exeley
CONNECTIONS
Article | DOI: 10.21307/connections-2019-009
Issue 1 | Vol. 39

Hairball Buster: A Graph Triage Method for Viewing and Comparing Graphs

Patrick Allen,* Mark Matties and Elisha Peterson

Johns Hopkins University Applied Physics Laboratory, Laurel, MD.

*E-mail: Patrick.allen@jhuapl.edu

Abstract

Hairball buster (HB) (also called node-neighbor centrality or NNC) is an approach to graph analytic triage that uses simple calculations and visualization to quickly understand and compare graphs. Rather than displaying highly interconnected graphs as 'hairballs' that are difficult to understand, HB provides a simple standard visual representation of a graph and its metrics, combining a monotonically decreasing curve of node metrics with indicators of each node's neighbors' metrics. The HB visual is canonical, in the sense that it provides a standard output for each node-link graph. It helps analysts quickly identify areas for further investigation, and also allows for easy comparison between graphs of different data sets. The calculations required for creating an HB display is order *M* plus *N* log *N*, where *N* is the number of nodes and *M* is the number of edges. This paper includes examples of the HB approach applied to four real-world data sets. It also compares HB to similar visual approaches such as degree histograms, adjacency matrices, blockmodeling, and force-based layout techniques. HB presents greater information density than other algorithms at lower or equal calculation cost, efficiently presenting information in a single display that is not available in any other single display.

Keywords
Graph analytic triage, Node-neighbor centrality, Standard canonical form for graphs, Comparing graphs.

Purpose and overview

The purpose of this paper is to describe a new method for analyzing relationships among nodes in a graph using a canonical representation that also enables comparison between different graphs. The approach is called 'node-neighbor centrality' (NNC), or more colloquially, 'hairball buster' (HB).

HB computes a centrality measure (such as node degree) for a node and its neighbors, and presents this computation in an efficient, standardized visual form that scales to very large graphs. Using the visual depiction of the measure, an analyst can quickly answer questions such as whether the graph is (generally) from a social network or a random graph. Additionally, the depiction retains information about relationships, so an analyst can also quickly determine whether high-degree nodes are connected to each other directly or through a mutually adjacent node, such as in a bipartite graph.

This paper presents examples of the HB approach addressing five types of analytic questions using four real-world data sets. HB is a canonical approach using node degrees that allows for comparison of different graphs, while extensions of HB include the display of selected graph attributes such as link weights. The use of alternative measures of centrality is also presented. The approach is compared and contrasted with other common graph algorithms. The paper concludes with the limitations of the HB approach and future planned features and applications.

The HB python code is available at https://github.com/PatAllen496/Hairball-Buster.

The need

The most commonly used graph visualization techniques include node-link visualizations that embed a graph's nodes and links in two-dimensional space, and adjacency matrix visualizations that show the entire space of possible connections in a large matrix. Each of these techniques has a number of advantages and disadvantages (Ghoniem et al., 2005).

A particularly challenging case is graphs that have so many elements and interconnected features that it becomes difficult to determine which nodes and links are most 'important.' When using standard graph visualization algorithms such as force-directed (Kamada and Kawai, 1989; Fruchterman and Reingold, 1991) or dimensional reduction, the usual starting point is a depiction of all nodes and links. For many kinds of graphs, especially those with high connectivity, this results in a 'hairball' as shown in Figure 1, which shows a link between every pair of jazz musicians that have performed together (Gleiser and Danon, 2003).

The purpose of graph visualization is to help an analyst understand features of the graph or a particular node, using visual queries (Freeman, 2000; Peterson, 2011; Ware, 2010). However, in this typical 'hairball,' it is difficult to determine at a glance the nodes with the highest degree, the distribution of nodes by degree, and whether the highest degree nodes are directly connected to each other. An analyst needs to apply a range of other algorithms to further dissect the graph, sometimes requiring multiple iterations, to determine how the various nodes relate to each other.

In addition, there are a number of additional challenges that arise when visualizing graphs that change over time (Bender-deMoll and McFarland, 2006; Peterson, 2011). There is a tendency in force-directed visualization for nodes and links to reposition themselves, every time new nodes or links are added or removed. This makes it difficult not only to identify key nodes, but to track them over time. A number of approaches have been suggested, but they do not fully address the issue and are computationally expensive (Bender-deMoll and McFarland, 2006; Brandes and Corman, 2003; Moody et al., 2005; Peterson, 2011; Zweig, 2016).

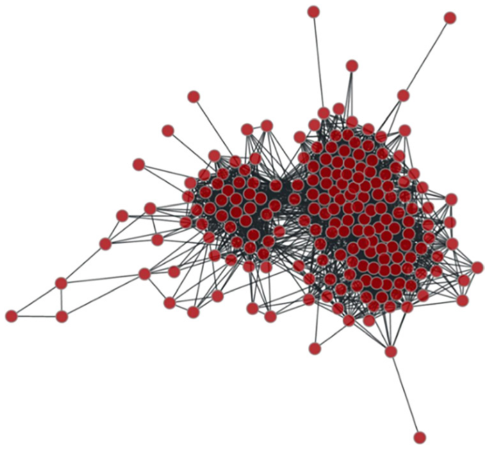


Figure 1: Sample 'Hairball' showing jazz players that performed with each other.

An alternative approach is the backbone layout, which attempts to directly resolve difficulties in visualizing particularly dense portions of a force-directed layout (Lehmann and Kottler, 2007, Nocaj et al., 2014, 2015). Figure 2 shows the same data set using Visone's Quadrilateral Simmelian backbone layout. While the big hairball of Figure 1 has been broken up into four clusters, one large hairball has turned into several smaller hairballs. One still cannot answer many questions of interest to a data analyst, e.g. which nodes have the highest degree or how nodes of high degree are connected to each other.

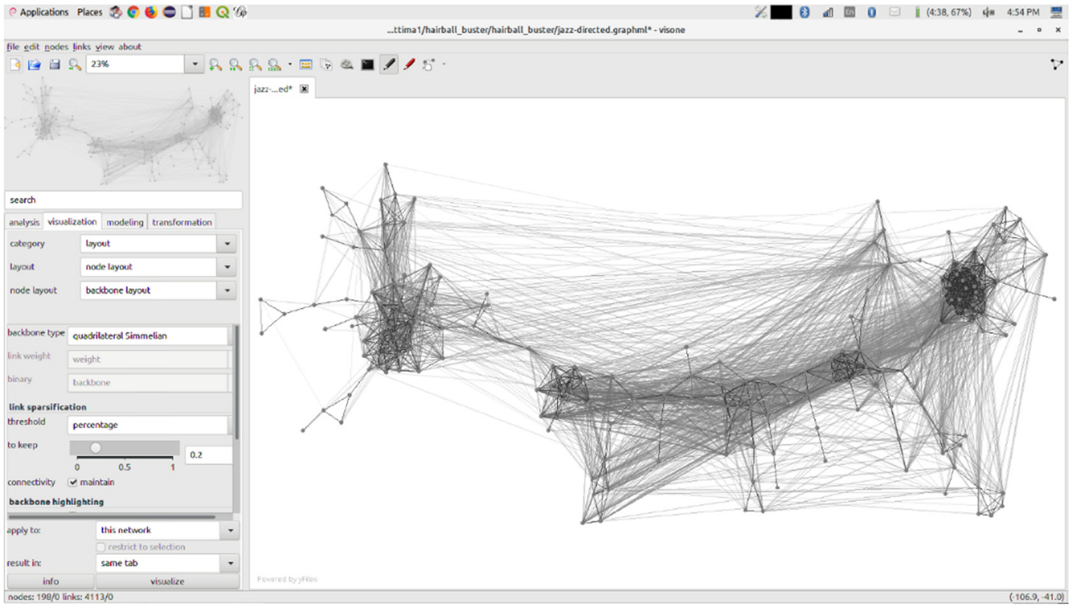
There is also an inherent performance cost when generating force-directed graph displays, most of which are at a minimum order N^2, where N is the number of nodes (Fruchterman and Reingold, 1991).

Because of the challenges in visualizing node-link diagrams in these cases, alternatives, such as an adjacency matrix visualization, are often proposed (Sheny and Maz, 2007). Adjacency matrices also can be a very effective way to visualize clusters, so they are often used when studying communities, sometimes referred to as clustering or blockmodeling (Wasserman and Faust, 1994; White et al., 1976). In one study, the authors found that the adjacency matrix is almost always superior for a certain set of tasks to the node-link diagram (Ghoniem et al., 2005). However, the authors did not include any graphs with more than 100 nodes in their study, and this is the principle drawback of adjacency matrix visualizations: they do not scale well to graphs with thousands or millions of nodes.

HB approach

HB is a new way of looking at graph data that scales effectively to large, dense graphs. The approach is simple to calculate and plot, and provides an easy way to identify by inspection the most connected nodes and most important links in the graph.

