Compressing Network Graphs

Anna C. Gilbert
AT&T Labs-Research
180 Park Avenue
Florham Park, NJ 07932
agilbert@research.att.com

Kirill Levchenko[*]
University of California, San Diego
9500 Gilman Drive Dept 0114
La Jolla CA 92093-0114
klevchen@cs.ucsd.edu

ABSTRACT

Graphs form the foundation of many real-world datasets ranging from Internet connectivity to social networks. Yet despite this underlying structure, the size of these datasets presents a nearly insurmountable obstacle to understanding the essential character of the data. We want to understand "what the graph looks like;" we want to know which vertices and edges are important and what are the significant features in the graph. For a communication network, such an understanding entails recognizing the overall design of the network (e.g., hub-and-spoke, mesh, backbone), as well as identifying the "important" nodes and links.

We define several compression schemes, including vertex similarity measures and vertex ranking. We present a system for condensing large graphs using both auxiliary information (such as geographic location and link type in the case of communication networks), as well as purely topological information. We examine the properties of these compression schemes, demonstrate their effects on visualization, and explore what structural graph properties they preserve when applied to both synthetic and real-world networks.

1. INTRODUCTION

Massive graphs are everywhere, from social and communication networks to the World Wide Web. The geometric representation of the graph structure imposed on these data sets provides a powerful aid to visualizing and understanding the data. However even with this aid, these graphs are too large to comprehend without additional processing. In this paper, we consider creating graphical summaries of the large graphs arising in the domain of enterprise IP networks. Our methods do not limit us to this domain; rather, this domain is rich in large highly engineered graphs.

We do this by transforming the original graph into a smaller one using structural features of the graph that have well-understood semantics in our domain. The resulting smaller, and usually annotated, graph can then be visualized, and, we hope, understood, by a human being confronted with such a dataset. We refer to this process as graph compression, or, more properly, semantic graph compression, to distinguish it from algorithmic graph compression where a graph is compressed in order to reduce the time or space complexity of a graph algorithm.

We define two categories of compression schemes, one based on the notion of vertex importance and another on the notion of vertex similarity, which we describe in Section 3, after discussing related work in the next section (Sec. 2). In Section 4 we experimentally verify our results on two synthetic and four real-world datasets, using Arc [6], a graph compression tool expressly developed for this purpose, and whose description we omit due to space constraints. Finally, Section 5 concludes the paper.

[*] A significant portion of this work was done while Kirill Levchenko was visiting AT&T Labs–Research.

2. RELATED WORK

Algorithmic Compression. Previous work in graph compression has been concerned exclusively with compressing graphs for input to algorithms, where the (smaller) compressed representation preserves some property used by the algorithm.

Feder and Motwani [5] consider transforming a graph into a smaller one (in terms of the number of vertices and edges) that preserves certain properties of the original graph, such as connectivity. A graph compressed in this manner is suitable for (more efficiently) computing certain graph functions, such all-pairs shortest paths. Although our shared medium compression scheme has similarities with this work, our goal is to preserve the *semantics* of the original graph, rather than its algorithmic properties.

Adler and Mitzenmacher [1] and Suel and Yuan [12] consider losslessly compressing the Web Graph for efficient search engine storage and retrieval. Although it differs from our work in that only the representation, and not the actual graph, is compressed, their compression process itself may reveal semantically-relevant information.

The Web Graph. The link structure of the World Wide Web has seen considerable study in the context of web search, an important step of which is ranking the results in order of relevance. The most widely studied algorithms for this—PageRank [10], HITS [7], and SALSA [9]—use a Markov model, motivated by the image of a web surfer visiting web pages by following links or jumping to a random page. Because our graph is undirected, such a process does not reveal additional information; however other spectral techniques, which we did not investigate, may have applications to our problem.

Recently, White and Smyth [14] introduced a number of new notions of vertex importance using both paths and random walks. Our paths scheme is a specialization of their weighted paths algorithms.

In [8], Kumar *et al.* infer certain relationships between nodes in the Web Graph based on the link structure. (The same idea is also found in Kleinberg's HITS [7], which infers hub-authority relationships between websites.) Our redundant vertex elimination scheme is based on this idea of recognizing local structures.

IP Network Measurement. Faloutsos *et al.* in [4] reported on the statistical properties of the autonomous system network graph as well as Internet router graphs. While their goal is to characterize the statistical properties of the network graph, we are interested in understanding a particular instance of a network. Nonetheless, their work has led to the emergence of Internet topology generators, one of which, `inet-3.0` [15], was used to generate one of our synthetic datasets.

Spring, Mahajan, and Wetherall in [11] used novel techniques to derive approximate real-world IP topologies. We use several of their publicly available datasets in experiments.[1]

Graph Visualization. Our work uses `neato`, a part of GraphViz [2], which is a publicly available graph visualization tool developed at AT&T Labs–Research. GraphViz also includes `gvpr`, a graph processing tool that has some of functionality of Arc [6], our graph compression tool.

3. COMPRESSION SCHEMES

With the application of communication network visualization in mind, let us define a compression scheme to be a transformation of a network graph into a much smaller, possibly annotated, graph that preserves the salient characteristics of the original network. What these might be is, necessarily, an elusive concept; nevertheless we will take these to be characteristics *that have a straightforward, real-world interpretation in the domain of communication networks.* One obvious such characteristic is connectivity: if an underlying network is connected, the compressed network should be connected also. Therefore, we shall attempt to justify our compression schemes based on their real-world interpretation.

One way to classify compression schemes is into those that are purely topological, that is, those that rely only on the underlying network graph, and into those that rely on additional vertex or edge attributes. In an IP network, for example, vertices (hosts) may be labeled as routers or workstations: for such a network a reasonable compressed representation may be the network graph induced by the set of router vertices. While such a compression scheme is inherently more powerful than a purely topological one, its requisite information may not always be available.

Orthogonal to the classification above, we have identified two basic types of compression schemes: those that compress the network based on a notion of node importance, and those based on a notion of similarity. An importance compression scheme ranks vertices in order of importance (or based on an importance predicate), producing the induced graph as the compressed network. Such compression schemes are presented in section 3.1. A similarity compression scheme, on the other hand, combines similar vertices into a single vertex in some manner; these schemes are covered in section 3.2. We let $G = (V, E)$ denote the network graph G with vertex set V and edge set E in what follows.

