploits hierarchical structure in directed graphs to quickly use Tamara Munzner's H3 algorithm to guarantee fast frame rate updates in graph navigation. The most restrictive requirement is a meaningful spanning tree as input in addition to an input graph. This directed spanning tree is created from some domain knowledge of the graph and must reach all nodes in the input graph from a single root. This requirement can be difficult on a general graph without domain knowledge and can even be impossible, as many directed graphs contain no such spanning tree. These difficulties proved to outweigh the beautiful pictures Walrus was capable of producing.

Pajek, given a graph on the order of 10000 nodes, was able to produce some meaningful visualizations, and thus was useful in the assessment of the data set and the sampling methods. The advantage Pajek had over other similar visualization tools was its layout algorithms. For all output we used the Fruchterman-Reingold[18] layout algorithm. This is a force directed algorithm, similar to GRIP in its goal. Nodes are positioned with some repulsion between nodes proportional to the distance between the nodes, but also some attraction between nodes connected by an edge. Through simulated annealing the graph converges to some final graph layout. Pajek reaches convergence for graphs of around 10000 nodes in about an hour and graphs of 5000 nodes in about 20-30 minutes on a fast desktop. This performance is adequate for graphs this size, but much larger graphs are not handled as well by Pajek.

Comparing Pajek's performance with the other visualization tools, it is clear that a smart automatic layout algorithm is required for any visual understanding general large graphs. Most of the tools could physically render large graphs within a few seconds, but the meaning of a graph cannot be conveyed without a useful layout. The spanning tree constraint of H3 above approaches this problem by restricting the graph, but for general graphs force directed layouts are quite effective for 2D and 3D layouts, both which Pajek supports.

2D visualizations were generated with Pajek for 5 different sampling techniques of the Epinions data set. For each sampling technique, multiple sample sizes were displayed with Pajek. The results are included below. In examining these images, the layout algorithm must be taken into account. Regions of densely packed nodes are strongly connected, because the attraction caused by the edges must outweigh the repulsion of the nodes being placed close together. Thus one thing to look for are these dense regions of nodes to tell us something about the data set. Conversely, nodes separated by sparse area are not strongly connected as the repulsion force has far outweighed the attraction. Around the periphery of some graphs are circles of nodes which are forced away from most edges. These edges are connected very weakly, if at all, to the main core of nodes and so are forced as far away as possible, constrained only by an oval which dictates the border of the possible node placements. Though the full unsampled graph was intractable to visualize with Pajek, observations can be made about the original graph and about the sampling methods from the sample visualizations.

Walrus was not used for visualization in the end, due to its major constraints. Walrus exploits hierarchical structure in directed graphs to quickly use Tamara Munzner's H3 algorithm to guarantee fast frame rate updates in graph navigation. The most restrictive requirement is a meaningful spanning tree as input in addition to an input graph. This directed spanning tree is created from some domain knowledge of the graph and must reach all nodes in the input graph from a single root. This requirement can be difficult on a general graph without domain knowledge and can even be impossible, as many directed graphs contain no such spanning tree. These difficulties proved to outweigh the beautiful pictures Walrus was capable of producing.

Pajek, given a graph on the order of 10000 nodes, was able to produce some meaningful visualizations, and thus was useful in the assessment of the data set and the sampling methods. The advantage Pajek had over other similar visualization tools was its layout algorithms. For all output we used the Fruchterman-Reingold[18] layout algorithm. This is a force directed algorithm, similar to GRIP in its goal. Nodes are positioned with some repulsion between nodes proportional to the distance between the nodes, but also some attraction between nodes connected by an edge. Through simulated annealing the graph converges to some final graph layout. Pajek reaches convergence for graphs of around 10000 nodes in about an hour and graphs of 5000 nodes in about 20-30 minutes on a fast desktop. This performance is adequate for graphs this size, but much larger graphs are not handled as well by Pajek.

Comparing Pajek's performance with the other visualization tools, it is clear that a smart automatic layout algorithm is required for any visual understanding general large graphs. Most of the tools could physically render large graphs within a few seconds, but the meaning of a graph cannot be conveyed without a useful layout. The spanning tree constraint of H3 above approaches this problem by restricting the graph, but for general graphs force directed layouts are quite effective for 2D and 3D layouts, both which Pajek supports.

2D visualizations were generated with Pajek for 5 different sampling techniques of the Epinions data set. For each sampling technique, multiple sample sizes were displayed with Pajek. The results are included below. In examining these images, the layout algorithm must be taken into account. Regions of densely packed nodes are strongly connected, because the attraction caused by the edges must outweigh the repulsion of the nodes being placed close together. Thus one thing to look for are these dense regions of nodes to tell us something about the data set. Conversely, nodes separated by sparse area are not strongly connected as the repulsion force has far outweighed the attraction. Around the periphery of some graphs are circles of nodes which are forced away from most edges. These edges are connected very weakly, if at all, to the main core of nodes and so are forced as far away as possible, constrained only by an oval which dictates the border of the possible node placements.

The mp sampling appears to model the original graph decently. The large graph has a small well connected center, which is a fixture of power law graphs. It is rightly located in the center of the image, as those nodes are well connected to many nodes all around it. In the medium and small graphs, the well connected locus remains, and is not quite in the center due to the fewer node connections. It is interesting that we can easily see other strongly connected locations in the large sample. This may reflect different groups within epinions where there are trust relationships in cliques of people due to varying common interests. The one problem with the sampling method may be the large number of isolated nodes around the periphery. Most likely the sampling isolated more nodes than were isolated to begin with. However the general essense of the original graph appears to have been retained.

The mnc sampling seems to have problems in that the sample is extremely dense around the center. Unrealistically, almost every node in the graph is extremely well connected. The proportion of nodes to edges gets very small as the sample becomes smaller, implying that only the highly connected nodes are being kept.

The ff sampling algorithm has almost the opposite problem, as it seems to sample nodes so that there is no longer any well connected region that stands out. This behavior carries over to all three graphs. This sampling algorithm may need to be more conscious of preserving well connected clique like structures.

ps Sampling looks better than most of the other sampling algorithms. It retains the well connected center, even when the sample gets much smaller. It should be noted that these graphs are larger than the others presented, due to the differences between the sampling methods. Again, many nodes are left stranded here.

The sampling for svd does not resemble any of the others. It exposes some structure in the graph in a clear way that is consistant through all of the generated samples. Comparing the medium and large graphs, the two graphs share similar structure characteristics like the well defined projection out from the well connected nodes. Also, it is easy to see the well connected nodes and where their connections lead. Coupling this interesting structure with svd's success on the statistical side provides evidence that this samples preserve some important characteristics of the original graph.

rns looks fine at the large level, but with respect to the other images, it is clear that these samples have no intelligence, as many nodes are chosen which are connected to no edges. These orphans become the majority as the sample shrinks, which is a clear indication that the proper structure is not being preserved.

From observations of the visualizations alone, svd, ps, and mp appear to retain qualities of the original graph well.

5 Conclusions

We introduced a range of new graph sampling algorithms, and began an investigation into the graph properties that these sampling algorithms preserve. There can be no perfect graph sampling algorithm, as by definition information is being thrown away when a graph is sampled. In practice, then, when a large graph is being analyzed the best approach will probably be to try a number of different sampling techniques, and choose the one that retains the most useful information and discards information not of interest for the particular problem being considered. While we can offer no general solution to finding such a sampling algorithm, we believe that by analysing the performance of the sampling algorithms with respect to a number of different measures of graph similarity we can offer some direction in selecting an appropriate sampling method for a given problem.

6 Figures

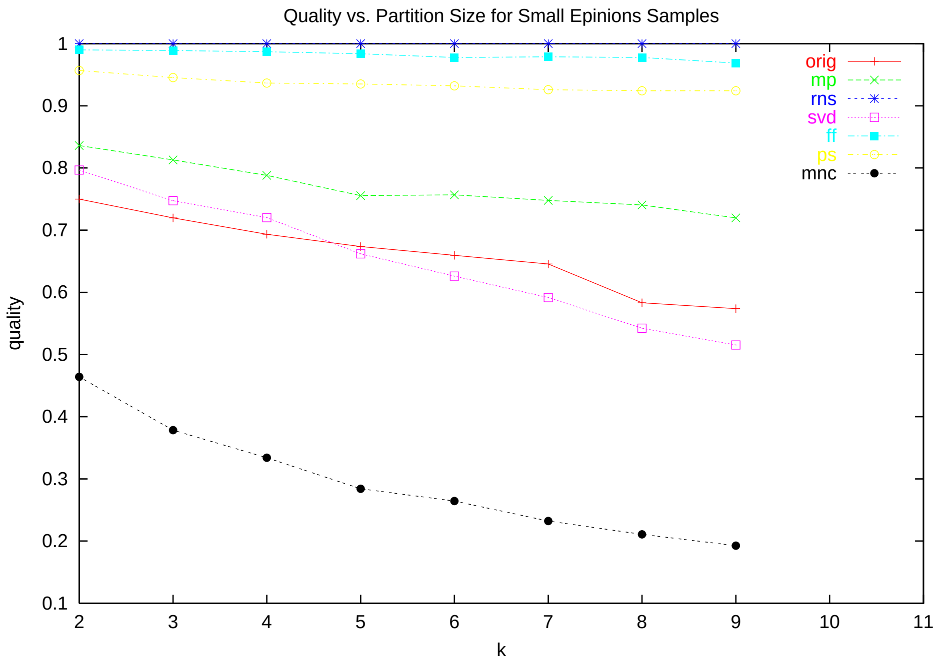


Figure 1: Epinions Small

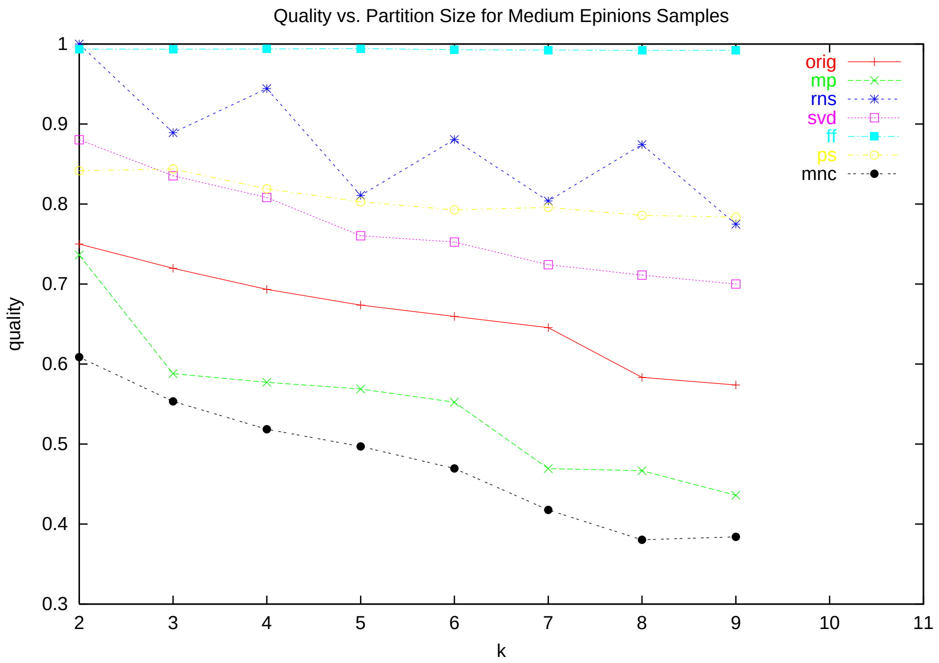


Figure 2: Epinions Medium

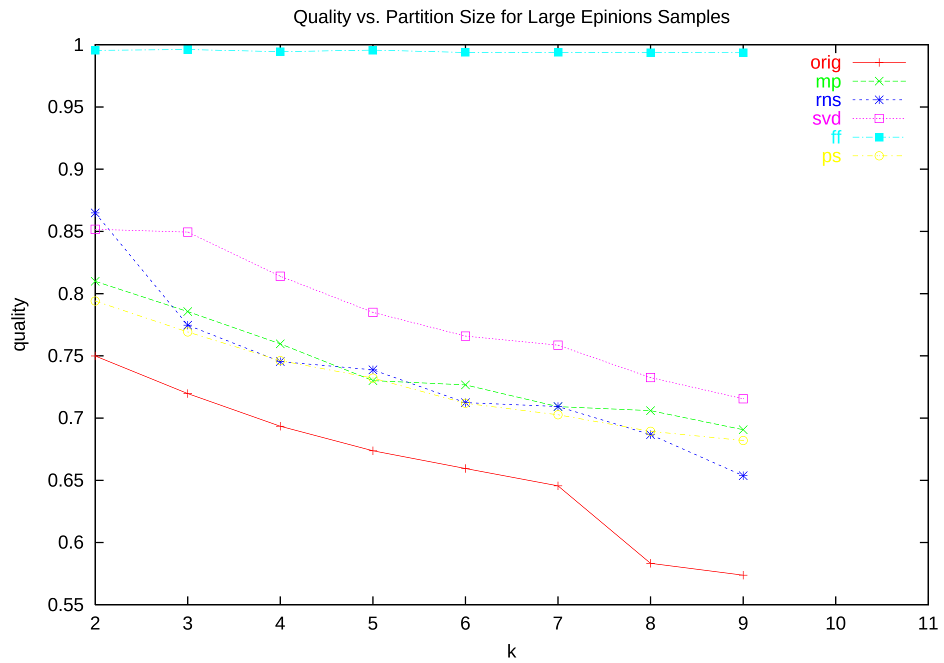


Figure 3: Epinions Large

References

[1] J. Abello, A. L. Buchsbaum, and J. R. Westbrook. A functional approach to external graph algorithms. *Algorithmica*, 32:437–458.