Assume there is a graph with N nodes and M links. The degree of each node is the number of links connected to that node. (Throughout this paper, we will use degree as our primary measure of centrality,



category: layout
layout: node layout
node layout: backbone layout
backbone type: quadrilateral Simmelian
link weight: weight
binary: backbone
link sparsification
threshold: percentage
to keep: 0.2
connectivity: ✓ maintain
backbone highlighting
apply to: this network
restrict to selection
result in: same tab
nodes: 198/0 links: 4113/0

Figure 2: Visone backbone layout of jazz player data set.

although HB representations of other centrality measures are presented later.) There are six steps to creating the HB plot as follows:

1. Calculate the centrality (degree) of each node (which requires *M* calculations).
2. Sort the nodes by degree, assigning ranks from 1 (the highest degree node) to *N* (which requires *N* log *N* calculations).
3. Plot (in one color) the monotonically decreasing curve of degree (vertical axis) vs. node rank (horizontal axis). (There will be *N* points on this curve, one for each node.) Call this 'the curve' and the nodes on it 'curve nodes.'
4. Calculate the neighbors of each node and place each neighbor on a list associated with each ranked node. (The placement of the neighbor on the list of neighbors for each node is accomplished during the initial *M* calculations in Step 1.)
5. Store the degree of the neighbor with the neighbor node. (This step uses an index for each node so that the degree of the neighbor is an indexed look-up.)
6. For each node, plot (in another color) the value of each of its neighbor's degrees on the vertical line at the same horizontal position as that node, so that each of its neighbors will be represented above or below that node's position on the curve. Call these the 'neighbor nodes.'

Optional calculations, such as ensuring canonicalization and parallelization, and display options for log–log, semi–log, inverse, and same degree offsets, are presented in Section 'optional steps of the HB algorithm.'

The computational efficiency of HB is on the order of *M*+*N* Log *N*. In contrast, traditional graph displays that look like the hairball shown in Figure 1 require order N^2 (Fruchterman and Reingold, 1991). In addition, some algorithms only sample the graph data set, while the HB approach deals with the whole data set in one pass (Squartini et al., 2015). See Section 'HB measures of performance' for further details.

In addition to computational efficiency, HB uses visual space more efficiently than an adjacency matrix, making it suitable for graphs of any size. It supports many of the same visual queries as an adjacency matrix, with the additional advantage that it can highlight not just a node's neighbors or clusters, but also how a node's centrality measure relates to those of its neighbors. In the remainder of this paper, these advantages are described in more detail by analyzing several real-world sample graphs.

First application: quickly identifying key nodes and relationships in a graph

This section uses the jazz data set to illustrate how HB can answer common analytic questions about

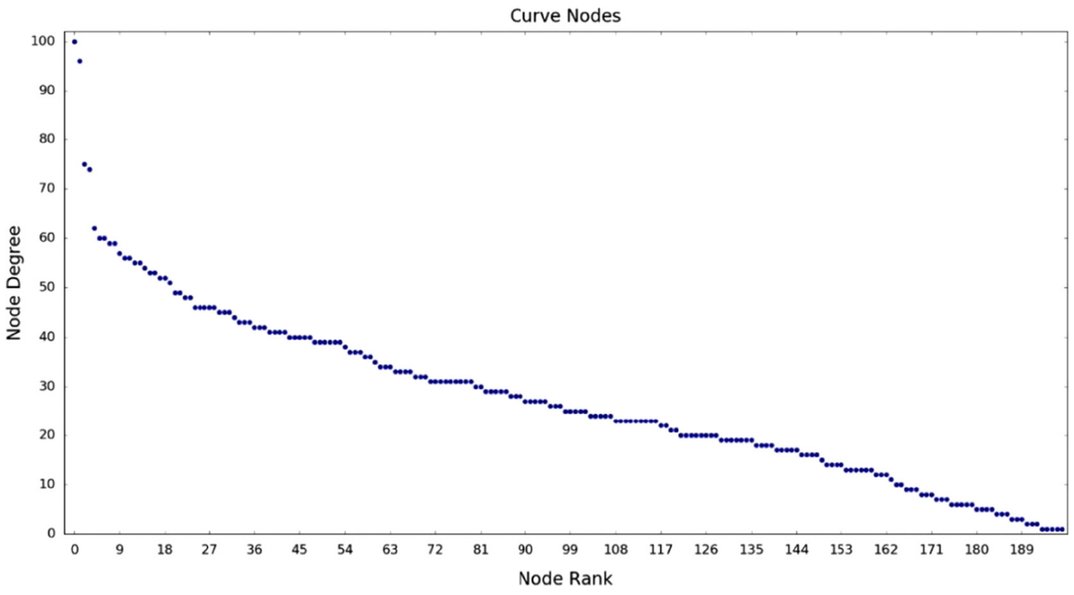


Figure 3: Sample HB curve for jazz players that performed with each other.

which nodes have the highest degree and how these are connected to other types of nodes.

Figure 3 depicts the degree curve (Step 3, above). Note that there are four very high-degree nodes in the upper-left-hand corner. The rest of the nodes follow a fairly linear path from upper left to lower right. (This pattern is typical for social networks.)

Figure 4 displays the neighbors of the curve nodes (Steps 4, 5, and 6 above). Each red dot represents one or more links on a traditional graph display. The dot's vertical position indicates the node at one end of the link and its horizontal position indicates the node at the other end. For example, the red dot at coordinate (2,100) is the link between

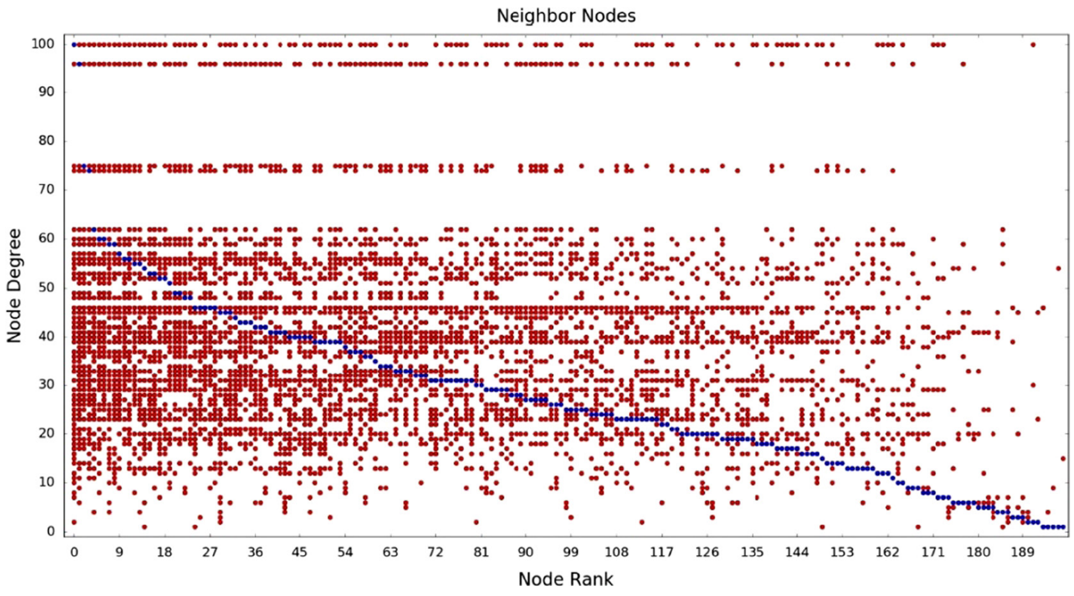


Figure 4: Neighbors plot for jazz players that performed with each other.

the first and second nodes on the curve (first two blue dots).

This curve is not quite the same as a histogram or node-degree distribution, where one dot represents many nodes of the same degree. As with node-degree curves, the shape can be useful for comparing different graphs. However, HB displays one dot for each node, since the horizontal axis is the degree rank of the node. This is an important distinction, because HB retains connectivity information about individual nodes that other techniques do not and can therefore answer a much broader class of questions. It can also address additional questions that an adjacency matrix cannot, as will be summarized later.

Unlike the backbone layout display, the HB chart clearly shows which nodes have highest degree, how much higher their degree is than other nodes, whether the highest degree nodes are directly or indirectly connected to other high-degree nodes, and how high-degree nodes tend to connect to low-degree nodes. This is summarized in Figure 5.

For instance, using the HB visual for jazz players in Figure 4, the top 8 nodes are all clearly connected to each other (forming a fully connected subgraph), since there is a red dot on the same row and column as the three blue dots representing the eight highest degree nodes on the curve. For example, the highest degree node (rank 1, degree 100) is connected to the second highest ranked node (rank 2, degree 96), indicated by the red dots plotted at rank 1/degree 96 and rank 2/degree 100. This pattern continues with the remainder of the top 8 nodes. This specific kind of connectivity information cannot be determined from a backbone or a histogram display.

Second, the highest degree jazz players rarely performed with the lowest-degree jazz players, as shown by the gaps in the far side of the upper right quadrant, which increase in frequency and length as the rank increases to the right. This means that the number of lower-ranked musicians with whom the most connected musician performed is small.

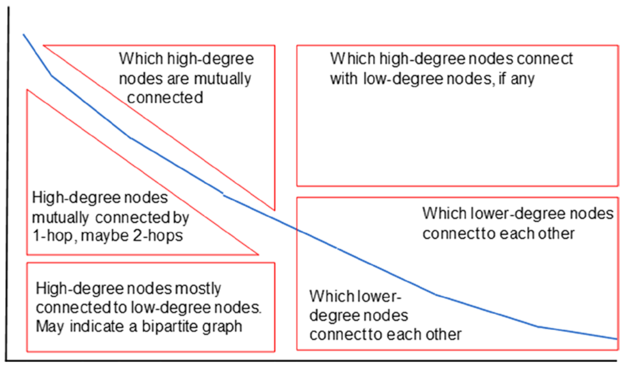

Which high-degree nodes are mutually connected
Which high-degree nodes connect with low-degree nodes, if any
High-degree nodes mutually connected by 1-hop, maybe 2-hops
Which lower-degree nodes connect to each other
High-degree nodes mostly connected to low-degree nodes. May indicate a bipartite graph
Which lower-degree nodes connect to each other

Figure 5: Questions addressed by location of neighbor nodes.