[1]Our affiliation should not be seen as an endorsement of the accuracy of Rocketfuel data.

3.1 Importance Compression Schemes

The notion of a node's importance in a graph has received considerable attention in the domain of Web graphs (in the context of ranking Web search results), as well as in the domain of social networks. We consider importance to be a weight function on vertices, and describe three such functions: two based on degree and one based on shortest paths in the graph. We then consider how an importance notion can be used to compress a graph in Section 3.1.3.

3.1.1 Degree-based Importance

The degree of a node is arguably one of the simplest and most intuitive measures of importance in a communication network. One may reasonably posit that the greater the degree, the more important the network node, especially in view of the preferential attachment interpretation of the power law phenomenon. There are a number of essentially equivalent ways to define such a weight function; let us choose

$$w_{\text{deg}}(v) \equiv |\{u \in V \;:\; \deg(u) \le \deg(v)\}|/|V|,$$

which gives a weight in the range $(0, 1]$.

Unfortunately, w_{deg} tends to favor dense components of a graph because the weight of a vertex is its relative rank compared to all the vertices in the graph. For this reason, we developed a localized variant:

$$w_{\text{beta}}(v) \equiv |\{u \in \text{N}(v) \;:\; \deg(u) \le \beta \cdot \deg(v)\}|/|\text{N}(v)|,$$

where β is a parameter and $\text{N}(v)$ is the set of vertices adjacent to v. The Beta weight function assigns weight relative to the neighborhood of a vertex, rather than the whole graph.

3.1.2 Shortest Path-based Importance

Distance is another fundamental graph property that has a natural interpretation in the domain of IP networks, and that is as the number of hops between two routers. It seems reasonable, then, to consider weight functions based on this quantity. Recall that the *eccentricity* of a vertex is defined to be the maximum distance between itself and another vertex. It can be thought of as a measure of how far the vertex is from the "center" of the graph. However because a communication network may not have a meaningful "center," we did not further pursue such a weight function.

Instead, we considered the following weight function. Define the *Path weight* w_{path} of a vertex (sometimes also called its *betweenness* in social networks) to be the number of shortest paths between any two (not necessarily distinct) vertices in the graph through the vertex, divided by the square of the total number of vertices in the graph; when there are multiple shortest paths between a pair of vertices, each is weighted equally so that the sum of the weights of all shortest paths between the two vertices is 1. More formally, define

$$w_{\text{path}}(v) \equiv \sum_{x,y \in V} \frac{|\{\pi \in \Pi(x,y) \;:\; v \in \pi\}|}{|V|^2 |\Pi(x,y)|},$$

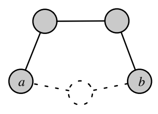

Figure 1: Running KeepOne on the graph above, with the set of important vertices K_1, shown shaded results in the removal of the dotted vertex, increasing the minimum distance between a and b by 1.

where $\Pi(x, y)$ is the set of all shortest paths between x and y. Thus the Path weight function, also called *betweenness* in social networks, favors vertices that lie on a shortest path a packet would take traveling between hosts in the network.[2]

3.1.3 Using Importance to Compress

Important vertices alone may not capture the topological features of the network graph. For example, the graph induced by the important vertices may be disconnected, even if the original graph is not. If this is the case, at the very least, we must reconnect the important vertices. We considered two ways of doing this. The first approach is to add the minimal number of vertices. That is, if K_1 is the set of important vertices, our goal is to find the minimal set K_2 such that the graph induced by $K_1 \cup K_2$ is connected. Unfortunately, this problem is NP-Complete. Our approximation algorithm, KeepOne, works by building a minimum spanning tree on the complete graph on K_1 where an edge (u, v) has weight equal to the length of a shortest path from u to v. The set K_2 then consists of any additional vertices along any "path" edge in the minimum spanning tree. The result is the graph induced by the vertices $K_1 \cup K_2$.

Unfortunately, while KeepOne preserves the connectivity of the original graph, it does not preserve distances between "important" vertices. Consider, for example, the graph in Figure 1. The important vertices (K_1) are shown shaded. Because the graph induced by K_1 is already connected, KeepOne will exclude the vertex shown with dotted lines, increasing the minimum distance between a and b by 1.

Our second algorithm, KeepAll, shown in Figure 2 rectifies this by keeping vertices that lie along a shortest path between any two vertices in K_1, preserving all minimum distances between important vertices.

3.2 Similarity Compression Schemes

In this section, we consider the second class of compression schemes which use a similarity measure or relation to combine similar vertices into one. A well-chosen similarity measure may lead to substantially less cluttered graphs at a minimal loss of information. We may derive this similarity measure from purely topological information or from vertex or edge attributes that are included in the dataset. A typical vertex attribute in network graphs is the geographic location of the router. Sometimes, we also have edge attributes such as link type and address which determine if it is a shared medium link (*e.g.*, Ethernet).

[2]However it is worth noting that in real IP networks a packet often does not take a shortest path through the network; see, e.g., [13].

```
KeepAll($G, K_1$):
    $K_2 \leftarrow \emptyset$
    For $u, v \in K_1$:
        $K_2 \leftarrow K_2 \cup \text{ShortestPath}_G(u, v)$
    $V' \leftarrow K_1 \cup K_2$
    $E' \leftarrow E_G \cap (V' \times V')$
    Return $(V', E')$
```

Figure 2: The KeepAll algorithm takes a graph G and a set K_1 of important vertices. It return the subgraph induced by the vertices K_1, combined with the vertices connecting them along a shortest path.

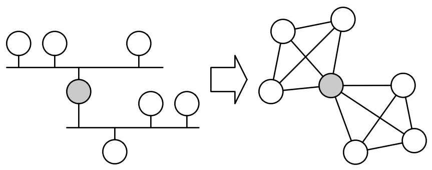

Figure 3: Hosts on a shared medium link appear as vertices in a clique in the network graph.