[2] Micah Adler and Michael Mitzenmacher. Towards compressing web graphs. In *Data Compression Conference*, pages 203–212, 2001.

[3] R. Albert and A.-L. Barabasi. Topology of evolving networks: local events and universality. *Physical Review*, 2000.

[4] L. Arge. The buffer tree: A new technique for optimal i/o algorithms. In *Proc. 4th WADS*, LNCS, pages 334–45. Springer-Verlag.

[5] Lars Arge, Michael A. Bender, Erik D. Demaine, Bryan Holland-Minkley, and J. Ian Munro. Cache-oblivious priority queue and graph algorithm applications. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 268–276. ACM Press, 2002.

[6] Lars Arge and et al. On external memory mst, sssp and multi-way planar graph separation (extended abstract).

[7] Lars Arge, Ulrich Meyer, Laura Toma, and Norbert Zeh. On external-memory planar depth first search. *Lecture Notes in Computer Science*, 2125:471–??, 2000.

[8] V. Batagelj and A. Mrvar. Pajek – program for large network analysis, 1998.

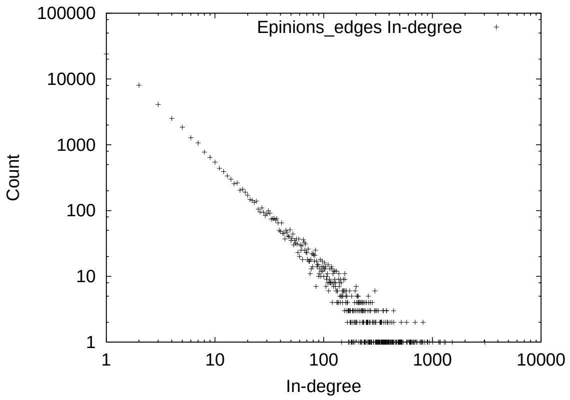
[9] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[10] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, and A. Tomkins. Graph structure in the web: Experiments and models. In *9th World Wide Web Conference*.

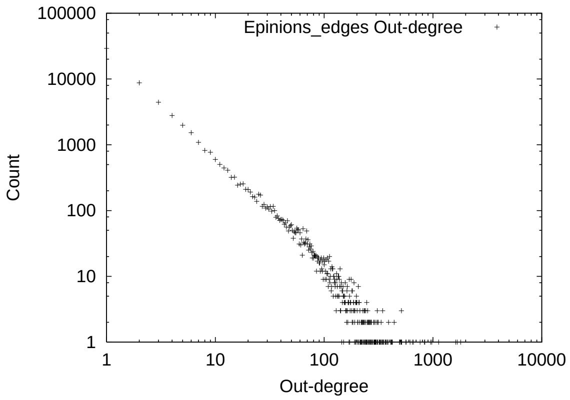
[11] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph Structure in the web. In 9^{th} *International World Wide Web conference*, 2000.

[12] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-MAT: A recursive model for graph mining. In *submitted to ICML*, 2003.

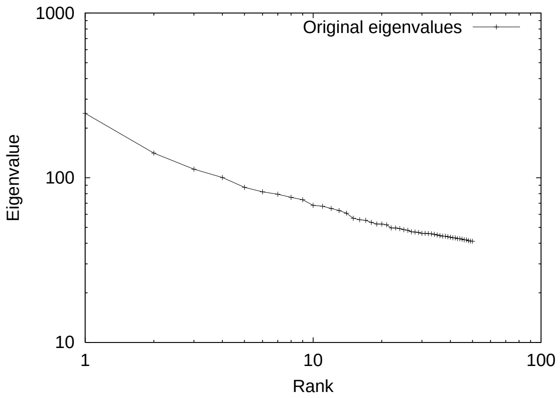
[13] Yi-Jen Chiang, Michael T. Goodrich, Edward F. Grove, Roberto Tamassia, Darren Erik Vengroff, and Jeffrey Scott Vitter. External-memory graph algorithms. In *Symposium on Discrete Algorithms*, pages 139–149, 1995.