Third, there are few dots near the bottom of the chart. This shows that jazz players who have performed with many others have tended to perform with other jazz players who have also performed with many others, and not with those who have performed with few.

In general, the upper-left 'quadrant' or section of Figure 5 shows which high-degree nodes are mutually connected to other high-degree nodes. If some of the highest degree nodes are not connected with each other, this can indicate that there are different clusters of nodes around some of the high-degree nodes (an observation that can be made without running a clustering algorithm). Conversely, if the highest degree nodes are mostly directly connected to each other, this provides a different pattern around a core group to analyze further.

In the upper right and lower left quadrants, it is easy to see which high-degree nodes connect with lower-degree nodes and which do not. A high-degree node with many connections to one-degree nodes indicates a common star pattern on traditional graph displays. However, if one finds the highest degree nodes are connected to low-degree nodes rather than each other, then one may have a bipartite graph or other distinguishing feature.

The lower right quadrant shows which lower-degree nodes connect with each other. If this area is sparse or empty, then the lower-degree nodes are only connecting with the higher-degree nodes. This is indicative of a star-like shape for some of the high-degree nodes.

The visual can also be used to find nodes that are indirectly connected via an intermediate node. If two nodes A and B have a common neighbor C, then C will be depicted as a neighbor node on the same horizontal line above or below each of A and B.

Figure 6 shows how the HB chart can appear for a directed graph. In this example, we randomly assigned a direction to the Jazz player data set, where green indicates an 'in' link to the node in that row, while red indicates an 'out' link. The Jazz player data set consisted of undirected links, and this figure just shows how directed graphs would appear if the links were directed.

Second application: quickly identifying core groups or multiple groups

This section shows how HB can quickly determine whether there is a single core group or multiple core groups in a data set. This example uses Toaster

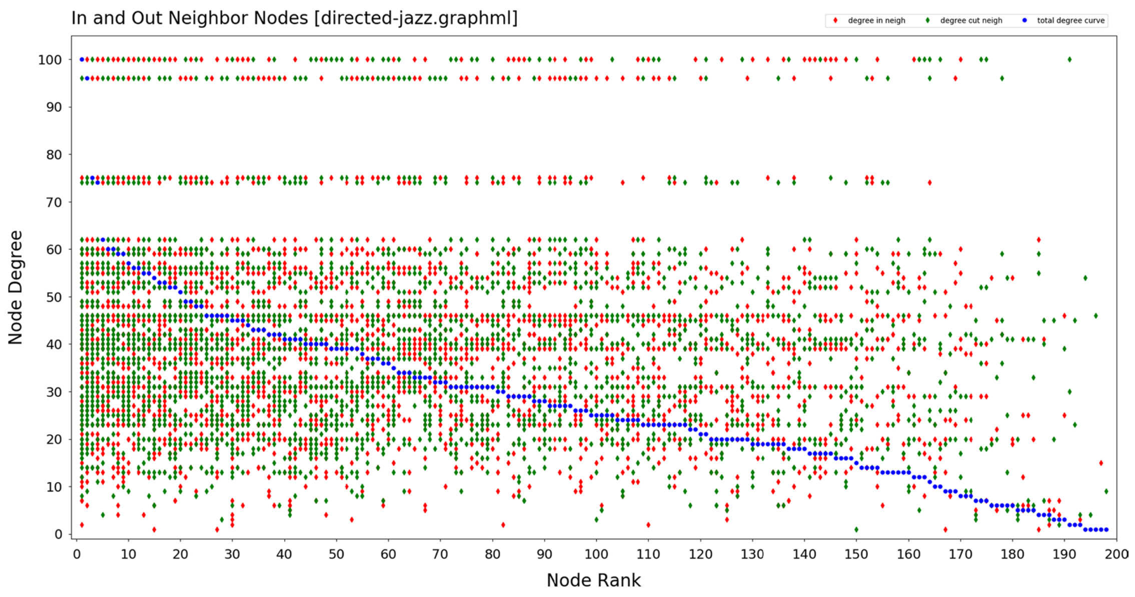


Figure 6: Sample directed neighbors plot for jazz player data set (Green = In, Red = Out).

(Toster dot ru), which is a Russian social media site for software support and help from a community of subject matter experts (SMEs). It is similar to the popular StackOverflow site, but the Toaster data set is smaller and provides a form of ground truth in terms of user-provided tags for purposes of comparison.

The Toaster data have a set of threaded discussions where a person posts a question, someone else posts an answer (usually an SME), and then others can comment on both the question and the answer. The data set at the time of download had over 30,000 nodes, 3,865,650 edges, and over 14,000 discussions. Initial work by others at APL examined how to find sub-communities within the larger community represented by the Toaster data set. See Figure 7 for a traditional force-directed visualization of the Toaster data set. This image definitely qualifies as a hairball! As shown in Figure 8, the backbone layout did not produce more informative results.

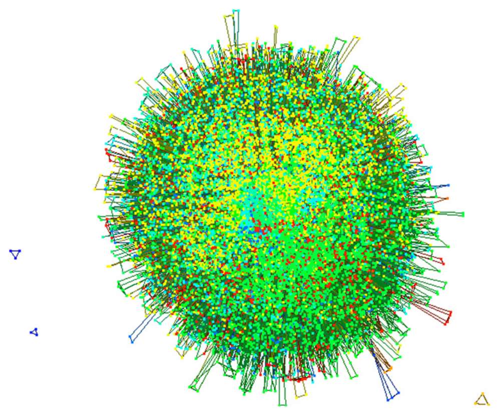


Figure 7: Force-directed representation of the Toaster data set.

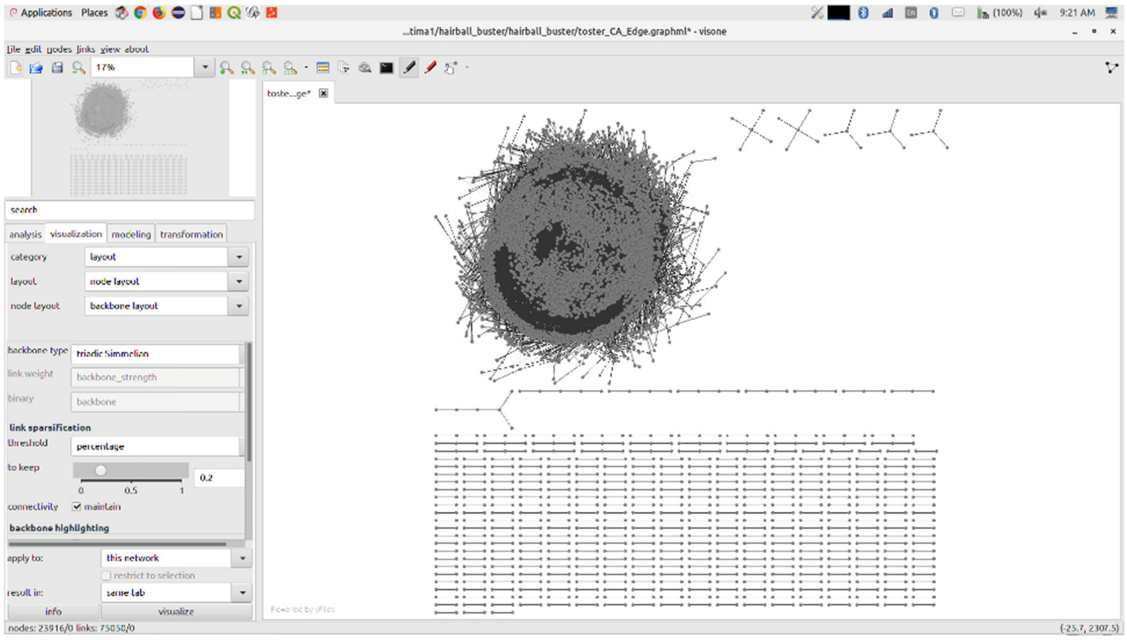
To apply the HB approach to this data set, we first removed duplications and focused entirely on whether any username in the data set communicated with any other username in the data set. The analytic question we are asking is ‘Who are the core members, and are there any large communities with unique core members?’

While the original data set had 30,000 nodes and almost 4 million edges, the de-duped data set had 23,916 nodes and 75,050 edges. Figure 9 shows the HB representation of the nearly 24,000 nodes.

Focusing on the highest-ranked nodes, the top 28 can be readily identified, while the remaining are difficult to visually distinguish. Each of the top-ranked nodes appears to connect to all the other high-ranked nodes, and the first obvious visual gaps occur at around 1,000 nodes. This indicates that the top-ranked nodes are either fully connected or very nearly fully connected.

While displaying the full data set provides the information described above for the first 28 nodes, it does not definitively indicate whether the highest degree nodes are fully connected, or whether they belong to separate clusters due to visual occlusion.

To address this limitation, it helps to view the ‘inverse’ of the neighbors – that is, to display the



category: layout
layout: node layout
node layout: backbone layout
backbone type: triadic Simmelian
link weight: backbone_strength
binary: backbone
link sparsification
threshold: percentage
to keep: 0.2
connectivity: ✓ maintain
backbone highlighting
apply to: this network
restrict to selection
result in: same tab
nodes: 23916/0 links: 75050/0
(-25.7, 2307.5)

Figure 8: Backbone layout representation of the Toaster data set.

missing links (the gaps), and to zoom in on the top nodes. (Zooming in on the display adds no additional computational penalty beyond rendering.) When there are no dots in the inverse, the graph is fully connected. Figure 10 shows this inverse display zoomed in on the first 250 nodes. It appears that the top 20 nodes are almost, but not quite, fully connected.