3.2.1 Redundant Vertex Elimination

A natural similarity measure on a pair of vertices is the number of neighbors they have in common. In fact, this has a natural interpretation in the domain of communication networks: important routers are sometimes duplicated for redundancy, and appear as a pair of vertices with nearly identical neighbors. Our redundant vertex elimination scheme merges two vertices having some minimum number of common neighbors into a single vertex which inherits their edges. When a vertex v shares many common neighbors with two distinct vertices u and w, the pair (u and v or v and w) with the greater number of common neighbors is chosen.

3.2.2 Geographic Clustering

In some cases, the network nodes may be labeled with geographic information. In this case, the similarity relation simply encodes the equivalence class of being in the same geographic region. This can be a highly effective compression scheme for understanding the global topology of a network.

3.2.3 Shared Medium Clustering

Local-area networks such as Ethernet connect communicating hosts on a shared medium, such that all hosts on the link can talk to each other. In the network graph, this appears as a clique (see Fig. 3), and introduces unnecessary edges into the graph. These cliques also run counter to network engineers' intuition of what the network looks like. When the link type and address is known, the Shared Medium Clustering scheme identifies such cliques and merges each into a single vertex. If a vertex is on two distinct shared medium links, we connect the resulting shared medium vertices with an edge (see Fig. 4).

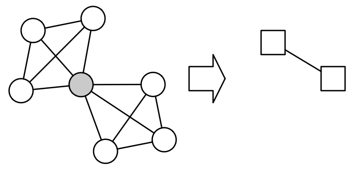

Figure 4: We eliminate unsightly cliques resulting from shared medium links by merging the vertices into a single vertex. The shaded vertex, which is connected to two shared medium networks, is represented by the edge in the resulting graph.

4. EXPERIMENTAL RESULTS

Because these algorithms are in part exploratory data analysis tools, it is not clear how to compare one compression scheme to another. With our definition of compression, there is no metric of bits per vertex or edge, nor is there a natural distortion metric with which to measure the proximity of the compressed graph to the original. It is also not clear what effect the different tunable parameters in each compression algorithm will have on a particular graph. To obtain a better understanding of the behavior of our compression schemes on large network graphs and to evaluate their performance, we apply them to both real and synthetic datasets.

4.1 Experimental Setup

To implement the compression schemes presented in Section 3, we developed Arc [6], a tool designed expressly for this purpose. Here, we describe the settings used in the experiments, as well as any implementation details. In all cases, the graphs are undirected and connected. For some datasets, this required adding directed edges to balance one-way links, and removing all but the largest connected component.

Degree and Beta Weight. For w_{beta}, we set $\beta = 1.0$, and then took the 20 heaviest vertices (based on the weight function) as the set K_1 of important vertices.

Path Weight. Rather than computing the set $\Pi(x, y)$ of all shortest paths from x to y exactly, we used just one shortest between x and y path, chosen arbitrarily, in order to simplify implementation. Because of this, vertex weights may change between different executions of the algorithm. The set K_1 consisted of the 20 largest-weight vertices.

The KeepOne Algorithm. We implemented the KeepOne algorithm as described in Section 3.1.3. We broke ties for the shortest path arbitrarily, which in some cases may produce arbitrarily bad approximation factors.

The KeepAll Algorithm. We implemented the KeepAll algorithm as described in Section 3.1.3. Again, ties for the shortest paths were broken arbitrarily.

Redundant Vertex Elimination. We implemented Redundant Vertex Elimination (RVE) as described in Section 3.2, breaking ties arbitrarily. We set the parameter k (the minimum number of common neighbors) to 10.

Lacking certain information on the link types or the geographic location of the routers in our public datasets, we did not perform any experiments involving the Shared Medium Compression or Geographic Clustering schemes. This data may, however, be available in proprietary information.

We used six datasets: two synthetic graphs, **INET** and **ERDOS**, and four real networks, **AS7018**, **AS1294**, **AS2913** and **AS3356**. **INET** is a power law graph as generated by `inet-3.0` [15]. It consists of 3076 vertices and approximately 30% of those vertices have degree one. The degrees of the other vertices follow a power law as the entire graph is designed the capture the salient features of the AS graph in the Internet. The average degree in **INET** is 3.16. The second graph, **ERDOS**, is an Erdős-Renyi random graph [3]. In an Erdős-Renyi random graph, each pair of vertices is joined with an edge with probability p. As in **INET**, **ERDOS** has 3076 vertices and we set p so that the expected degree is 3.16 as in INET. The four real networks are publicly available from [11]. These datasets include a number of networking details. We chose to use the authors' classification of backbone routers so that we could judge the efficacy of our schemes in finding important vertices (assuming that backbone routers are important ones). We also chose to use the datasets for which the Rocketfuel authors are sure these routers belong to the autonomous system. These datasets are smaller and are more likely to have been designed with a special purpose in mind. All the data networks have several hundred vertices (ranging from 600 to 900) and they each have several thousand edges (from 4000 to 10000). They each have about 400 backbone routers.

4.2 Before and After

Figures 5–9 show each network before and after compression. We chose the after pictures on a purely visual basis–what figure seemed to capture the salient features of the original graph without also including spurious vertices and edges. No one compression scheme consistently yielded the best visual compression. In addition, the best visual compression scheme was not the scheme that pruned the largest fraction of edges and vertices although it did remove a significant fraction. We do not show the before and after figures for **ERDOS** as the compression schemes either removed few edges and vertices or they removed far too many. We are, in fact, heartened by this result as we do not expect to find significant structures in a random graph. We do, however, see that Beta weight with the KeepAll algorithm seems to bring out the highly variable degree structure of **INET**, a power law graph. This compression scheme highlights the highest degree vertex at the center and removes or decreases the proportion of degree one vertices.

Table 1 summarizes the compression factors for each graph and each compression scheme. It displays the original number of vertices, backbone vertices (where appropriate), and edges. It also shows how many vertices, backbone vertices, and edges (and their percentage of the original values) are retained under the different compression schemes. Over all networks RVE compresses the least. In fact, on the two random graphs, it performs no compression. Not surprisingly, the KeepAll algorithm retains more vertices and edges than the KeepOne algorithm as KeepAll includes all the vertices along a shortest path that connect important vertices. We also observe that a degree-based importance measure tends to compress the graph less than a path-based measure.