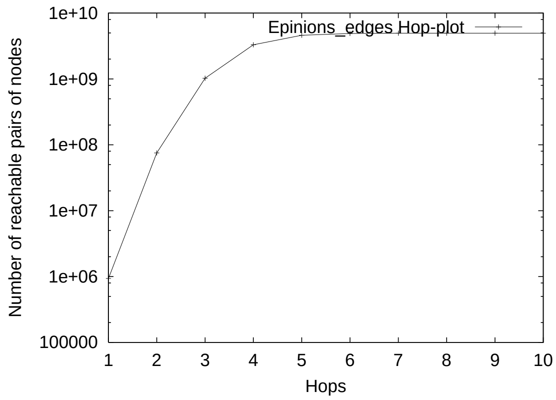
Count-vs-In-degree



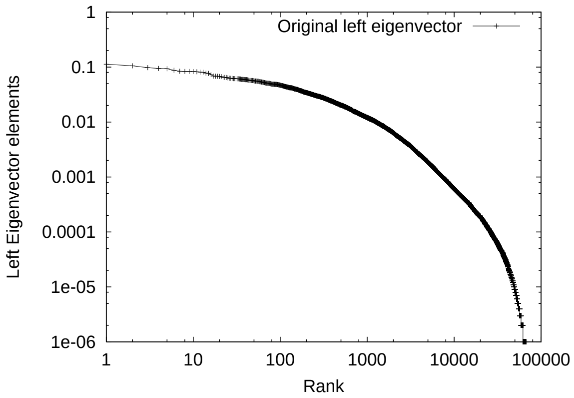
Count-vs-Out-degree



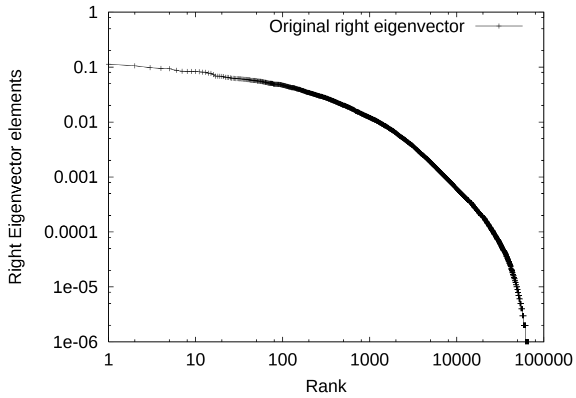
Eigenvalue-vs-Rank



Hop plot

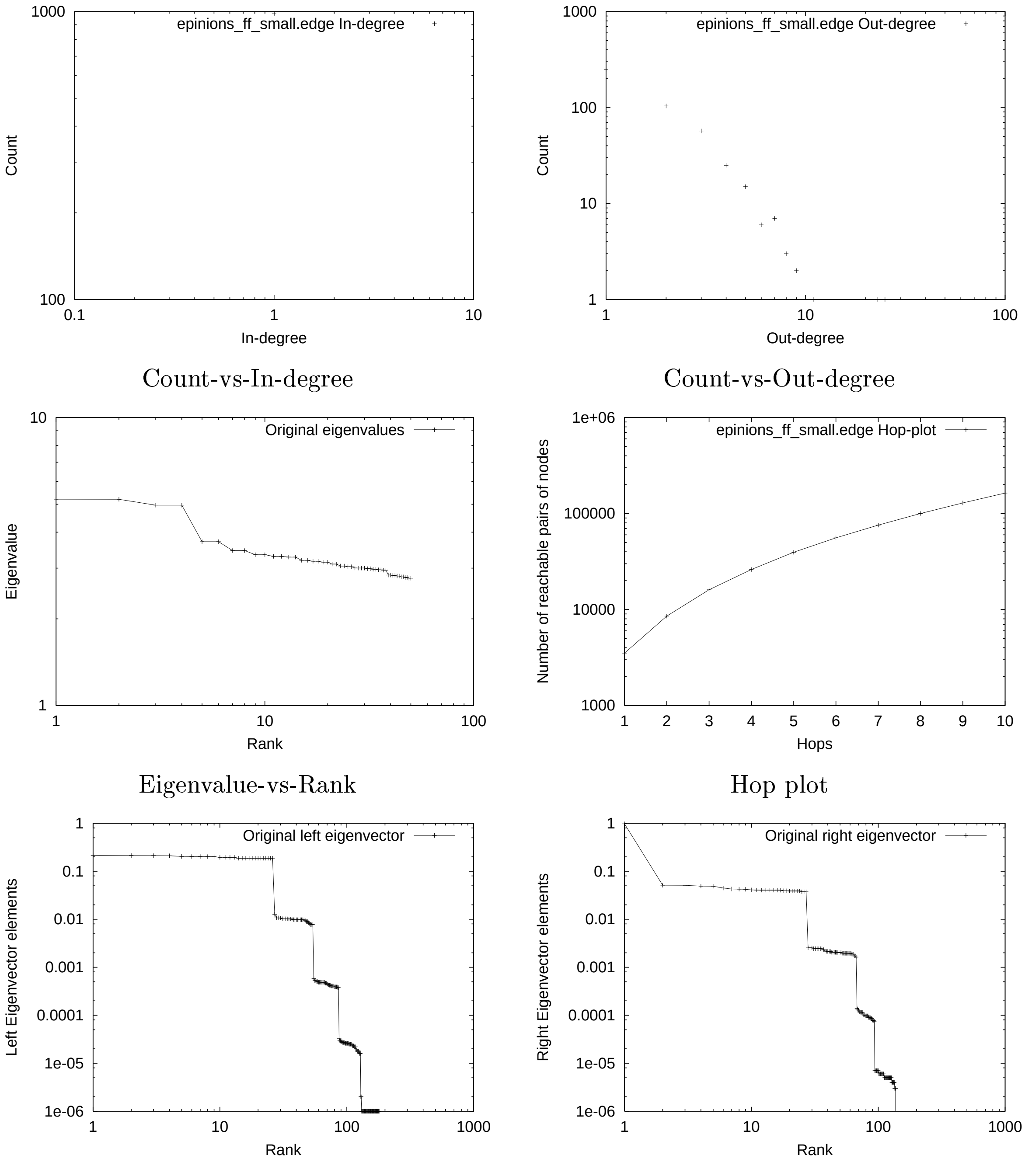


Left Eigen Vector



Right Eigen Vector

Figure 4: Epinions- original graph



Count-vs-In-degree

Count-vs-Out-degree

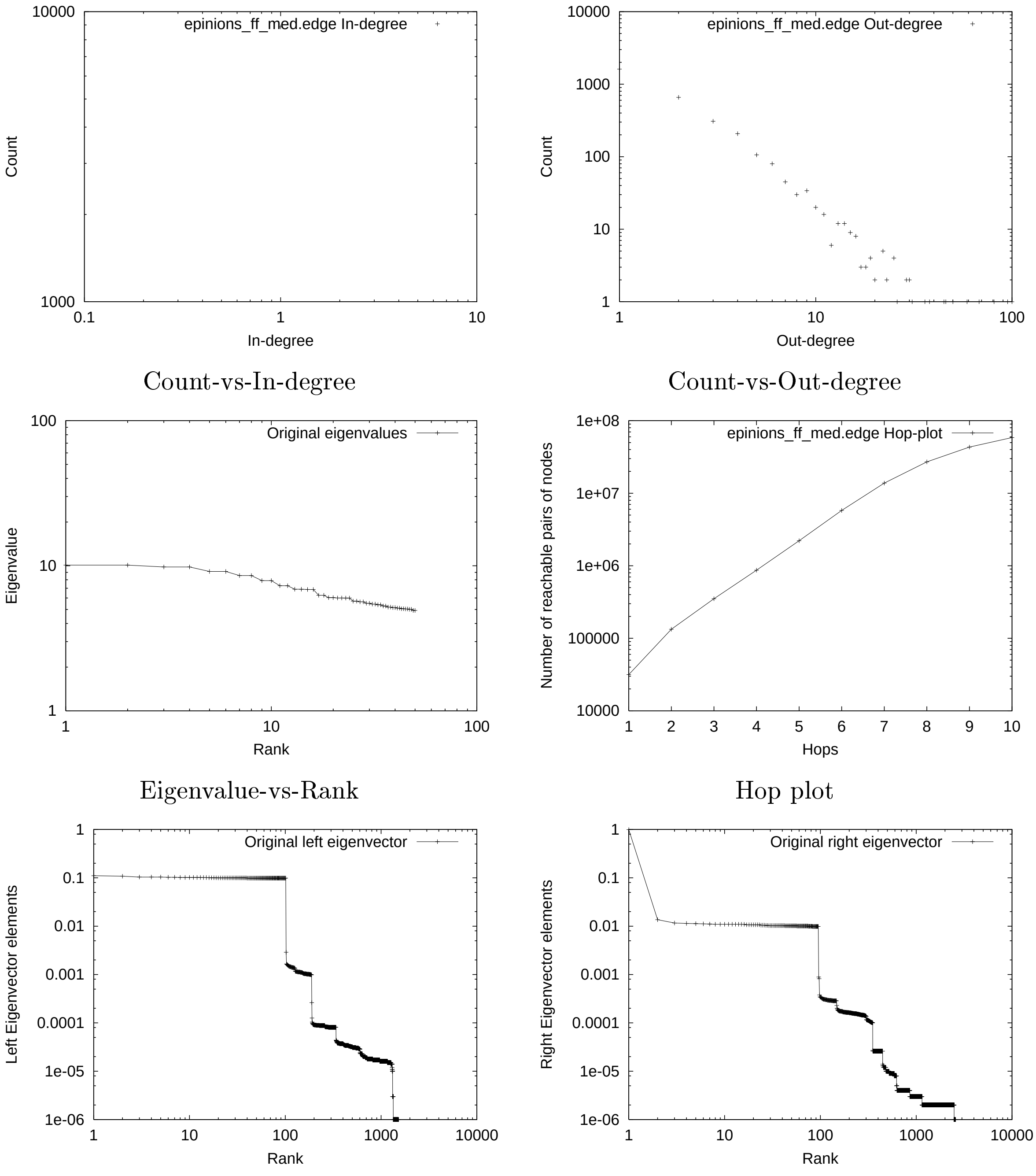
Eigenvalue-vs-Rank

Hop plot

Left Eigen Vector

Right Eigen Vector

Figure 5: Epinions: sampling by ff - small size



Count-vs-In-degree

Count-vs-Out-degree

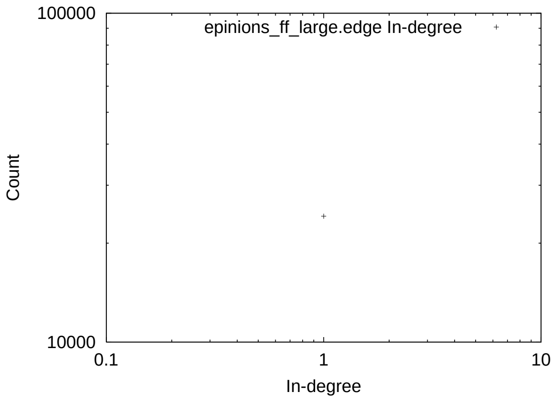
Eigenvalue-vs-Rank

Hop plot

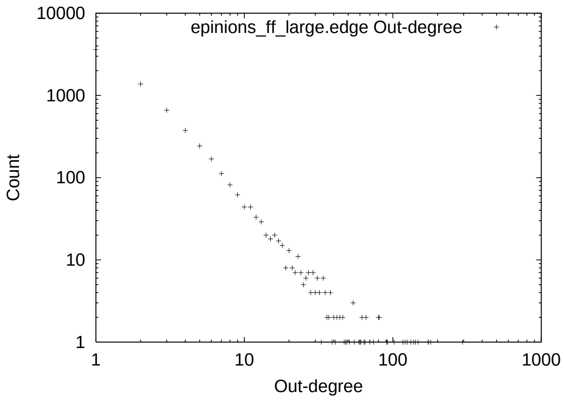
Left Eigen Vector

Right Eigen Vector

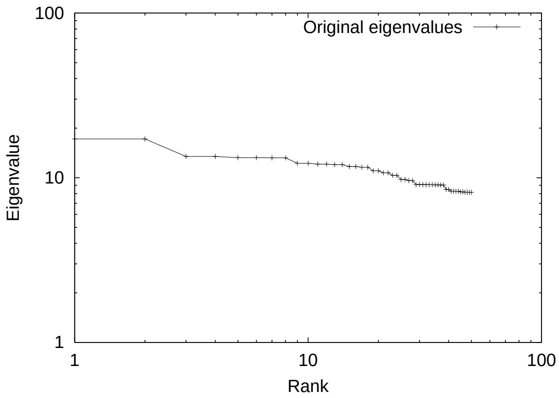
Figure 6: Epinions: sampling by ff - med size



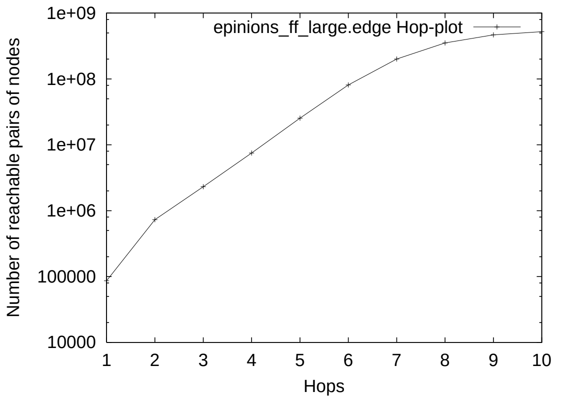
Count-vs-In-degree



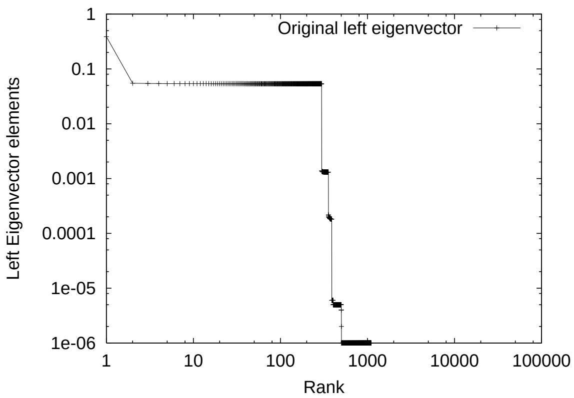
Count-vs-Out-degree



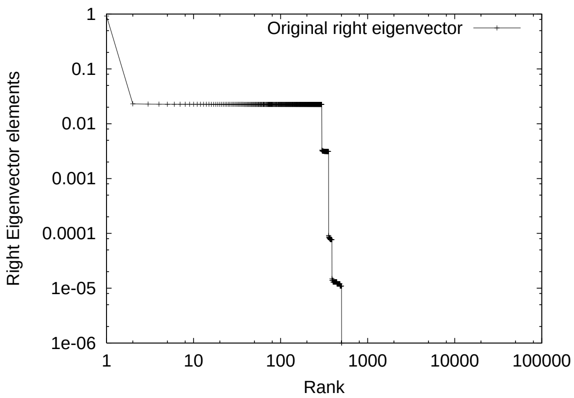
Eigenvalue-vs-Rank



Hop plot

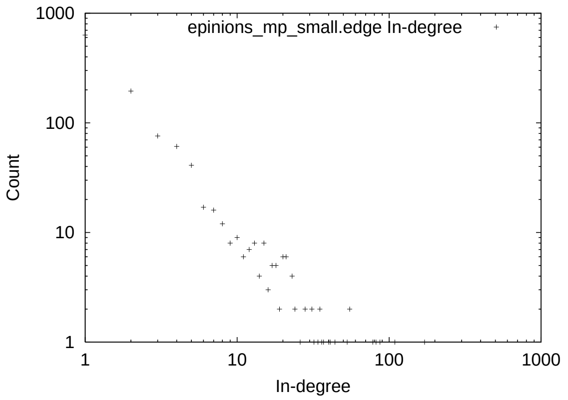


Left Eigen Vector

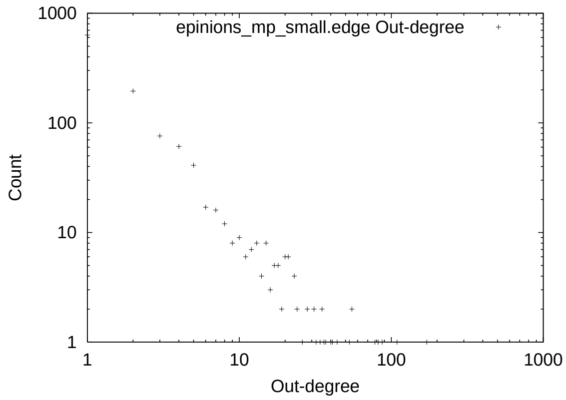


Right Eigen Vector

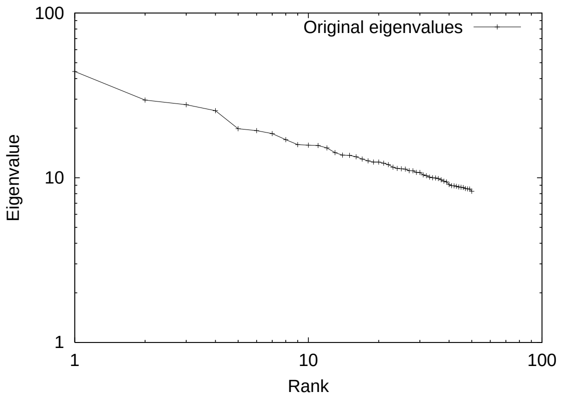
Figure 7: Epinions: sampling by ff - large size



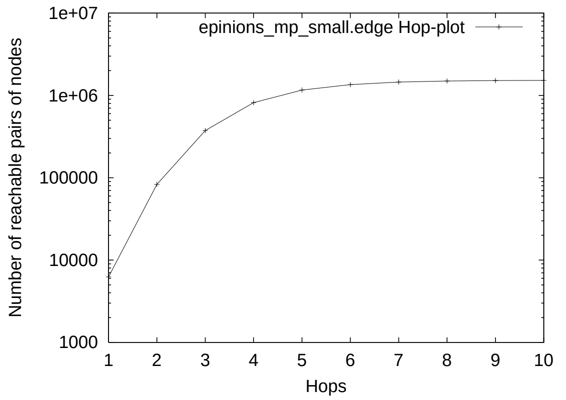
Count-vs-In-degree



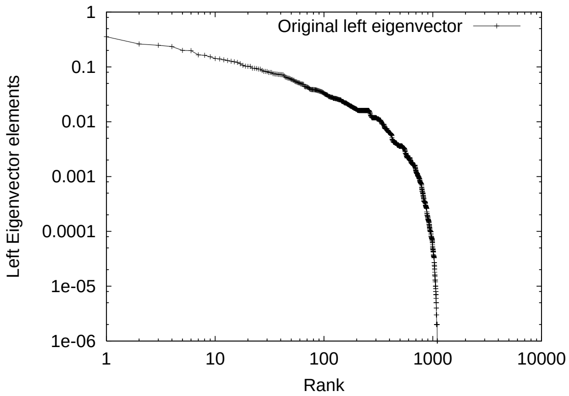
Count-vs-Out-degree



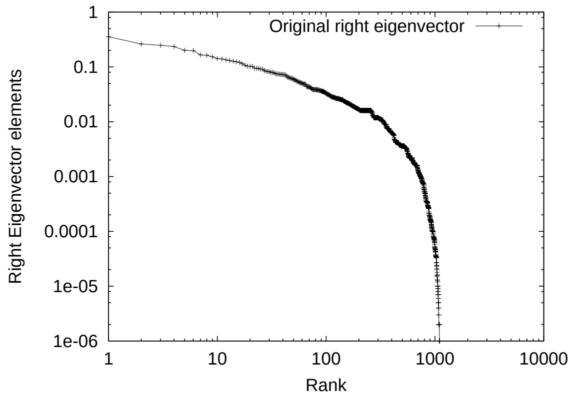
Eigenvalue-vs-Rank



Hop plot

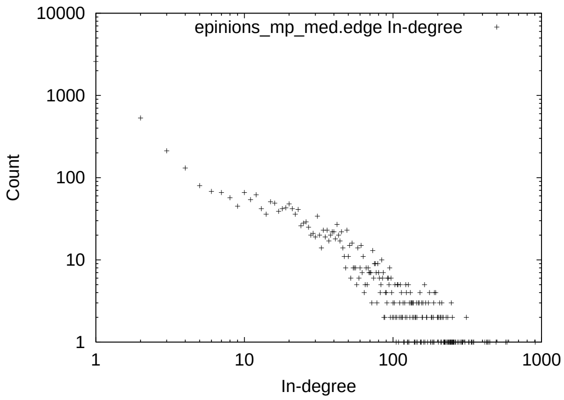


Left Eigen Vector

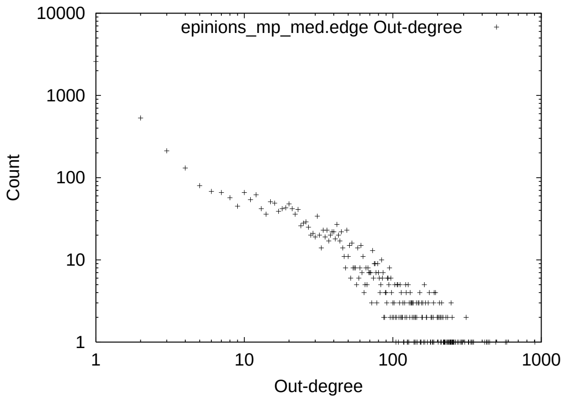


Right Eigen Vector

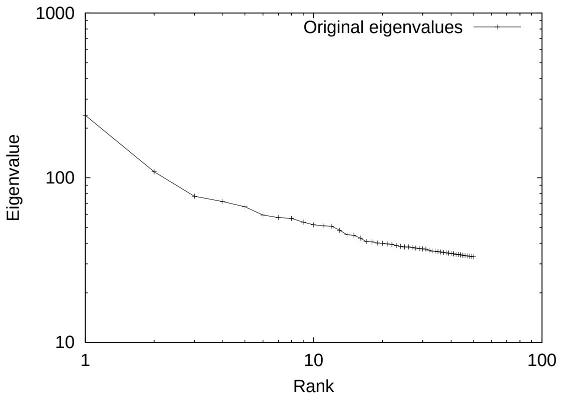
Figure 8: Epinions: sampling by mp - small size