Figure 11 zooms in further on just the top 20 nodes, again showing the 'inverse' neighbors. It is clear that nodes 1 through 15 are fully connected and that nodes 1 through 20 are almost fully connected.

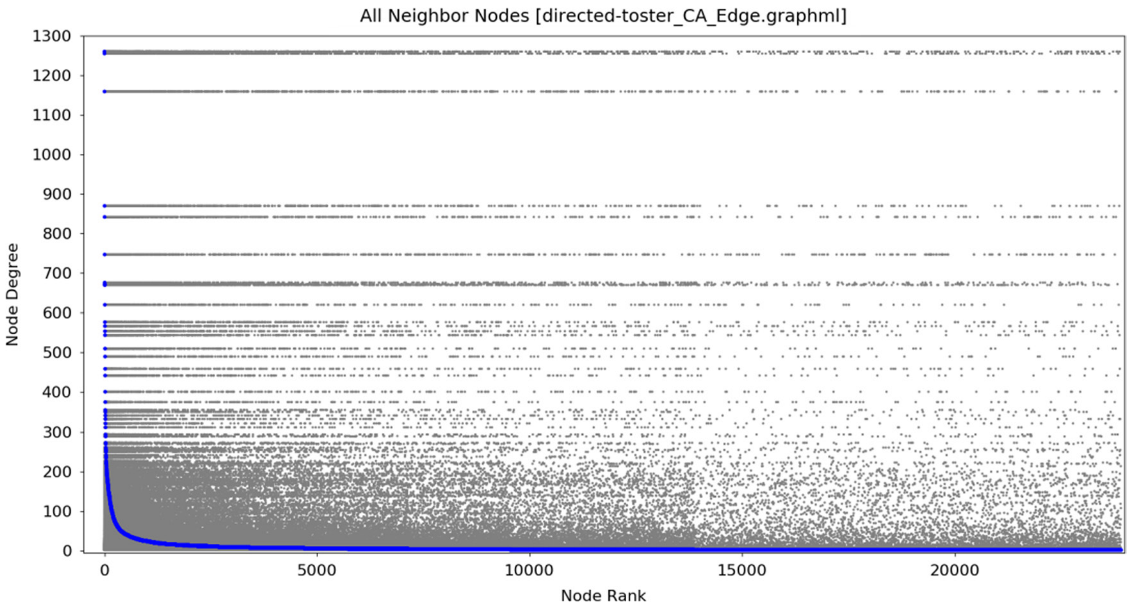


Figure 9: HB representation of the Toaster data set (directionality ignored).

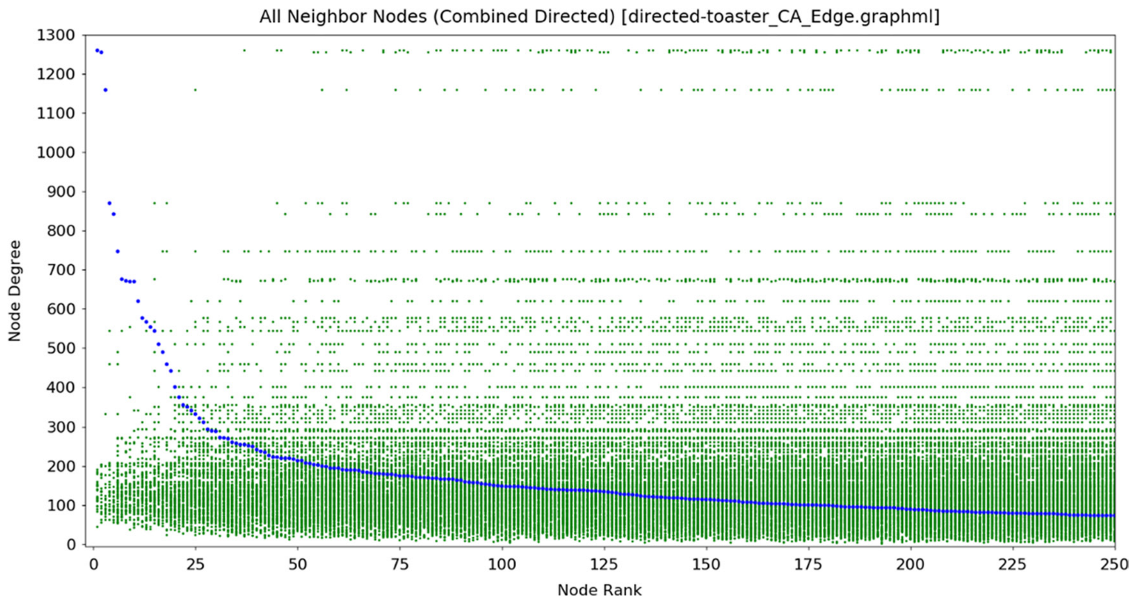


Figure 10: HB representation of the inverse of neighbor nodes (e.g. gaps).

Note that zooming in on the inverse neighbors was a simple way to gain a more definitive understanding of the graph while incurring virtually no additional computational cost.

In summary, using our triage approach based on HB, we can quickly see that the top-ranked 15 SMEs in the Toaster data set have all commented on, or been commented on, by each other, and the top 20 nearly so. This means that there is likely to be just one core group in the Toaster community all connected with each other.

In contrast, it takes more than one algorithm and manual steps to provide similar data. For example, we ran a histogram on the Toaster data set, which

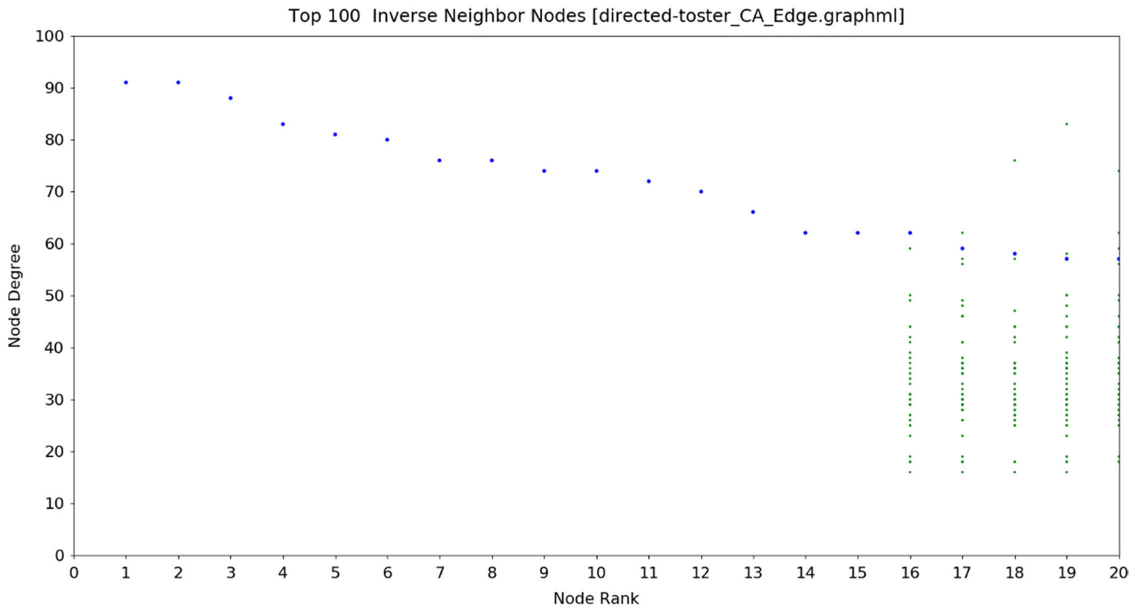


Figure 11: HB inverse representation of just the top 100 ranked nodes with each other in Toaster data set.

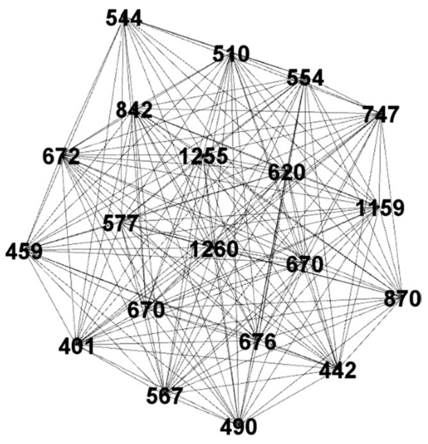


Figure 12: Force Atlas 2 on top 20 nodes in Toaster data set.

identified the top 25 to 30 nodes as having the highest degree. We then ran Gephi, ranking the nodes by degree and manually copied the top 20 nodes to visualize using Force Atlas 2. Figure 12 shows the results using degree as the node label. While the process took roughly 5 min, HB provided the results in 1 sec for the initial display and then for the inverse display. The Gephi example does show highly connected nodes, but does not conclusively show which are fully connected, and required greater time and calculation cost than HB.

While k-truss is not available in Gephi, it is useful in identifying clusters in data sets. However, when the top nodes are fully or nearly fully connected, the k-truss algorithm will not provide additional useful information about these nodes.

Third application: HB and temporal graphs

A significant benefit of the HB chart approach is that the canonical format allows multiple curves/graphs to be compared at once. As an example, we divided the Toaster data set into blocks of 3,500 connections representing approximate slices of the data set over time (since the initial data set was in roughly chronological order). We then compared the HB depictions of the first 3,500 node connections with the second and third blocks. Figures 13 to 15 show these three batches of nodes and links plotted with the same axis scales for ease of comparison.