4.3 Finding Important Features

To evaluate our compression schemes, we relied on the

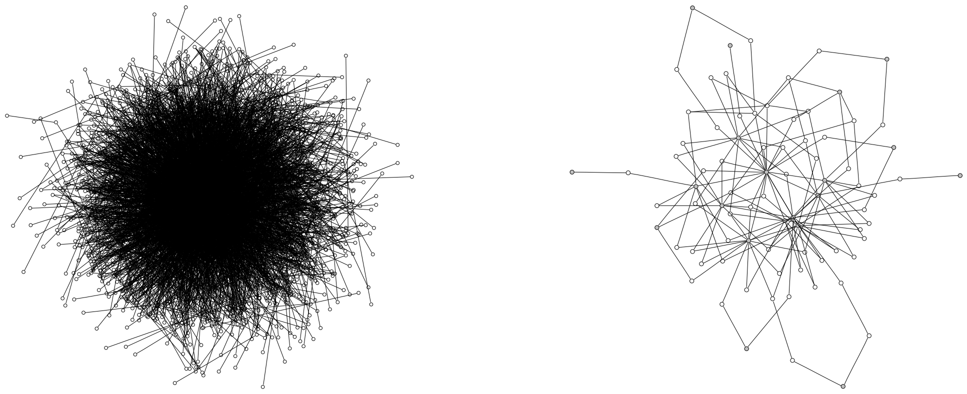

Figure 5: The INET graph before compression (left) and after compression using the Beta weight function with the KeepAll algorithm (right). In the compressed graph, important vertices are shown shaded.

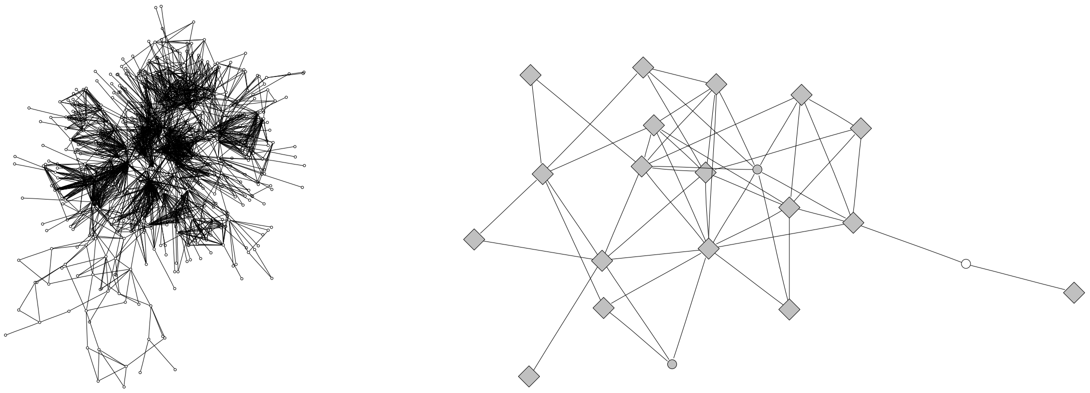

Figure 6: The AS1239 graph before compression (left) and after compression using Redundant Vertex Elimination followed by the Degree weight function with the KeepOne algorithm. In the compressed graph, merged vertices are shown as diamonds and important vertices are shown shaded.

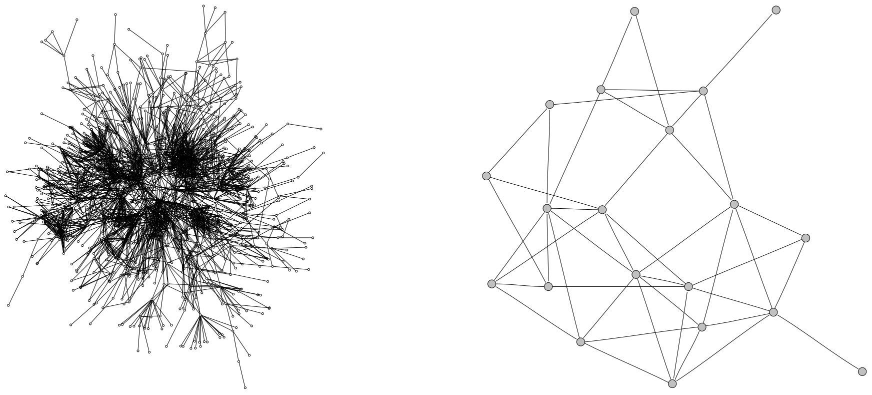

Figure 7: The AS2914 graph before compression (left) and after compression using the Path weight function with the KeepOne algorithm. In the compressed graph, important vertices are shown shaded.

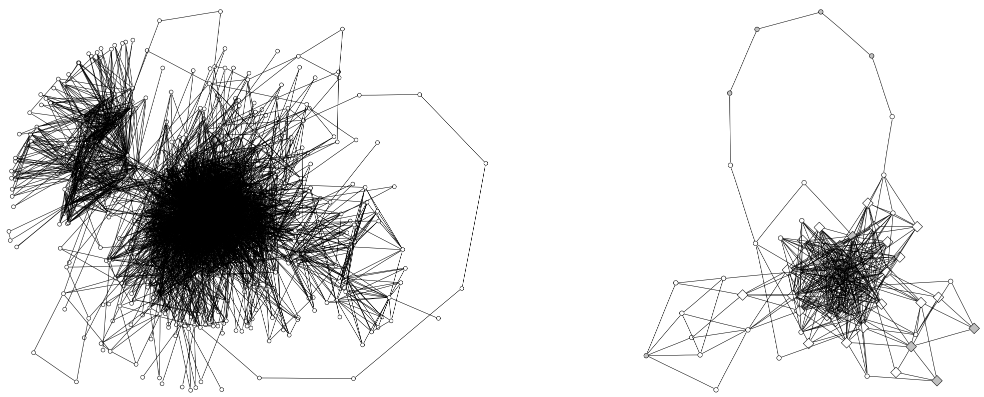

Figure 8: The AS3356 graph before compression (left) and after compression using Redundant Vertex Elimination followed by the Beta weight function with the KeepAll algorithm. In the compressed graph, important vertices are shown shaded.