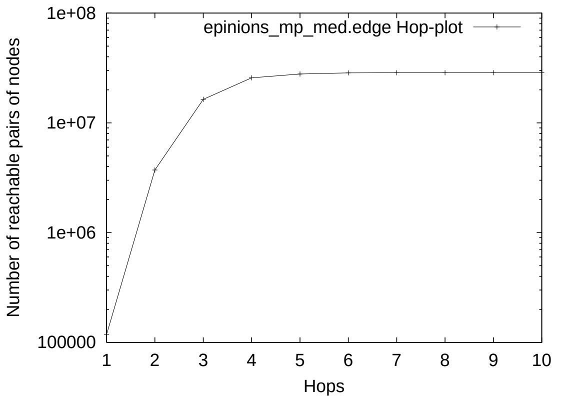
Count-vs-In-degree



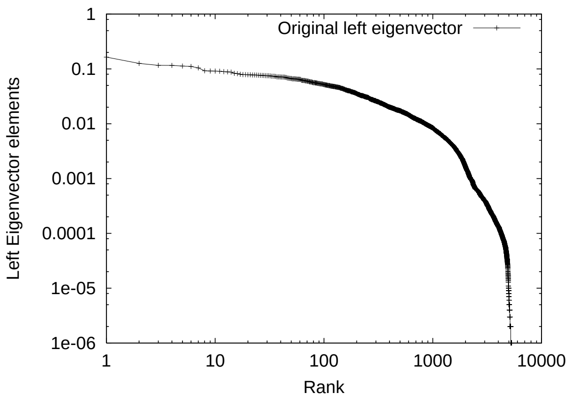
Count-vs-Out-degree



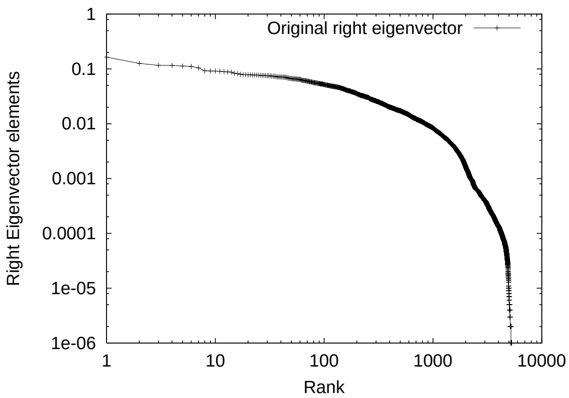
Eigenvalue-vs-Rank



Hop plot

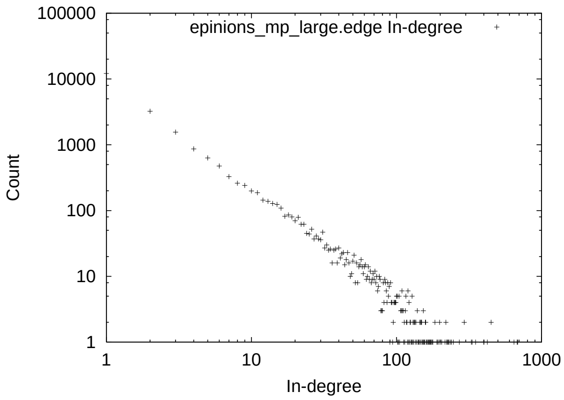


Left Eigen Vector

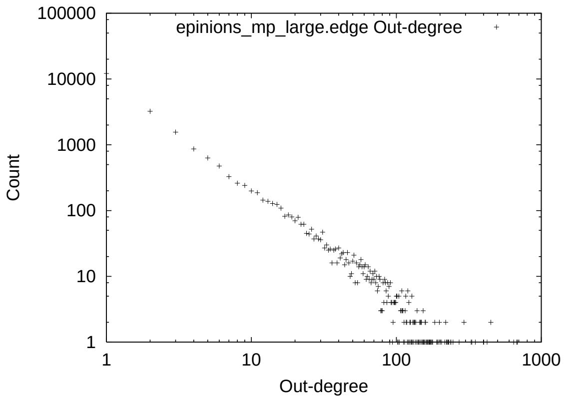


Right Eigen Vector

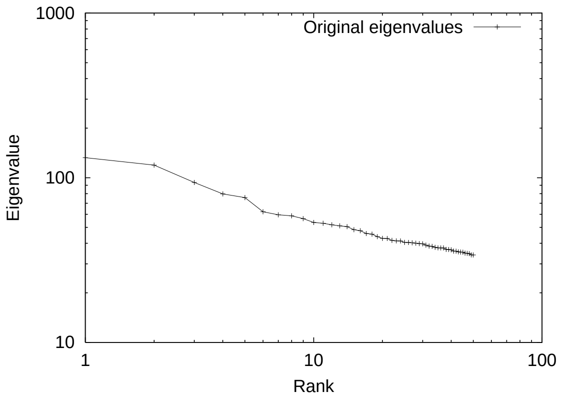
Figure 9: Epinions: sampling by mp - med size



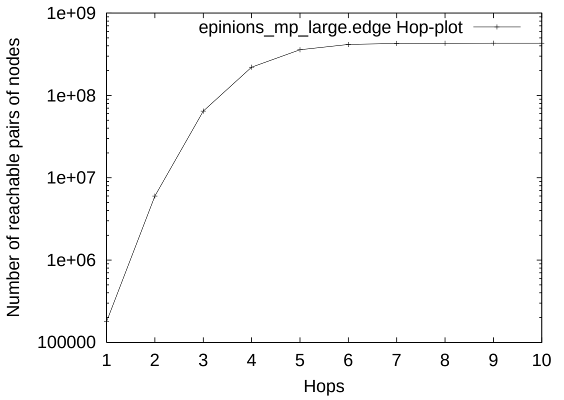
Count-vs-In-degree



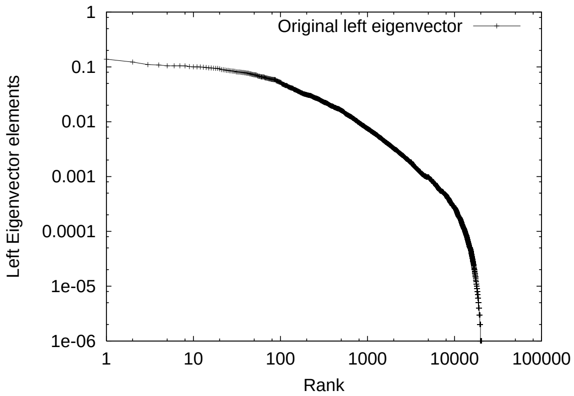
Count-vs-Out-degree



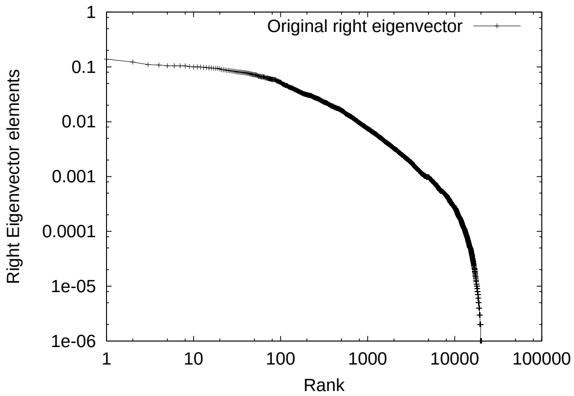
Eigenvalue-vs-Rank



Hop plot

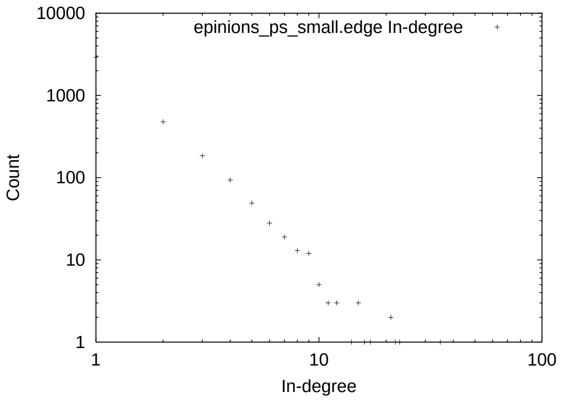


Left Eigen Vector

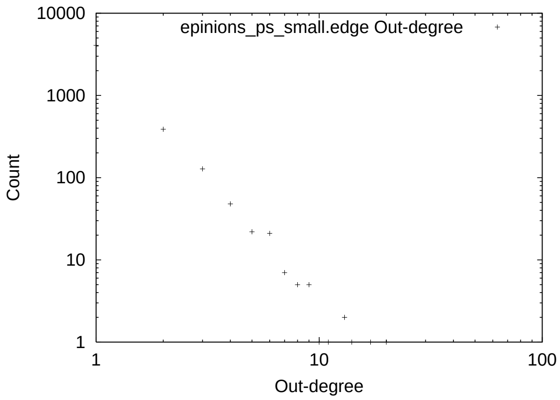


Right Eigen Vector

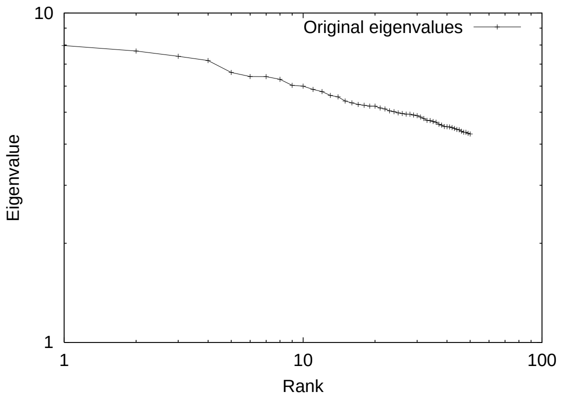
Figure 10: Epinions: sampling by mp - large size