In Figure 13, there is one node above degree 180, which is a much higher degree than any of the other nodes. The next highest node is around 110, followed by a couple at 70 and then a fairly smooth curve toward the lowest degrees. Figure 14 shows that in this next time period, the 110-degree node is the highest-ranked node and the 70-degree nodes are still present, but there is also an interesting bump in the curve around degree 20. Figure 15 also has a

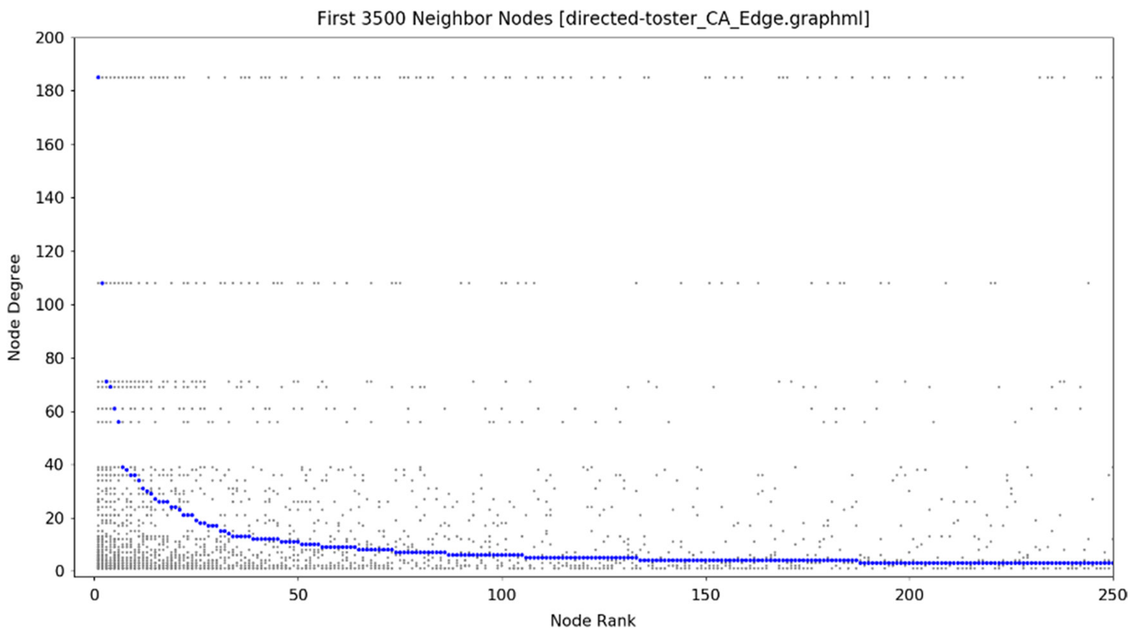


Figure 13: HB chart of first 3,500 connections in Toaster data set.

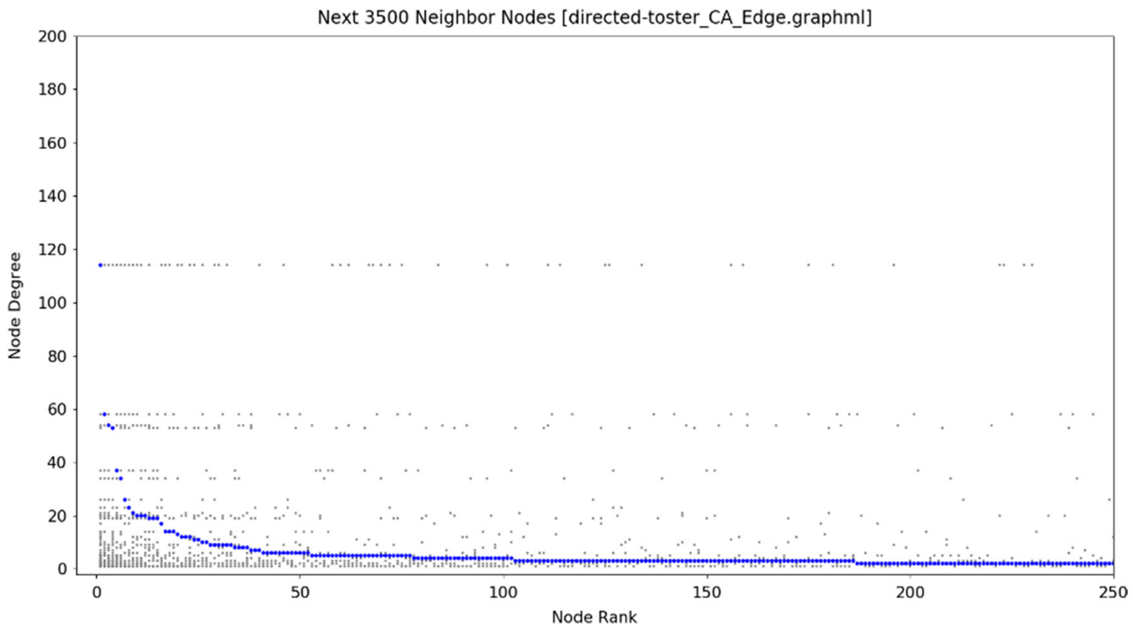


Figure 14: HB chart of second 3,500 connections in Toaster data set.

maximum degree node at 110, but also one at 90, 80, and 60. This third set of 3,500 connections also has a 'bump' in the curve around degree 20 that is similar to the bump in Figure 14.

In typical node-link visualizations, visualizing changing graphs compounds many of the issues associated with visualizing static graphs. In addition, there are new forms of 'visual noise': nodes/edges that are displayed or removed without warning, and nodes/edges that move rapidly from one time period to the next without warning (Peterson, 2011). Many approaches have been suggested to address these

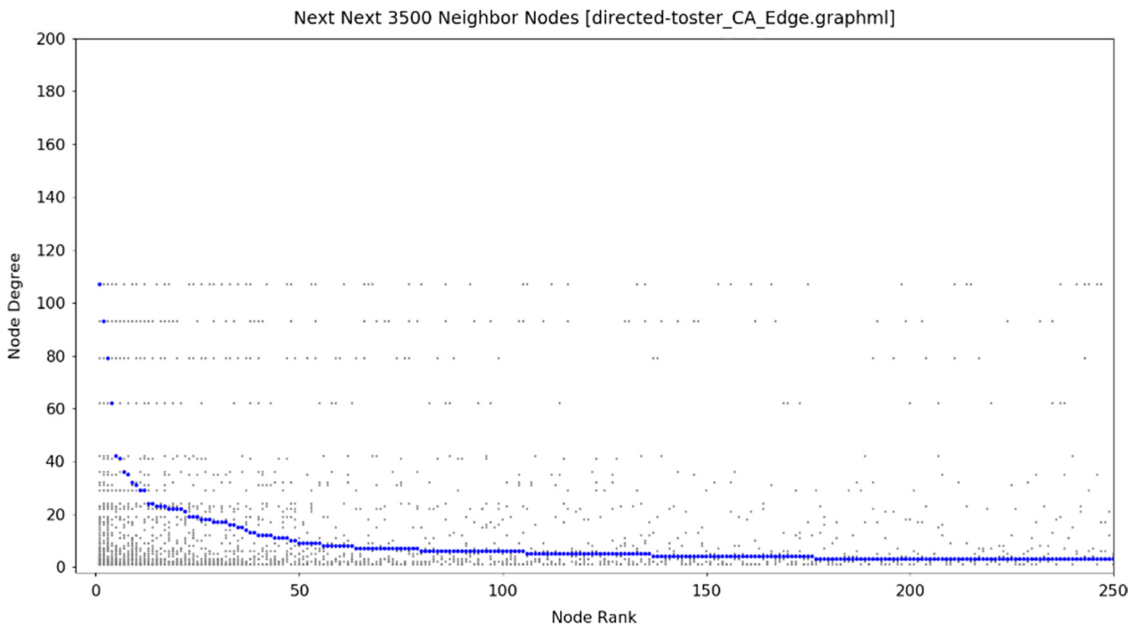


Figure 15: HB chart of third 3,500 connections in Toaster data set.

issues, but they are computationally expensive and only work well in limited cases (Bender-deMoll and McFarland, 2006; Brandes and Corman, 2003; Moody et al., 2005; Peterson, 2011; Zweig, 2016). In contrast, the HB approach is computationally inexpensive and high in information content.

HB does not address all of these issues, but allows the analyst to focus on how the centrality of a specific node changes over time, or how the distribution of centrality changes over time. For example, does the shape of the curve change over time? This is shown in Figures 13 to 15 for the Toaster data set. Does the rank of each node change over time? This can be added in a later version of the HB code. See Section 'future features and applications of HB' for such an approach.

Fourth application: identifying anomalous features

This section describes how HB can be used to quickly identify selected anomalous features in a data set associated with the highest degree nodes, using suspended Iranian Twitter™ accounts obtained from https://about.twitter.com/en_us/values/elections-integrity.html#data (Twitter™, 2018). This data set for user-id replies with no retweets between nodes included 228,626 nodes and 440,244 edges. Figure 16 was calculated in about 40 sec on a single-threaded laptop, and shows that one node dominated with over 260,000 replies. The next two highest nodes had around 90,000 replies and just under 30,000 replies, respectively. (Displaying over 200,000 nodes and 400,000 links would not be feasible in an adjacency matrix. See Section 'comparing HB to other algorithms' for comparisons to both the adjacency matrix and blockmodeling.) Figure 16 shows how much larger the degree of highest node is compared to all other nodes, as well as for the second and third degree nodes.