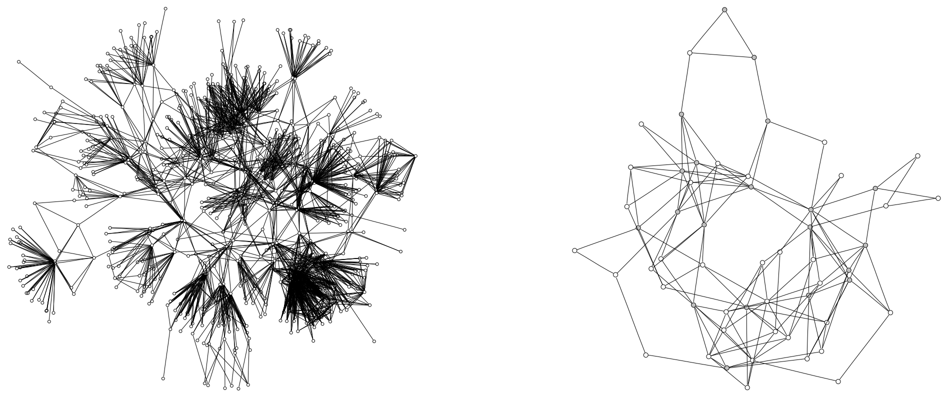

Figure 9: The AS7018 graph before compression (left) and after compression using the Path weight function with the KeepAll algorithm. In the compressed graph, important vertices are shown shaded.

	ERDOS				INET				AS1239						AS2914						AS3356						AS7018					
	V	%	E	%	V	%	E	%	V	%	BB	%	E	%	V	%	BB	%	E	%	V	%	BB	%	E	%	V	%	BB	%	E	%
Original	2937	100	4989	100	3076	100	4859	100	604	100	362	100	2268	100	960	100	453	100	2821	100	624	100	431	100	5298	100	631	100	385	100	2078	100
RVE	2937	100	4989	100	3074	100	4836	100	553	92	311	86	1605	71	909	95	406	90	2212	78	491	79	303	70	3831	72	576	91	332	86	1337	64
Deg/One	56	2	55	1	20	1	34	1	25	4	24	7	35	2	23	2	22	5	30	1	20	3	20	5	55	1	38	6	34	9	56	3
Beta/One	76	3	75	2	32	1	35	1	36	6	35	10	44	2	53	6	41	9	69	2	26	4	26	6	49	1	44	7	40	10	61	3
Path/One	35	1	34	1	20	1	37	1	20	3	20	6	31	1	20	2	20	4	46	2	20	3	20	5	56	1	20	3	20	5	34	2
Deg/All	409	14	574	12	107	3	238	5	107	18	98	27	390	17	88	9	85	19	362	13	82	13	79	18	716	14	90	14	82	21	261	13
Beta/All	522	18	731	15	84	3	151	3	134	22	126	35	444	20	136	14	121	27	427	15	96	15	93	22	733	14	102	16	93	24	306	15
Path/All	280	10	384	8	100	3	225	5	64	11	61	17	182	8	65	7	65	14	241	9	70	11	68	16	449	8	45	7	43	11	108	5
RVE+Deg/One	55	2	54	1	20	1	35	1	21	3	21	6	51	2	22	2	20	4	36	1	20	3	20	5	135	3	34	5	32	8	53	3
RVE+Beta/All	505	17	709	14	89	3	162	3	104	17	95	26	315	14	118	12	100	22	329	12	77	12	72	17	643	12	100	16	89	23	259	12
RVE+Path/One	33	1	32	1	20	1	35	1	20	3	20	6	46	2	21	2	21	5	50	2	20	3	17	4	87	2	21	3	21	5	35	2

Table 1: A summary of the performance of our compression schemes, showing the number of vertices, backbone vertices, and edges retained by each.

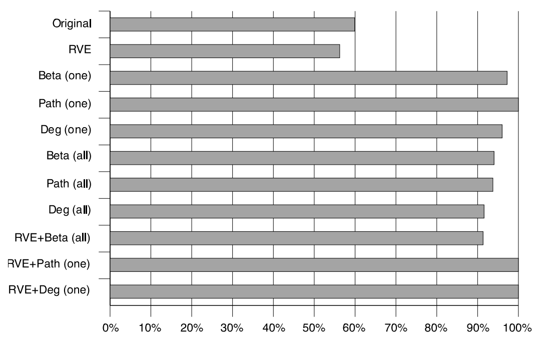

Figure 10: The fraction of backbone vertices in the AS1239 graph under different compression schemes.

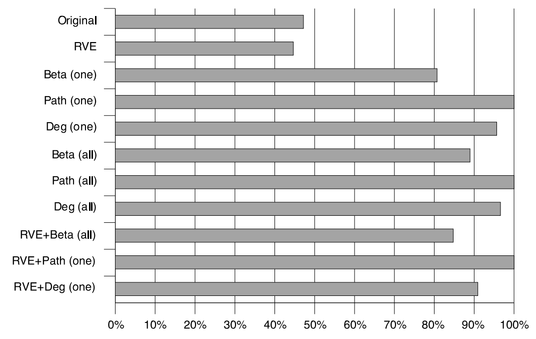

Figure 11: The fraction of backbone vertices in the AS2914 graph under different compression schemes.

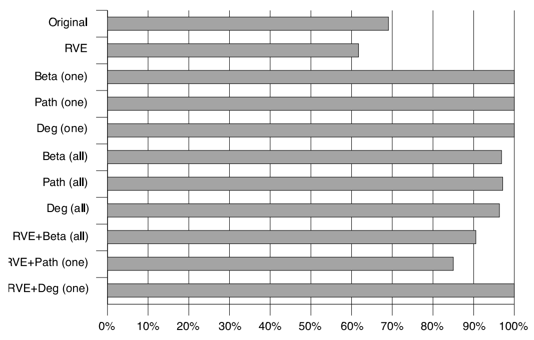

Figure 12: The fraction of backbone vertices in the AS3356 graph under different compression schemes.