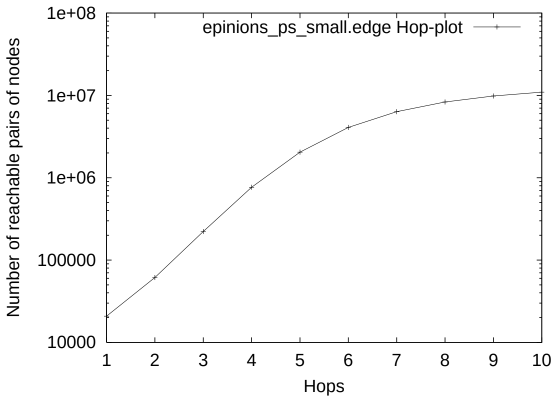
Count-vs-In-degree



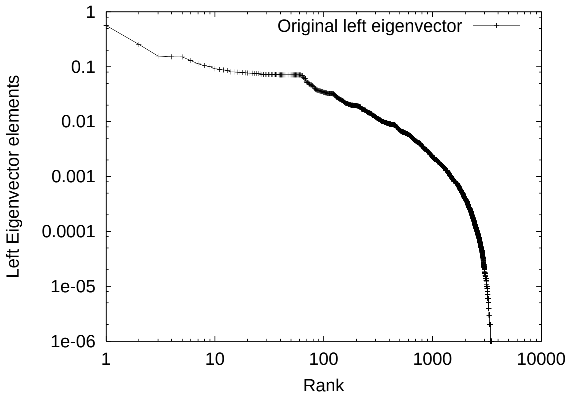
Count-vs-Out-degree



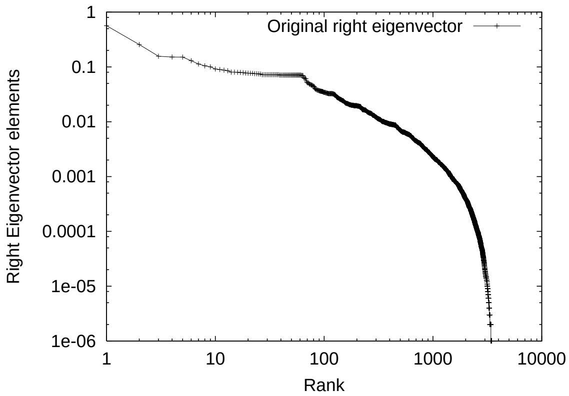
Eigenvalue-vs-Rank



Hop plot

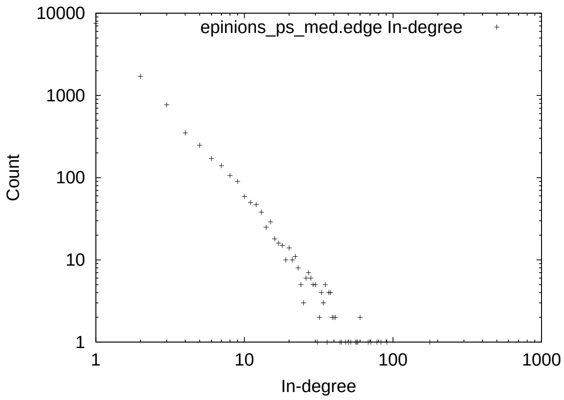


Left Eigen Vector

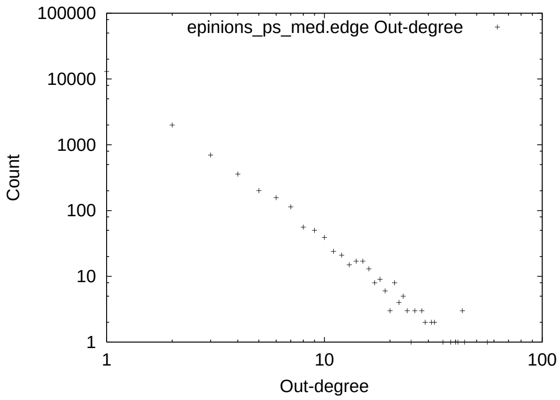


Right Eigen Vector

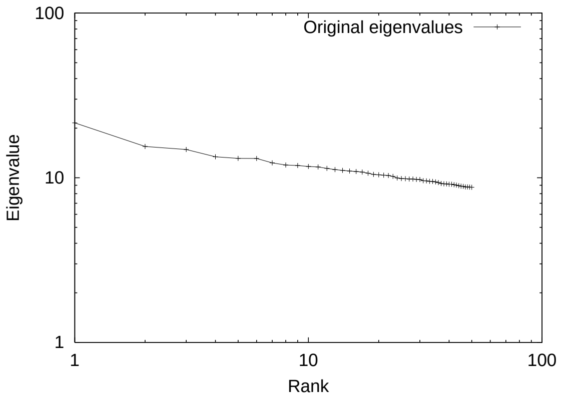
Figure 11: Epinions: sampling by ps - small size



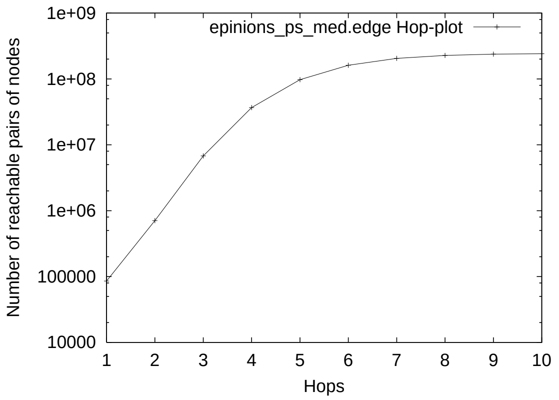
Count-vs-In-degree



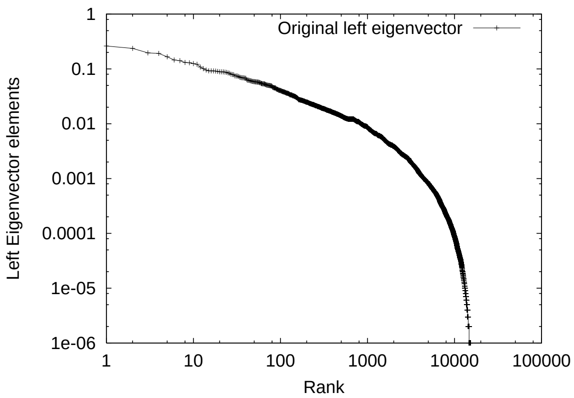
Count-vs-Out-degree



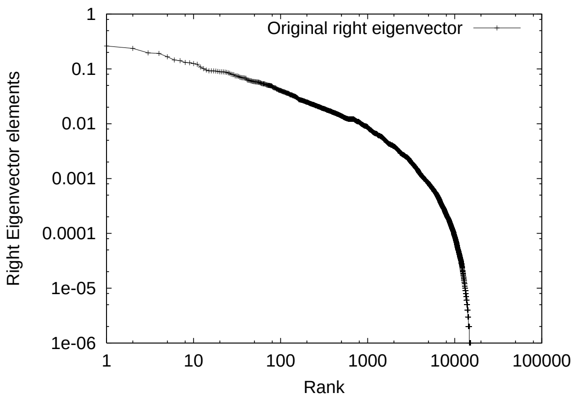
Eigenvalue-vs-Rank



Hop plot

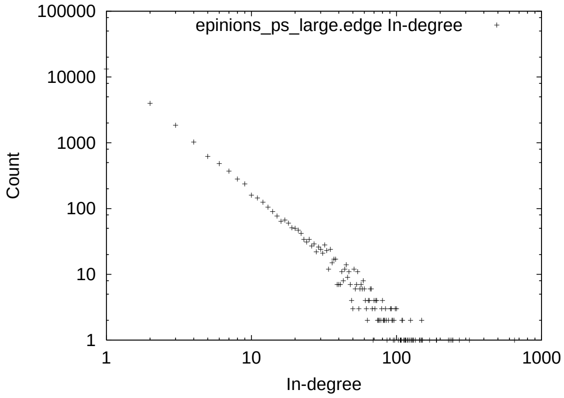


Left Eigen Vector

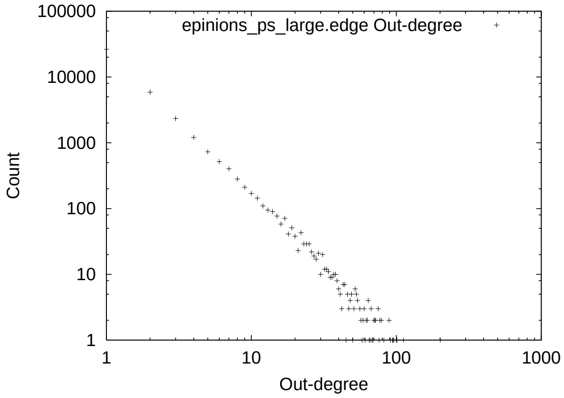


Right Eigen Vector

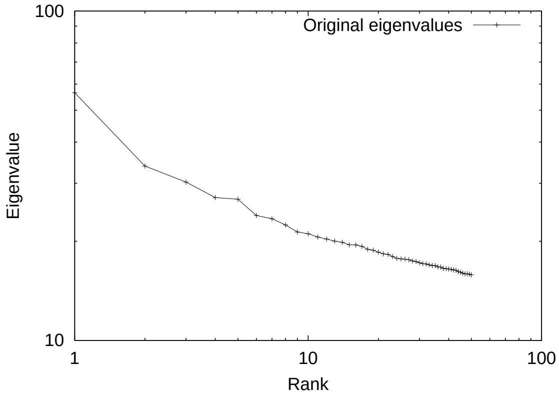
Figure 12: Epinions: sampling by ps - med size



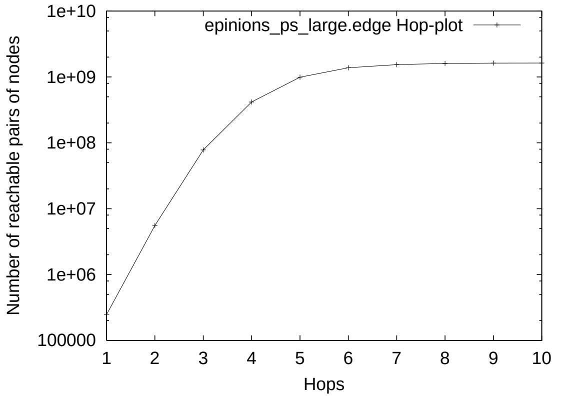
Count-vs-In-degree



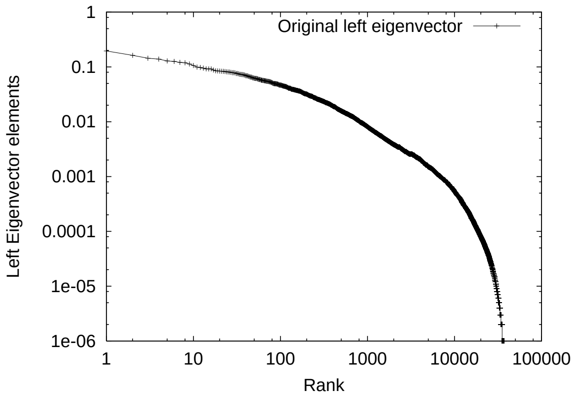
Count-vs-Out-degree



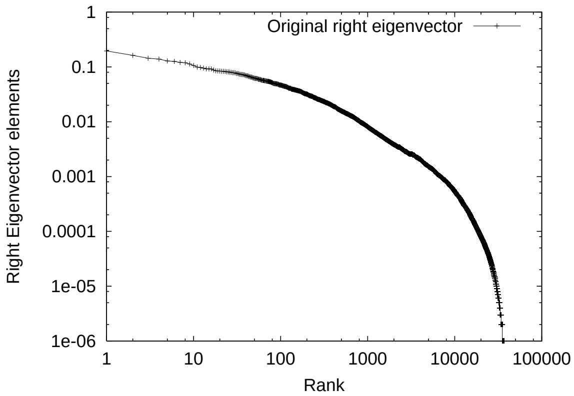
Eigenvalue-vs-Rank



Hop plot

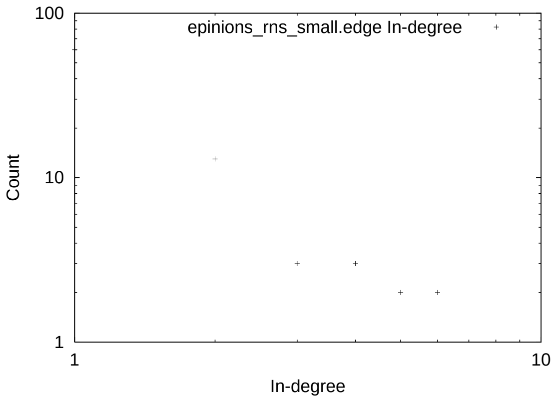


Left Eigen Vector

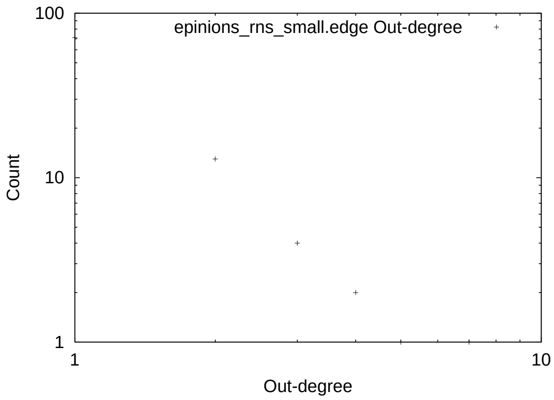


Right Eigen Vector

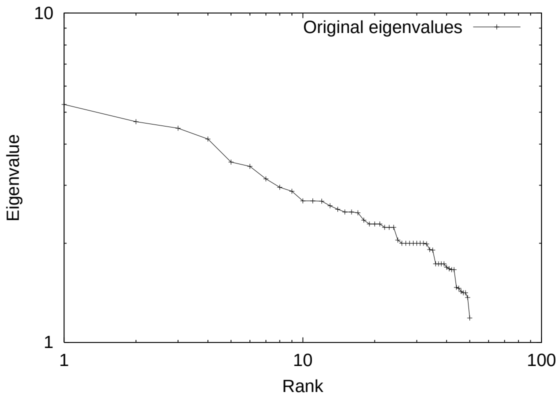
Figure 13: Epinions: sampling by ps - large size