More importantly, this figure shows an interesting pattern in gaps in the reply pattern of these top 3 nodes, as well as the highest of the next lower-degree-ranked nodes. For example, the highest degree mode appears to connect with most of the rest of the nodes except around nodes ranked at 160K, while the second highest degree node has multiple gaps and the third highest degree node does not appear to connect with about half the nodes.

Zooming in on the left-hand side, Figure 17 shows the same data limited to the first 200 nodes. Note that the three top-ranked nodes connected with each other and many other nodes, but did not connect with the next 40 highest-ranked nodes except in one case. What is the reason for such an unusual pattern? The authors do not know for sure, but it may be that nodes 4 through 44 are bots run by a different team than those running the first three nodes. (Of the 4th through 44th nodes, the 4th node communicated with most of the other 43 nodes, but most of the rest of the 43 nodes did not communicate with each

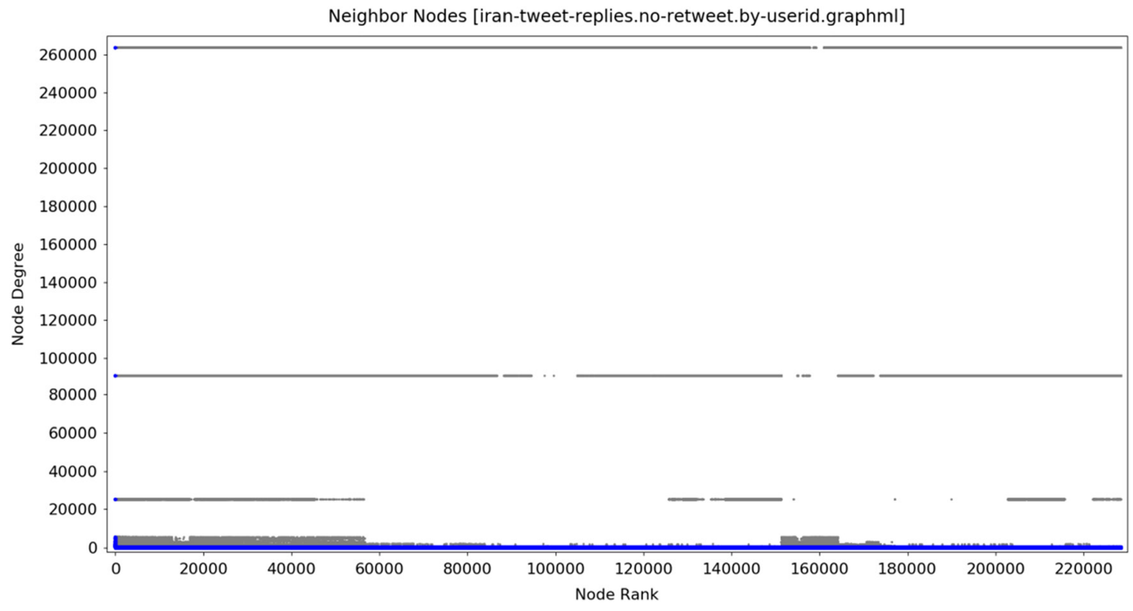


Figure 16: HB chart of suspended Iranian Twitter™ accounts, user-id replies, and no retweets.

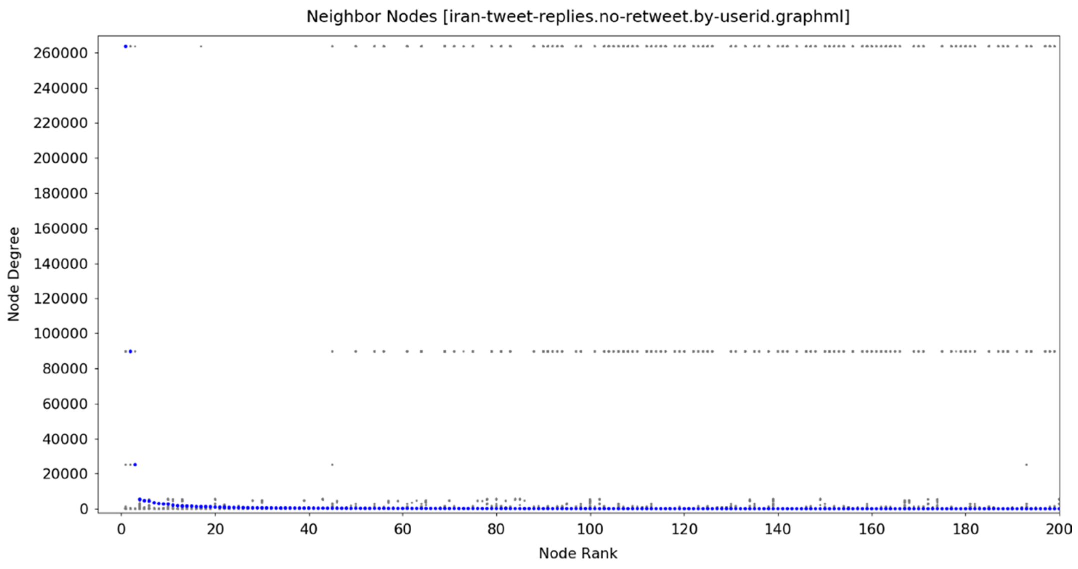


Figure 17: HB chart of suspended Iranian Twitter™ accounts, user-id replies, no retweets, first 200 nodes showing gaps among the top 3 and the next 40 nodes.

other.) In any case, HB is an efficient way to quickly identify areas of interest and further investigation into anomalous data without having to slowly whittle down a huge hairball display. This approach might also be useful in helping identify other Twitter™ bots in the future based on similar patterns.

Fifth application: quickly identify nodes connected by highest-value link weights

This section shows how HB can be used to identify key relationships in graphs with link weights. This example uses output data from a tool called CodeDNA™, a patented malware analysis tool developed at JHU/APL that provides a fast, reliable, automated means for recognizing related malware binaries and linking variants. It 'supports crowd-sourcing of information by providing a robust malware identifier (fingerprint) that is deterministic and repeatable for correlating reports, analyses, and other information about attackers, yet cannot be used to re-create the original malware' (Maughan and Carlsten, 2018). By generating DNA-like fingerprints from input files, and computing similarity between these fingerprints, CodeDNA™ can effectively identify clusters of related malware in very large data sets. Figure 18 shows some samples of clusters of malware previously produced by CodeDNA™.

For purposes of this paper, we obtained a data set based on Linux coreutils rather than real malware, and processed the data through CodeDNA™ software. Figure 19 shows the seven clusters produced by CodeDNA™, where each cluster represents elements of the code that have 'similarity scores' between 0.6 and 1.0. A similarity score is an output of CodeDNA™ that determines how similar one code binary is to another code binary. In Figure 19, the red lines represent a similarity score of 1.0, meaning the code samples are nearly identical. The blue lines represent the score of 0.6, meaning that roughly 60% of the code is similar according to CodeDNA's algorithms. The remaining links between the nodes are shaded between blue and red as the similarity score increases.

Using 0.6 as the lowest similarity score that defines a related cluster, Figure 19 shows that the outputs divide into seven clusters. To challenge the HB approach, we selected the cluster that had the most nodes (27) and the most links (292). This is almost a fully connected graph, which would have 378 links.

Figure 20 shows the first attempt at displaying this cluster's data in HB. The problem is that many of the nodes have exactly the same rank, which makes it difficult to discern how the nodes relate to each other. It would also be useful to color code the similarity scores of each edge to provide further detail about how these nodes relate to each other.

The solution is to offset the nodes slightly on the vertical in order to allow for each unique link between nodes to be displayed. In Figure 21, nodes with the same degree are increased or decreased by 0.1 so that the monotonically decreasing curve is maintained,

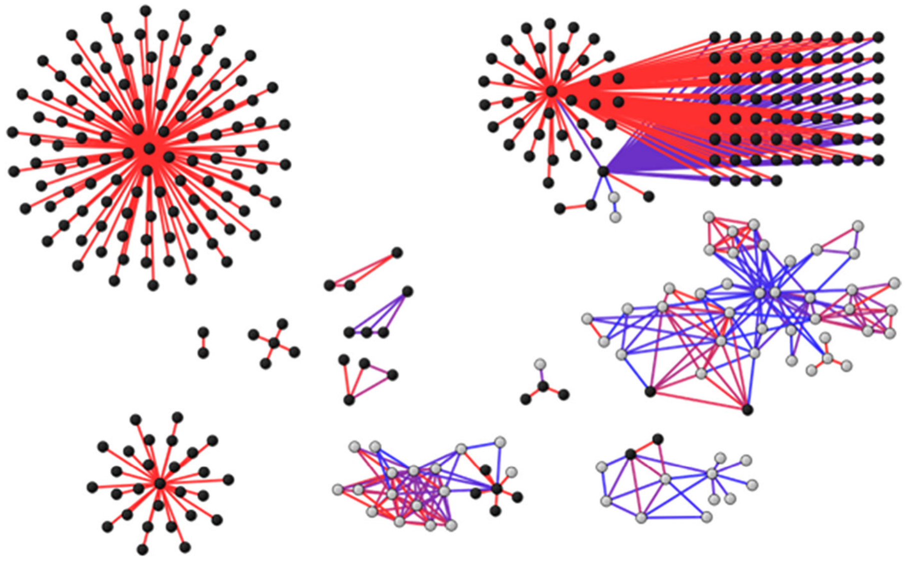


Figure 18: Sample chart of CodeDNA™ cluster outputs of malware binaries.

and is centered on the original degree value. (If there are more than nine nodes with the same degree, use a smaller offset to fit them in between the next whole number degree rows.) We added a line to connect the nodes to make it easier to see the curve.