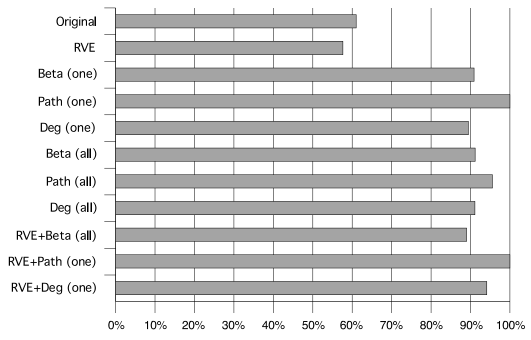

Figure 13: The fraction of backbone vertices in the AS7018 graph under different compression schemes.

"backbone" label in Rocketfuel dataset, from which the **AS1239**, **AS2914**, **AS3358**, and **AS7018** graphs are derived, with the assumption that backbone routers are intrinsically more important than other types. Figures 10–13 show the fraction of backbone vertices in the original graph and the graph after applying our compression schemes. In all cases, the compression schemes were able to identify the backbone routers, in some cases producing a graph composed exclusively of backbone routers. (Note that a decrease in the backbone router ratio after applying Redundant Vertex Elimination indicates that disproportionately more backbone vertices were merged than ordinary vertices.) In most cases, the Path weight algorithm (either alone or in conjunction with RVE) produced a graph which consists exclusively of backbone routers. This suggests that backbone routers are on many shortest paths between routers, as to be expected! It is interesting to note that the Path weight algorithm employs a measure which tends to compress the graph more than a degree-based measure. Not only does this measure remove more extraneous information in the graph but it also retains more important information.

4.4 Vertex and Edge Compression

Because we fix the size of K_1 at 20 throughout these experiments, it is worthwhile investigating how many vertices each compression scheme added back (in addition to the important ones) to reconnect the graph. This number is the size of K_2 and reflects the connectivity of the important vertices. Figure 14 shows for each network and for each compression scheme the number of additional vertices, expressed as a percentage of the total number of vertices. Because RVE compressed each network poorly, we removed it from the data. **INET** seems to have a different connectivity structure among its important vertices from the **AS** and **ERDOS** graphs. Considerably fewer vertices were added to **INET** in order to reconnect the important ones. For the real world networks, the Path algorithm added the fewest vertices which is not surprising as it tends to favor vertices that lie on a shortest path and, as such, generates important vertices that are already likely connected.

Although each compression scheme operates only on vertices and not on edges, each algorithm does change the number of edges in the graph. Figure 15 shows, for each compression scheme and for each network, the percentage of total edges retained. This percentage is a reflection of the degree distribution of the vertices kept in the compressed graph and it shows what percentage of the edges are incident to important vertices. We observe that the overall structure of this measure is quite similar to the percentage of additional vertices each scheme requires to reconnect the graph. Not surprisingly, the more vertices added, the more edges added. What is more interesting is the small relative changes. For example, for **AS1239**, the percentage of additional vertices for the two schemes Deg (all) and RVE+Beta (all) are about the same, 14%. The percentage of edges retained, however, by the two schemes are different, 17% for Deg (all) and 14% for RVE+Beta (all). Deg (all) classifies as important vertices those with a higher degree than RVE+Beta (all)'s important vertices.

4.5 KeepOne versus KeepAll

In Section 3.1.3 we motivated the KeepAll algorithm by arguing that the KeepOne algorithm does not preserve the distances between important vertices. Figure 16 shows the most severe distance distortion, which occurred using Path weight on the **ERDOS** graph. In practice, the KeepOne did not differ significantly from the KeepAll algorithm: both generally shifted the distribution but kept its overall shape. As an example, figure 17 shows Path weight on the **AS1239** graph.

4.6 Choosing Parameters

Throughout the experiments, we fixed the size of K_1 to be 20. One may ask whether this is an optimal choice, or whether the graph has a "natural" number of important vertices. Figure 18 shows the fraction of vertices retained as a function of the Beta weight threshold in the range [0.5, 1]. Response to the threshold is essentially smooth, with jumps at small-integer fractions (e.g., 1/2, 2/3, etc.).

Figure 19 shows the complementary cumulative distribution of the number of common neighbors between pairs of vertices. The tail suggests that there are vertices with very large sets of common neighbors. Figure 20 shows the complementary cumulative density of a slightly different similarity measure, which is the ratio of the number of common neighbors to the total number of neighbors of a vertex. While this measure is interesting, it does not seem as useful for clustering redundant vertices.

5. CONCLUSION

The emergence of large network connectivity datasets has

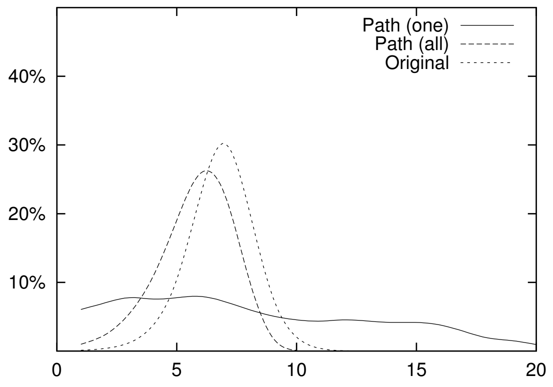

Figure 16: The distance distribution on all distinct vertex pairs in the ERDOS graph using Path weight with the KeepOne algorithm and with the KeepAll algorithm shown for comparison.

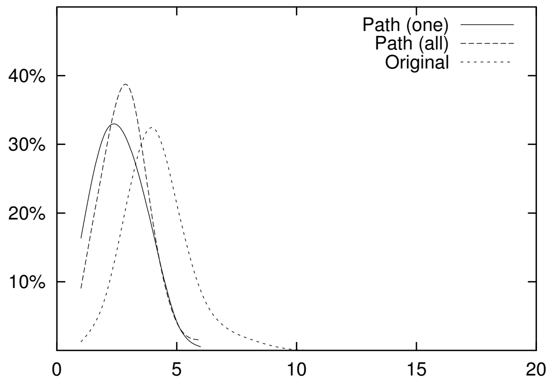

Figure 17: The distance distribution on all distinct vertex pairs in the AS1239 graph using Path weight with the KeepOne algorithm and with the KeepAll algorithm shown for comparison.