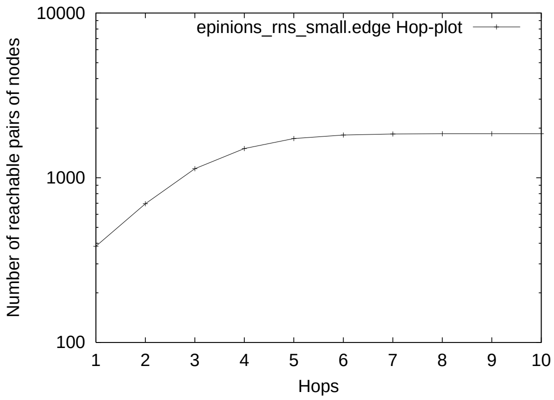
Count-vs-In-degree



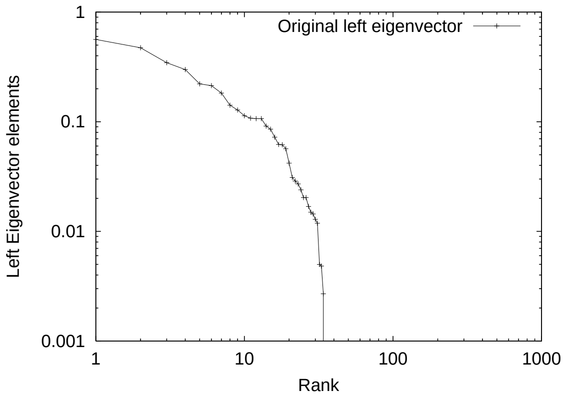
Count-vs-Out-degree



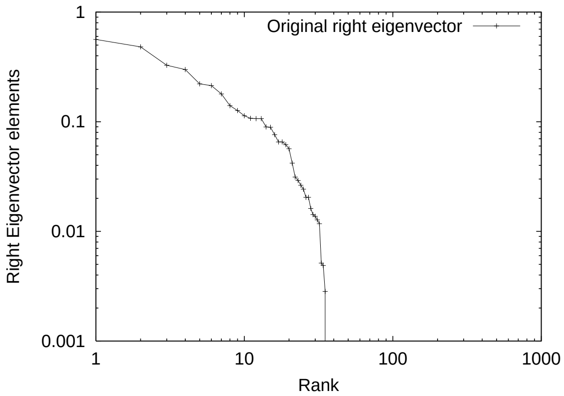
Eigenvalue-vs-Rank



Hop plot

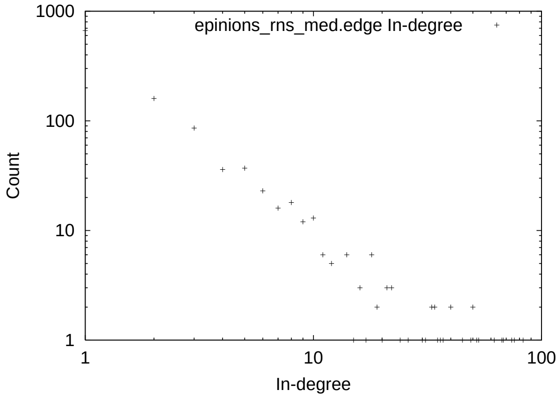


Left Eigen Vector

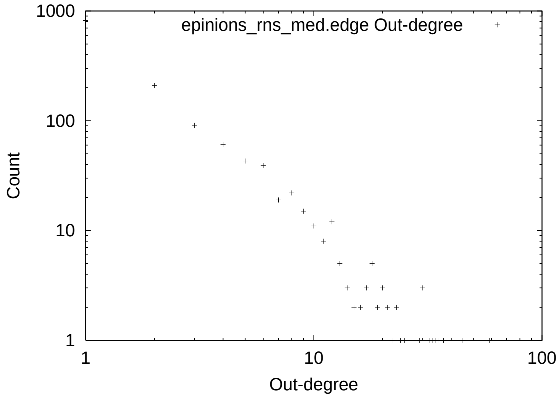


Right Eigen Vector

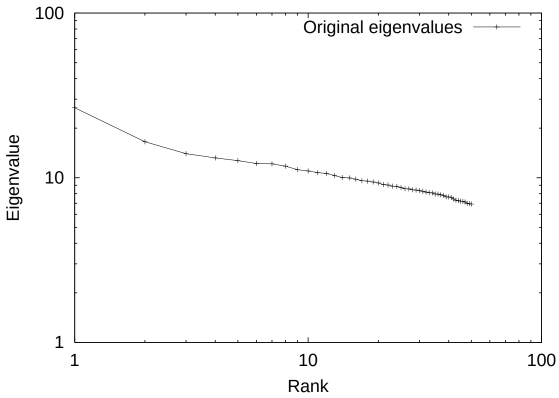
Figure 14: Epinions: sampling by rns - small size



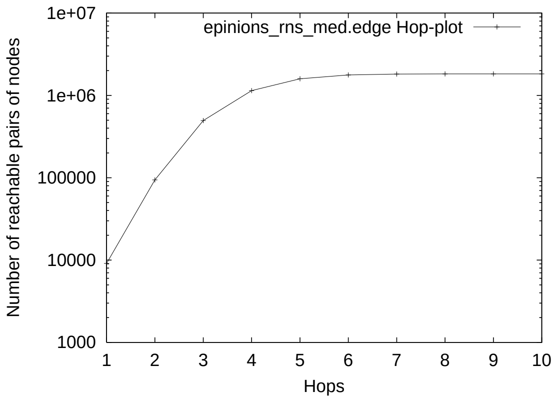
Count-vs-In-degree



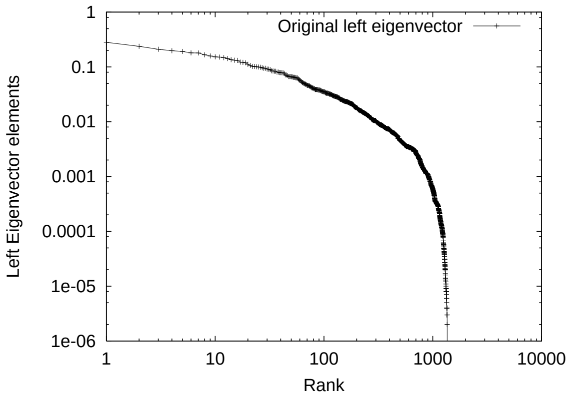
Count-vs-Out-degree



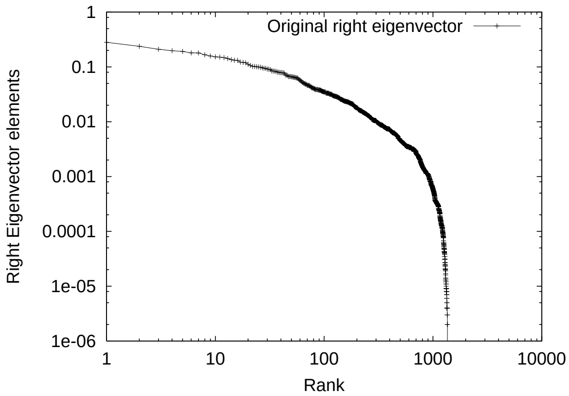
Eigenvalue-vs-Rank



Hop plot

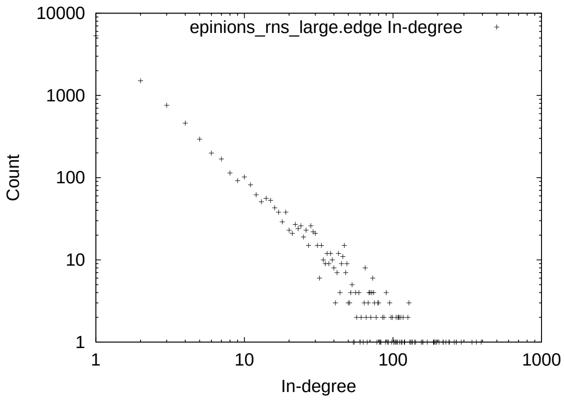


Left Eigen Vector

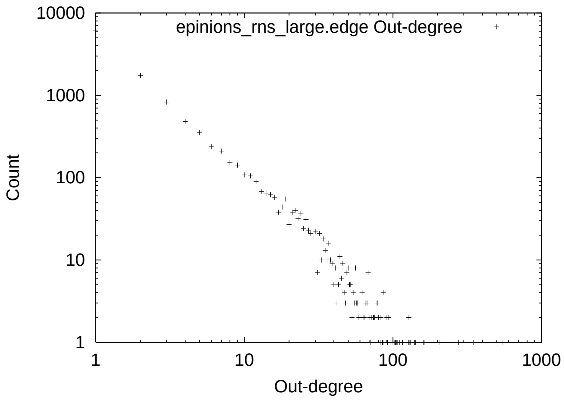


Right Eigen Vector

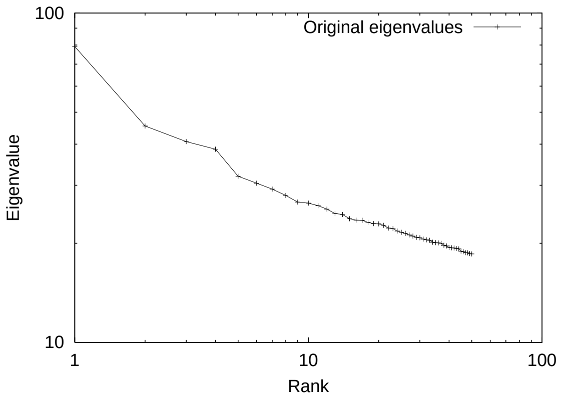
Figure 15: Epinions: sampling by rns - med size



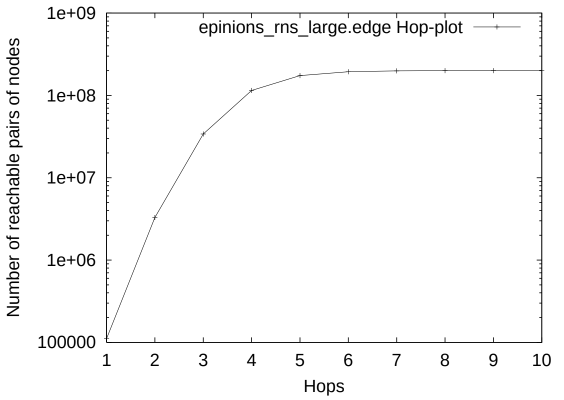
Count-vs-In-degree



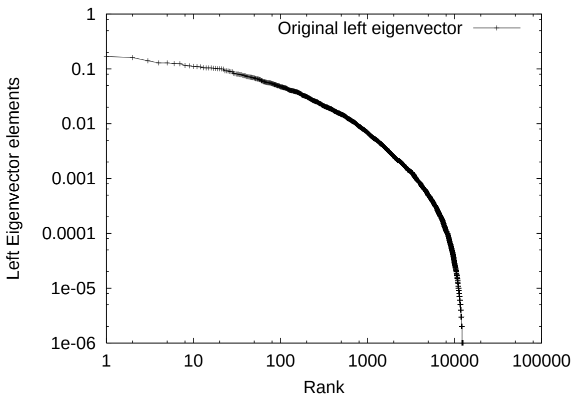
Count-vs-Out-degree



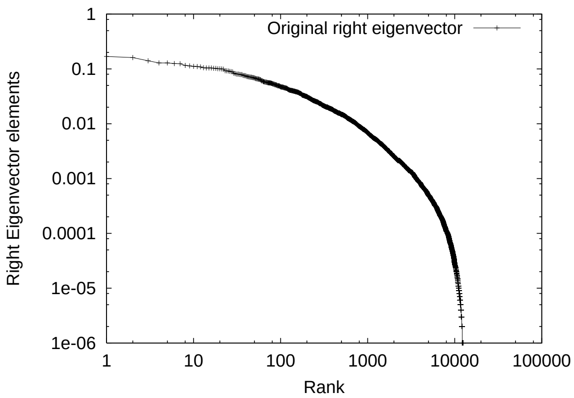
Eigenvalue-vs-Rank



Hop plot

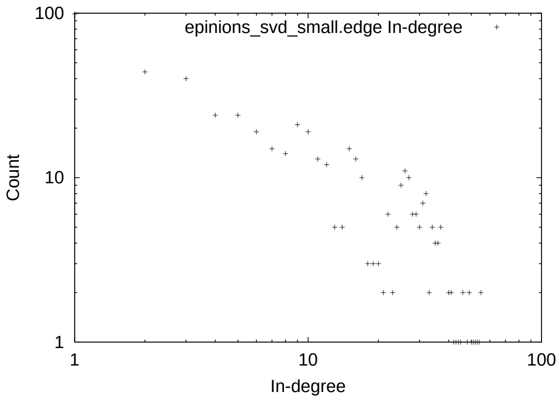


Left Eigen Vector

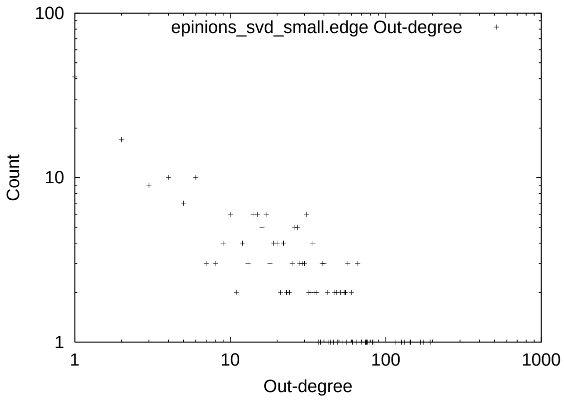


Right Eigen Vector

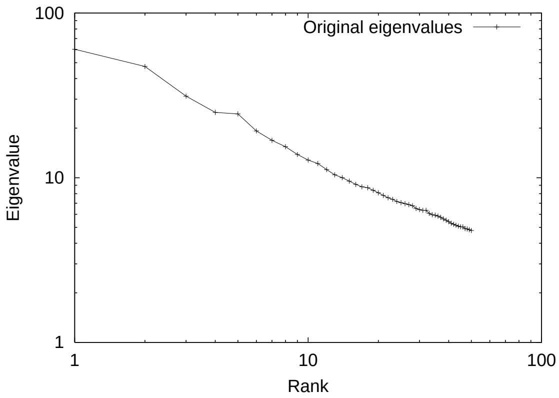
Figure 16: Epinions: sampling by rns - large size