In addition to the vertical offset, we further color-coded the similarity values of each link or edge. The following colors represent the different ranges of similarity scores: orange = 0.6 to 07, red = 0.7 to 0.8, green = 0.8 to 0.9, and purple = 0.9 to 1.0. Note that nodes 9 through 15 have high similarity scores, and a bit less similarity with nodes 7, 8, and 16, even though they share the same degree. Nodes 9 through 15 are also similar to the nodes with degree 23. Likewise, nodes 20 through 24 are similar to each other and to the nodes with degree 23. Figure 22 highlights these areas of high similarity by placing purple boxes around them.

HB algorithm is canonical

The applications above have illustrated how hairball buster can be used to answer key analytic questions. We now move on to a general discussion of the benefits of the approach and how it compares to similar existing techniques.

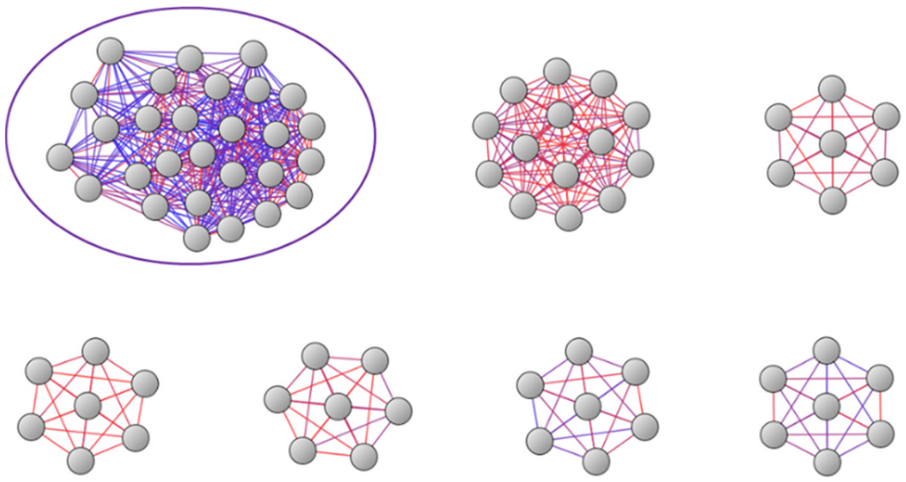


Figure 19: Sample CodeDNA™ cluster outputs of Linux coreutils binaries.

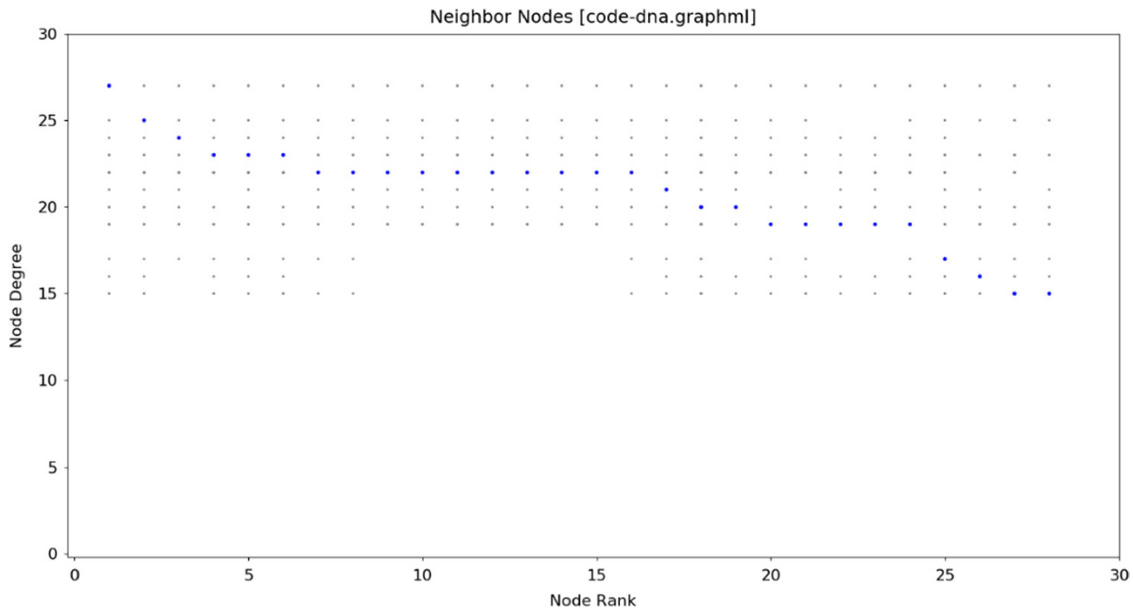


Figure 20: Sample CodeDNA™ cluster output in standard hairball buster (blue = nodes, gray dots = links).

HB is canonical in the sense that each graph has a unique visual representation. This consistency is a significant benefit, allowing different data sets, or different time slices of the same data set as in Section 'third application: HB and temporal graphs,' to be compared, regardless of size.

To ensure uniqueness, the ranking of nodes by degree must be consistent, and this ordering must be chosen at the outset. The simplest way to do this is to assign a unique label to each node and sort them alphanumerically, thus ensuring a canonical display. This was done 'behind the scenes' in the above applications.

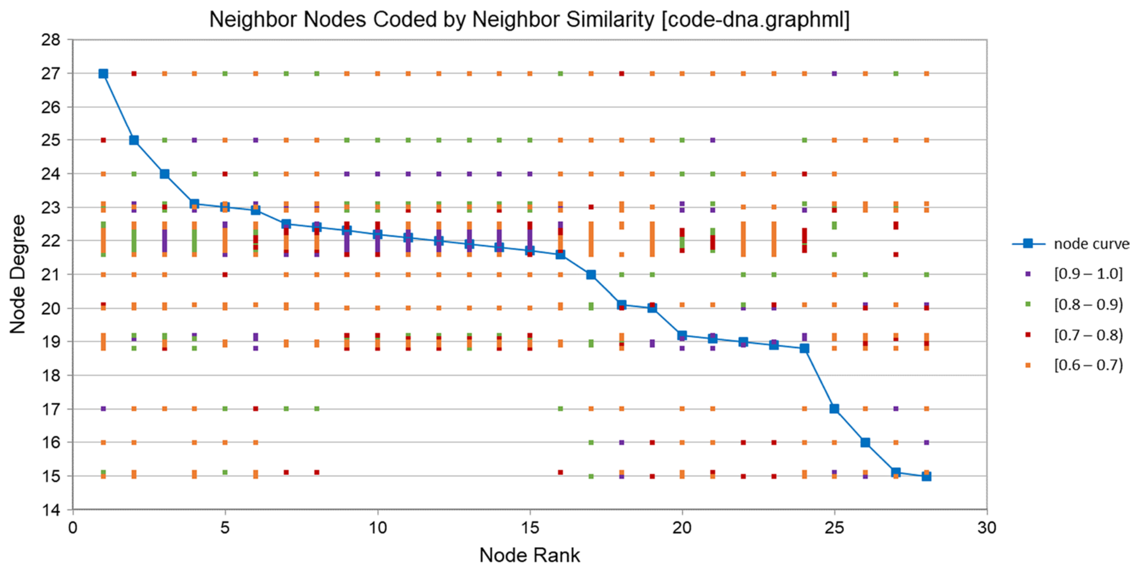


Figure 21: Sample CodeDNA™ cluster output in HB with vertical offset.

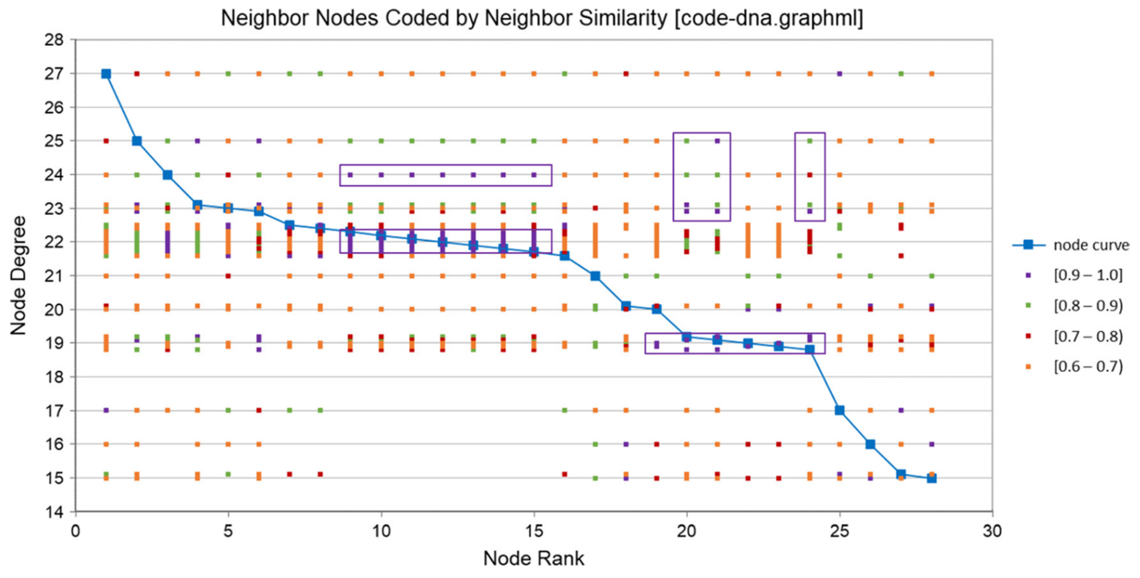


Figure 22: Sample CodeDNA™ cluster output in HB with vertical offset and highlighting nodes with highest similarity scores.