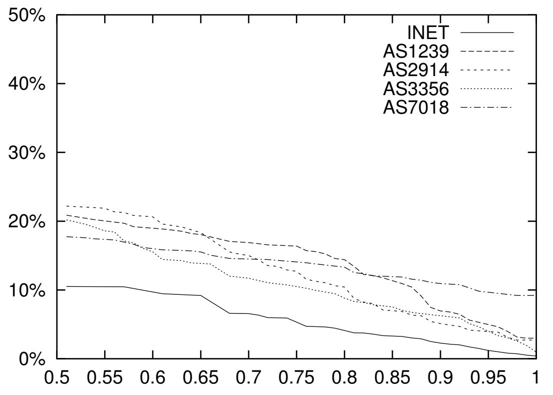

Figure 18: The fraction of vertices retained as a function of the Beta weight threshold.

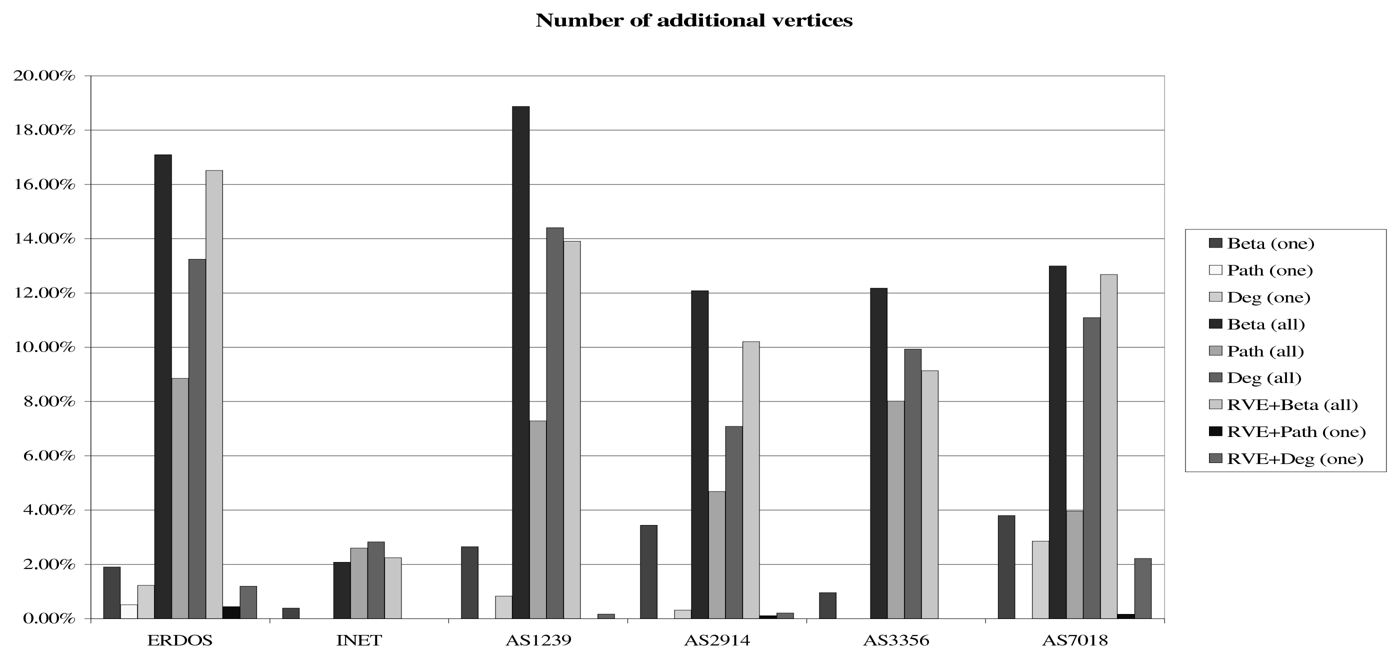

Figure 14: The percentage of total vertices that each compression scheme adds to the set of important vertices in each network in order to reconnect the compressed graph.