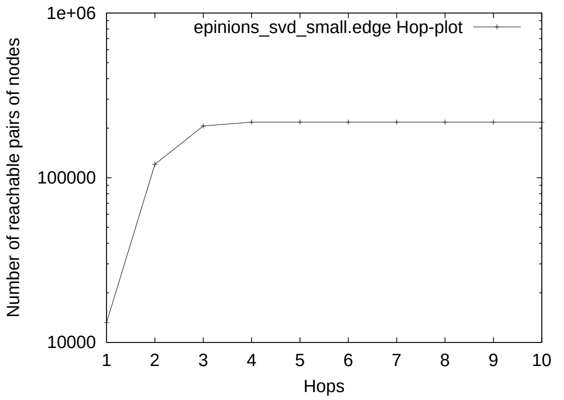
Count-vs-In-degree



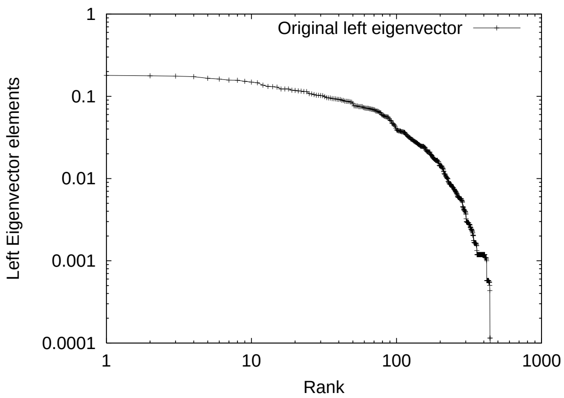
Count-vs-Out-degree



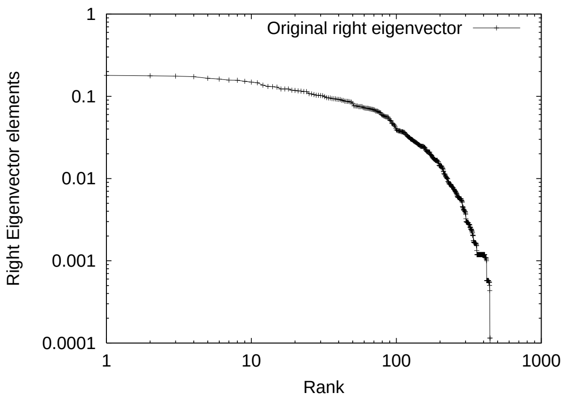
Eigenvalue-vs-Rank



Hop plot

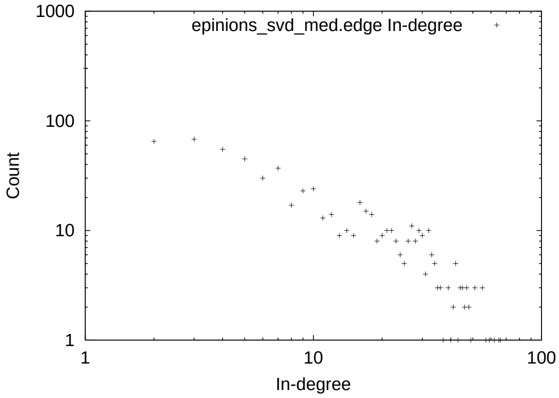


Left Eigen Vector

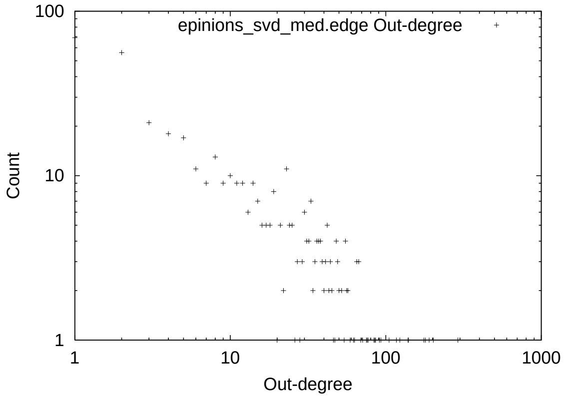


Right Eigen Vector

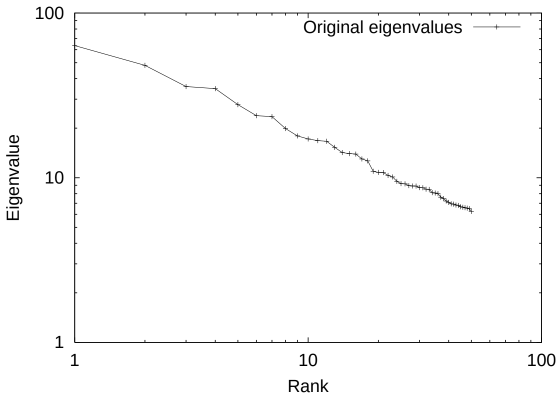
Figure 17: Epinions: sampling by svd - small size



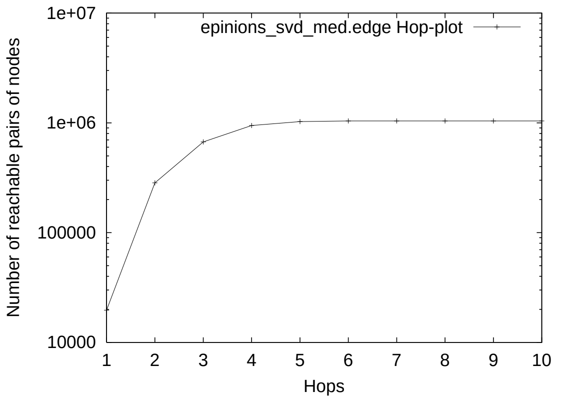
Count-vs-In-degree



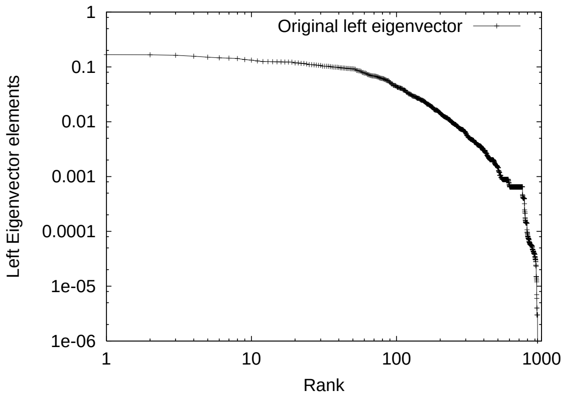
Count-vs-Out-degree



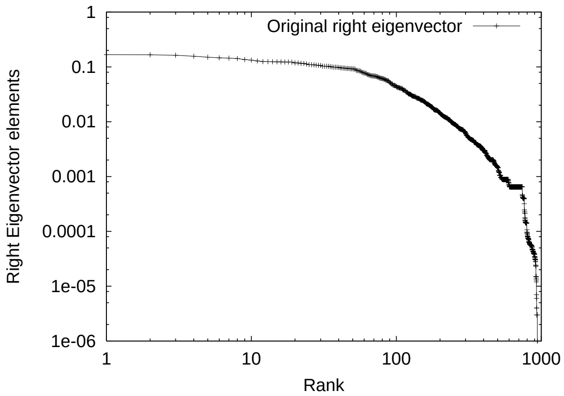
Eigenvalue-vs-Rank



Hop plot

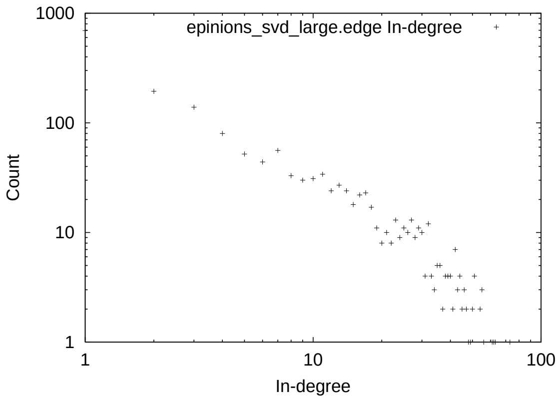


Left Eigen Vector

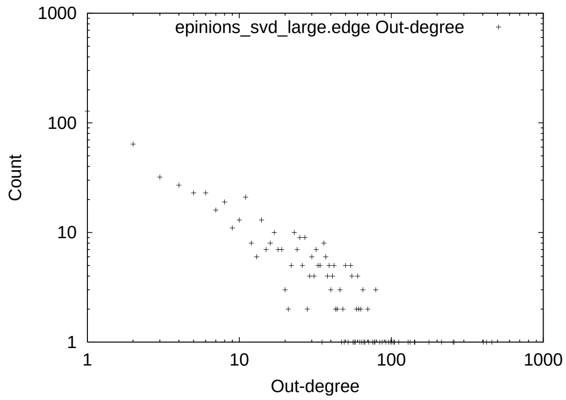


Right Eigen Vector

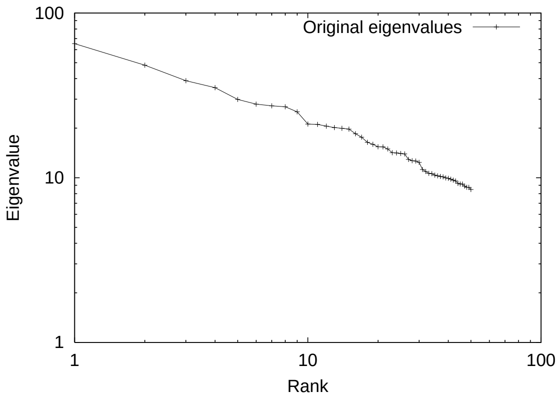
Figure 18: Epinions: sampling by svd - med size



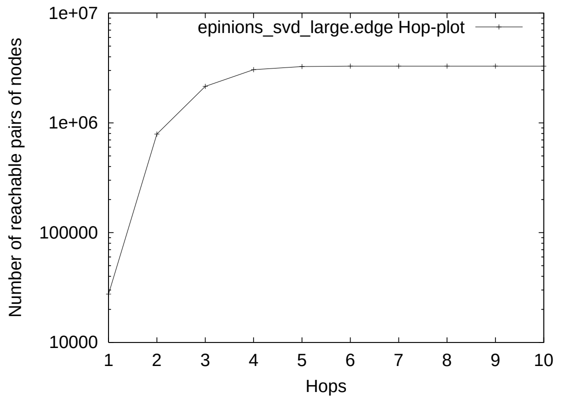
Count-vs-In-degree



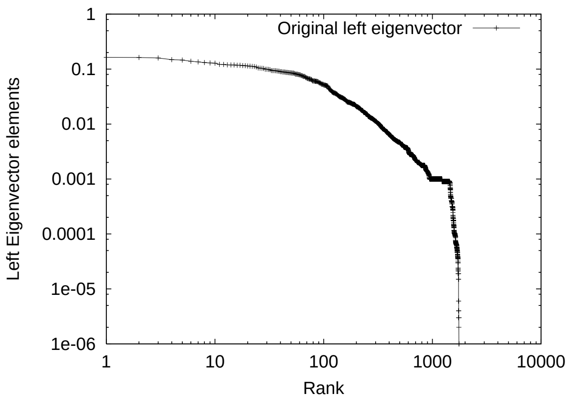
Count-vs-Out-degree



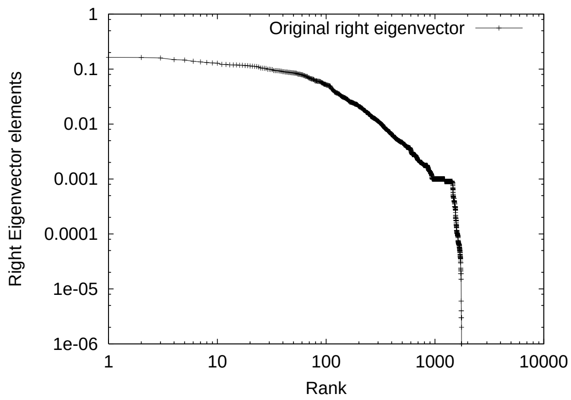
Eigenvalue-vs-Rank



Hop plot

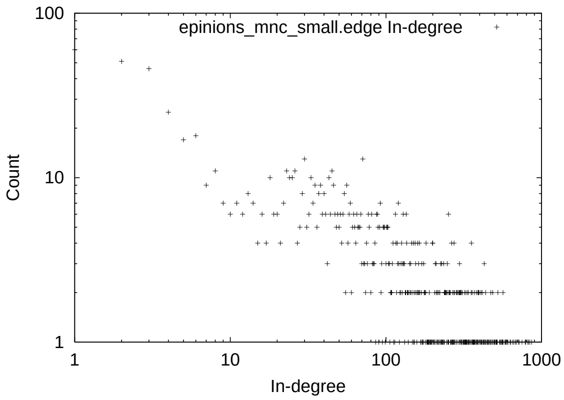


Left Eigen Vector

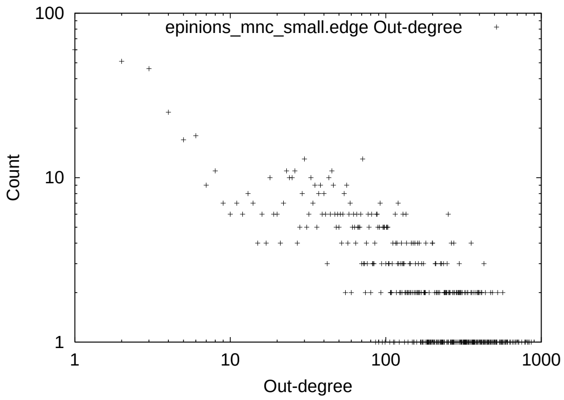


Right Eigen Vector

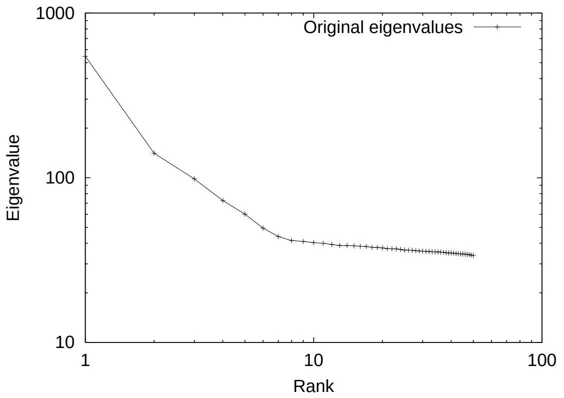
Figure 19: Epinions: sampling by svd - large size



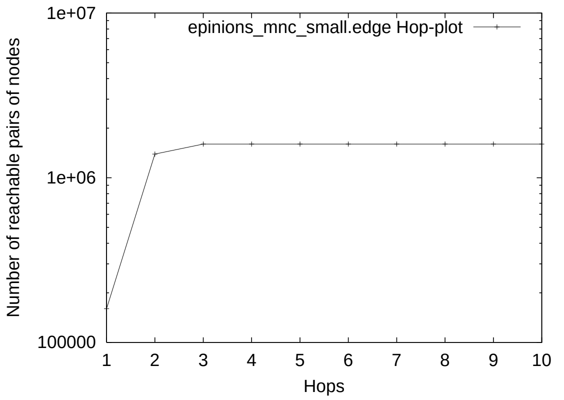
Count-vs-In-degree



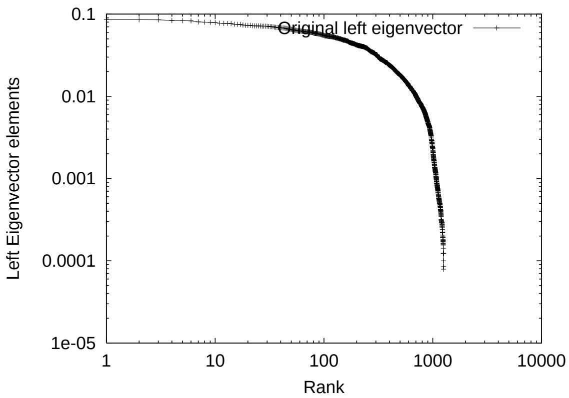
Count-vs-Out-degree



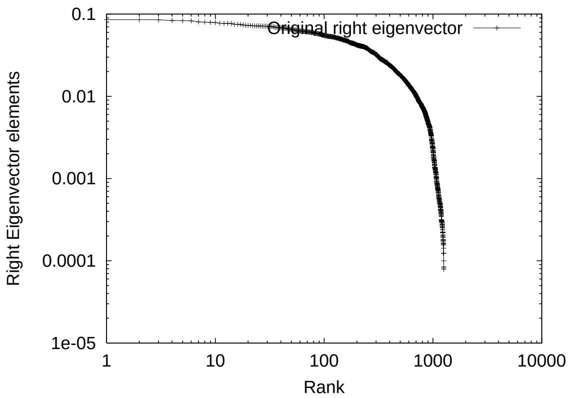
Eigenvalue-vs-Rank



Hop plot

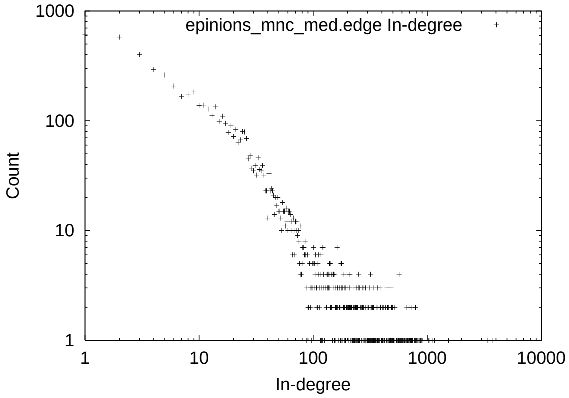


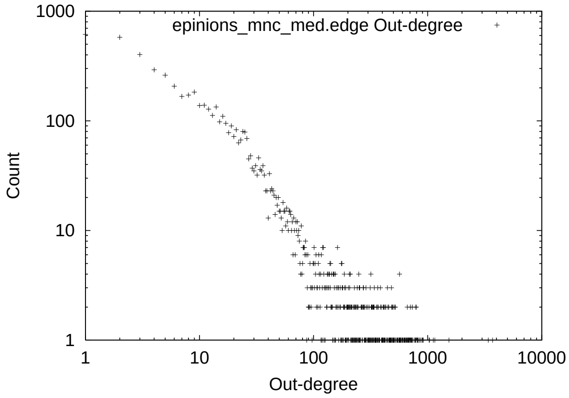
Left Eigen Vector



Right Eigen Vector

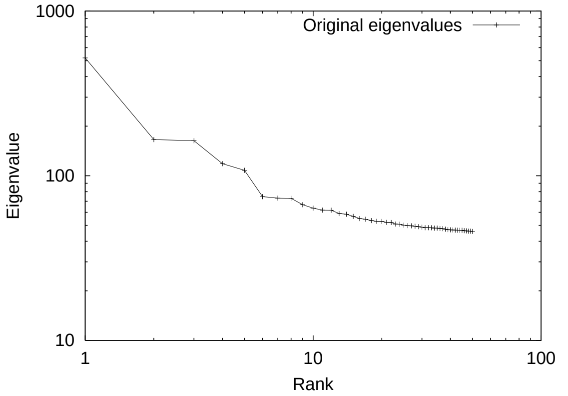
Figure 20: Epinions: sampling by mnc - small size

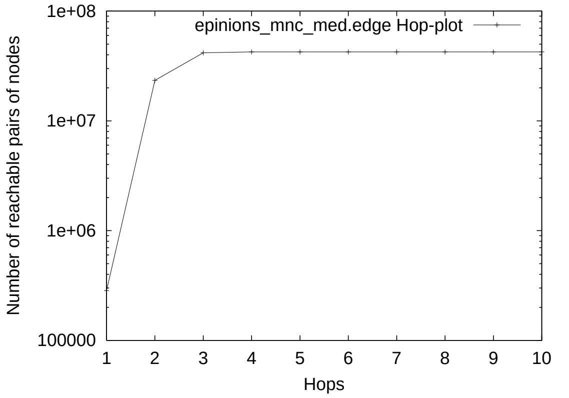




Count-vs-In-degree

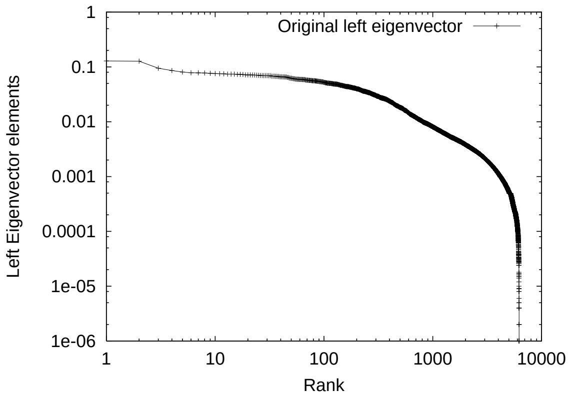
Count-vs-Out-degree

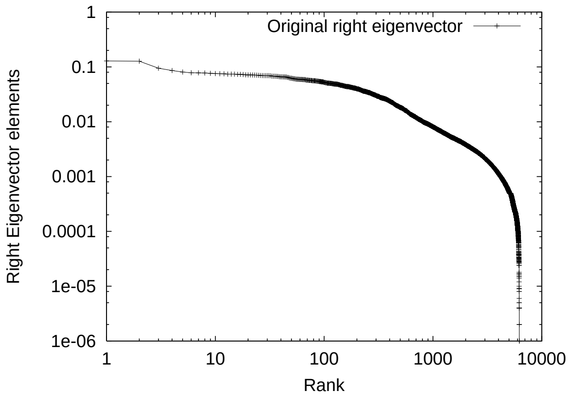




Eigenvalue-vs-Rank

Hop plot





Left Eigen Vector

Right Eigen Vector

Figure 21: Epinions: sampling by mnc - med size

[14] Fan Chung, Linyuan Lu, and Van Vu. Eigenvalues of random power law graphs. In *submitted*, 2003.

[15] Giuseppe DiBattista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry: Theory and Applications*, (4):235–282, 1994.

[16] Stephen Dill, S. Ravi Kumar, Kevin S. McCurley, Sridhar Rajagopalan, D. Sivakumar, and Andrew Tomkins. Self-similarity in the web. In *The VLDB Journal*, pages 69–78, 2001.

[17] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

[18] T.M.J. Fruchterman and E.M. Reingold. Graph drawing by force-directed placement. *Software Practice and Experience*, 21(11):1129–1164, 1991.

[19] P. Gajer, M. Goodrich, and S. Kobourov. A fast multi-dimensional algorithm for drawing large graphs, 2000.

[20] Oded Goldreich. Combinatorial property testing (a survey). In *Electronic Colloqium on Computational Complexity*, 1997.

[21] Google. Google. `http://www.google.com`.

[22] J. Guillaume, M. Latapy, and L. Viennot. Efficient and simple encodings for the web graph, 2002.

[23] Brian Hayes. Graph theory in practice: Part II. *American Scientist*, March-April, 2002.

[24] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report TR 95-035, 1995.

[25] George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.

[26] Jon Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.

[27] Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S. Tomkins. The Web as a graph: Measurements, models and methods. *Lecture Notes in Computer Science*, 1627, 1999.

[28] Vaishnavi Krishnamurthy, Junhong Sun, Michalis Faloutsos, and Sudhir Tauro. Sampling internet topologies: How small can we go? In *submitted to The 4th International Conference on Internet Computing*, 2003.

[29] V. Kumar and E. J. Schwabe. Improved algorithms and data structures for solving graph problems in external memory. pages 169–177.

[30] Anil Maheshwari and Norbert Zeh. I/o-efficient algorithms for graphs of bounded treewidth. In *Symposium on Discrete Algorithms*, pages 89–90, 2001.

[31] M. S. Marshall, I. Herman, and G. Melançon. An object-oriented design for graph visualization. *Software Practice and Experience*, 31(8):739–756, ???? 2001.

[32] Kurt Mehlhorn and Ulrich Meyer. External-memory breadth-first search with sublinear i/o.

[33] Metis. Metis. `http://www-users.cs.umn.edu/~karypis/metis/`.

[34] Tamara Munzner. *Interactive Visualization of Large Graphs and Networks.* PhD thesis, Stanford University, 2000.

[35] Christopher R. Palmer, Phillip B. Gibbons, and Christos Faloutsos. Anf: A fast and scalable tool for data mining in massive graphs. In *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.

[36] K. Randall, R. Stata, R. Wickremesinghe, and J. Wiener. The link database: Fast access to graphs of the web, 2001.

[37] Torsten Suel and Jun Yuan. Compressing the graph structure of the web. In *Data Compression Conference*, pages 213–222, 2001.

[38] Hongsuda Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network topology generators: Degree-based vs. structural. In *ACM SIGCOMM*, 2002.

[39] Sudhir L. Tauro, Christopher Palmer, Georgos Siganos, and Michalis Faloutsos. A simple conceptual model for the internet topology. In *Global Internet*, 2001.

[40] Teoma. Teoma. `http://www.teoma.com`.

[41] Graham J. Wills. NicheWorks — interactive visualization of very large graphs. *Journal of Computational and Graphical Statistics*, 8(2):190–??, 1999.

[42] Jared Winick and Sugih Jamin. Inet-3.0: Internet topology generator.

Contents