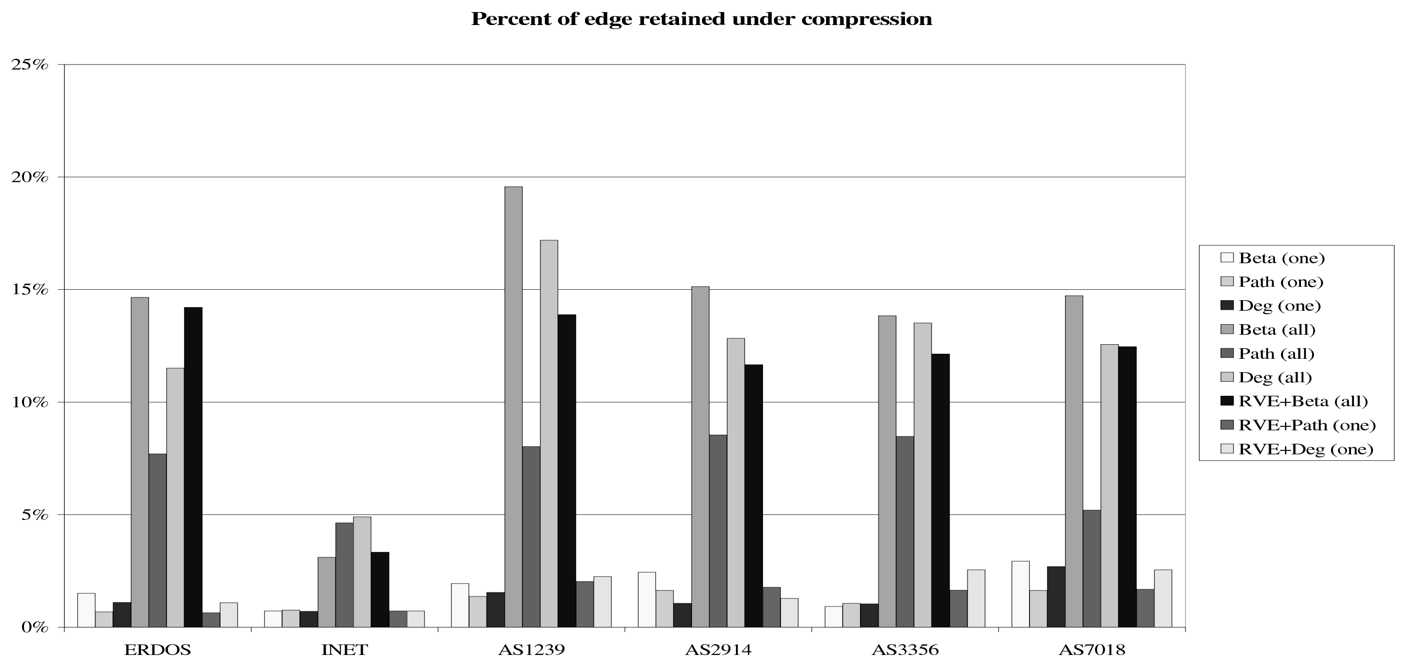

Figure 15: The percent of total edges retained under each compression scheme for each network.

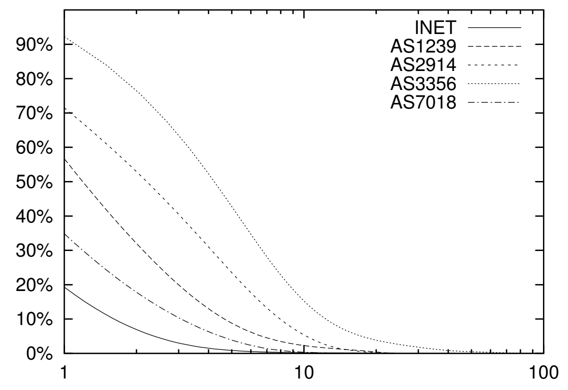

Figure 19: Complementary cumulative density function of the number of common neighbors between pairs of vertices.

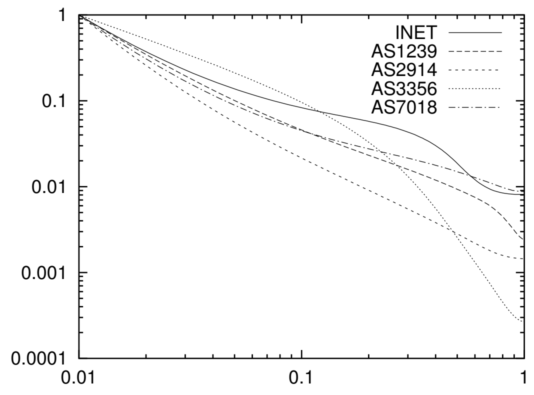

Figure 20: Complementary cumulative density function of the ratio of common neighbors to neighbors between pairs of vertices.

led us to consider the problem of condensing or summarizing the information presented in a network graph. We proposed three semantically-motivated importance measures on vertices in a network graph: Degree, Beta, and Path. All three were able to identify backbone vertices in our real-world network graphs, although we found that Path weight favored smaller vertex neighborhoods than Degree or Beta weight, which included more peripheral nodes. In practice, it may be necessary to try several schemes with varying parameters.

Having identified important vertices, we described two ways of producing a compressed graph based on these. We found that although our KeepOne algorithm produced cleaner, more sparse graphs, it was too conservative compared with KeepAll, which preserved more of the "context" of the important vertices.

We also described some similarity measures that can be used to merge similar vertices into one. Of these, Redundant Vertex Elimination did an excellent job of trimming down the graph while retaining its structure.

We expect our techniques will find application in other domains where the underlying graph has semantics similar to communication networks. However even in the domain of IP networks, as a first step in the direction of summarizing network graphs, our work leaves a number of directions open for exploration, most notably in finding new importance measures in such networks. Also, the KeepOne and KeepAll algorithms suggest that there is a more judicious way to connect important vertices without omitting or keeping too many vertices. We also did not consider the efficiency of our algorithms. Both the Path weight function and the KeepOne and KeepAll algorithms required computing all-pairs shortest paths, which may be too costly for large datasets.

6. REFERENCES

[1] M. Adler and M. Mitzenmacher. Towards compressing web graphs. In *Proceedings of the IEEE Data Compression Conference*, pages 203–212, 2001.

[2] AT&T Labs–Research. Graphviz. `http://www.research.att.com/sw/tools/graphviz/`.

[3] P. Erdös and A. Renyi. On the strength of connectedness of random graphs. *Acta Math. Acad. Sci. Hungar.*, 12:261–267, 1961.

[4] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proc. of ACM SIGCOMM '99*, pages 251–262, 1999.

[5] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. *Journal of Computer And System Sciences*, 51:261–272, 1995.

[6] A. Gilbert and K. Levchenko. Arc. `http://www.cs.ucsd.edu/ klevchen/arc/`.

[7] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46:604–632, 1999.

[8] R. Kumar, P. Raghavan, S. Rajagopalan, and A. S. Tomkins. Extracting large-scale knowledge bases from the web. In *Proceedings of the 25th VLDB Conference*, 1999.

[9] R. Lempel and S. Moran. The stochastic approach for link-structure analysis (salsa) and the tkc effect. In *Proceedings of the 9th International World Wide Web Conference*, 2000.

[10] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[11] N. Spring, R. Mahajan, and D. Wetherall. Measuring isp topologies with rocketfuel. In *ACM SIGCOMM '02 Proceedings*, pages 133–145, August 2002.

[12] T. Suel and J. Yuan. Compressing the graph structure of the web. In *Proceedings of the IEEE Data Compression Conference*, pages 213–222, 2001.

[13] H. Tangmunarunkit, R. Govindan, S. Shenker, and D. Estrin. The impact of routing policy on internet paths. In *Proc. of IEEE INFOCOM '01*, volume 2, pages 736–742, 2001.

[14] S. White and P. Smyth. Algorithms for estimating relative importance in networks. In *ACM SIGKDD '02 Proceedings*, pages 266–275, August 2003.

[15] J. Winick and S. Jamin. Inet-3.0: Internet topology generator. Technical Report CSE-TR-456-02, University of Michigan, 2002.