Another approach is to rank nodes that are tied by having the same degree by their connections to neighbors with the highest rank. For example, all of the one-degree nodes will be 'tied' with each other, but a tie-breaker is the degree of the node to which it is connected. If ties still exist after a first pass, then nodes can be ranked by the neighbor's neighbors. Repeat as necessary. If there is still ambiguity between any nodes, simply assign a label to the node and republish the data set so all parties interested in that data set may use the same labels.

HB measures of performance

Many traditional graph layout approaches are computationally intensive because they position each node based on distance relative to every other node and on which nodes share (or are otherwise affected by) links. This N^2 computation means that significant time is required to render a single 'hairball' graph with several thousand nodes, and many additional computationally intensive steps may be required to break the hairball into something more readily understandable.

In contrast, the HB algorithm has order $N \log N$. For small data sets such as the Jazz data set, on a single-threaded laptop there is no noticeable difference in the performance between the two. For large data sets such as the Twitter™ data set with over 200,000 nodes, however, the difference between N^2 and N Log N becomes very significant. Table 1 shows experimental results. While the run time is similar for HB and backbone for small data sets, the larger the data set, the better HB performs compared to backbone layout. For 500K nodes, backbone could not complete in 20 min, whereas HB completed in less than 30 sec. Moreover, HB consistently completed for graphs of 1 million nodes in around 45 sec, whereas the backbone algorithm could not be tested because visone could not load this volume of data.

The HB approach uses Python code to calculate the curves and neighbors from networks defined in standard graphml or csv form. It then creates Cartesian plots for the actual visualization. The approach could be probably executed much faster if optimized and parallelized.

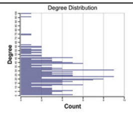
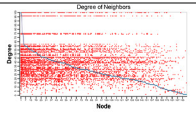
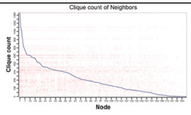
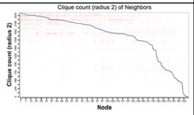
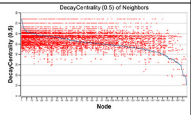
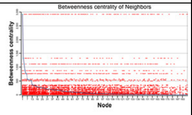
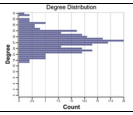
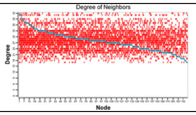
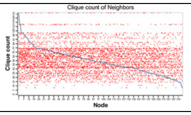
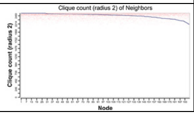
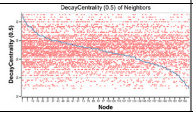
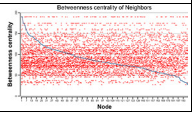
In HB, every node and neighbor's location is well defined, easily calculated, and does not change significantly when a new set of nodes or links are added. It does not answer every question, but can identify features that help effectively target what subsets of nodes to use as inputs to more computationally intensive algorithms that do answer deeper questions.

HB using other measures of centrality

In addition to degree, HB can also display the rank of the nodes by other centrality measures. These expand the type of questions that can be answered by analysts using HB.

Table 1. Performance calculations comparisons for HB vs backbone layout.

	Data sets			hb run time (s)				visone run time – quad Sim (s)				visone run time – tri Sim (s)			
Filename	File size (B)	No. of nodes	No. of edges	1	2	3	Avg	1	2	3	Avg	1	2	3	Avg
random-1000-nodes.graphml	341,365	1,000	5,002	0.25	0.25	0.25	0.25	2.0	1.7	1.6	1.8	1.5	1.1	1.3	1.3
random-10000-nodes.graphml	3,555,915	10,000	49,826	0.67	0.69	0.70	0.69	7.3	6.9	6.8	7.0	7.0	7.1	6.8	7.0
random-100000-nodes.graphml	37,271,224	100,000	500,061	10.01	11.74	6.55	9.43	139.4	120.1	118.5	126.0	129.0	119.7	119.1	122.6
random-250000-nodes.graphml	95,452,841	250,000	1,250,487	16.84	15.36	15.24	15.81	349.3	357.3	361.3	356.0	356.8	352.7	334.5	348.0
random-500000-nodes.graphml	193,263,339	500,000	2,501,346	26.21	25.71	24.47	25.46	>1,200							
random-1000000-nodes.graphml	388,461,043	1,000,000	4,997,089	44.25	43.75	45.19	44.40	Visone could not load graphml file. Insufficient memory							
code-dna.graphml	155,222	28	292	<1 sec	<1 sec	<1 sec		<1 sec	<1 sec	<1 sec		<1 sec	<1 sec	<1 sec	
jazz-directed.graphml	361,796	198	4,113	<1 sec	<1 sec	<1 sec		<1 sec	<1 sec	<1 sec		<1 sec	<1 sec	<1 sec	
toster_CA_Edge.graphml	5,349,861	23,916	75,050	1.02	0.96	0.96	0.98	20.6	19.8	20.1	20.2	17.1	18.9	17.8	17.9
iran-tweet-replies.no-retweet.by-userid.graphml	294,153,484	228,626	440,244	1.26	1.12	1.13	1.17	>1,200							

Graph	Degree Distribution	Degree Centrality	Clique Count	Clique Count (order 2)[1]	Decay Centrality (decay parameter 0.5)	Betweenness Centrality[2]
Jazz N = 198, E = 2742						
Random Edges N = 198, E = 2742						

1. Order 2 means displaying the number of triangles in the subgraph formed from all nodes within two hops of the chosen node
2. Betweenness Centrality is expensive to compute, so this version of the plot lacks the benefit of fast computation

Figure 23: Displaying different measures of centrality in HB.

We focused on the following measures. The clique count of a node with *N* neighbors as the count of the $\binom{N}{2}$ neighbor pairs that are connected. The clique count (order 2) is the number of node pairs within distance 2 of the node that are connected. The decay centrality of a node *i* is the weighted sum decay $(i) = \sum_{j \neq i} \delta^{l(i,j)}$, where δ is a decay parameter and $l(i,j)$ is the minimum distance from *i* to *j* (or infinite if the nodes are in different components). The betweenness centrality of a node measures how likely the node is to lie along the shortest paths between other nodes in the graph. More details on these centrality measures are available in the studies of Wasserman and Faust (1994) and Jackson (2010).

Figure 23 shows the HB visual for each of these measures for both the jazz data set and a randomly generated graph. The resulting figures emphasize how a node's centrality measure is related to those of its neighbors. Each chart illustrates major structural differences between the jazz graph and a random graph. For example, in the decay centrality figures, the centrality of neighbors in the jazz data sets differs significantly, whereas in the random graph case the decay centrality of neighbors is very similar (note the vertical axis shows values between 52 and 61 only).

These examples also show how HB can be applied to floating point as well as integer-valued metrics. (Note that some metric calculations, betweenness centrality in particular, are computationally expensive and may negate some of the performance advantages of the HB approach.)

Comparing HB to other algorithms

This section describes how HB differs from other commonly used graph analytic and visualization algorithms. Table 2 lists 14 analytic questions and features grouped into three sections: (i) understanding node relationships and graph characteristics, (ii) representing large or directed networks or graphs with weighted links, and (iii) the ability to represent other centrality measures and representing graphs in a standard format at low calculation cost. These are used to compare HB with several other techniques: a histogram of node centralities, a standard force-directed layout, a backbone layout, a standard adjacency matrix with nodes sorted by degree, and an adjacency matrix where nodes have been ordered based on clusters (i.e. blockmodeling).

The histogram does not provide information about neighbors, connectivity, number of clusters, weighted links, or directedness. Similarly, the adjacency matrix cannot represent centrality measures other than degree, has no visual cues for comparing one node's degree to its neighbors or finding clusters, is not usable for very large data sets, and has no log–log or semi–log representation. (This paper assumes that the adjacency matrix has already been sorted by degree in terms of both rows and columns in order to compare well against HB.)

While blockmodeling can depict the number of clusters in a graph, it cannot represent other centrality measures, log–log or semi–log representation, is not canonical, and requires a large number of calculations. Moreover, even when successfully representing clusters, blockmodeling works well only when there are several communities highly connected within blocks and having only sparse connections between blocks. Blockmodeling requires at least N^2 calculations, and most references cite N^3 calculations being required (White et al., 1976; Wasserman and Faust, 1994; Girvan and Newman, 2002; Jackson, 2010; Gopalan et al., 2012).

While force-directed and backbone layout visualizations can show some clustering, they cannot provide the distribution of nodes by degree and the

Table 2. Comparing HB features to other graph analytic and visualization algorithms.

Feature	Hairball buster	Histogram/ node-degree display	Force-directed	Visone backbone	Adjacency matrix	Block modeling
Understanding node relationships and graph characteristics						
1. Distribution of nodes by degree	Yes	Yes	No	No	No[f]	No[f]
2. Quickly determine the number of high-degree nodes	Yes	Yes	No	No	Yes	No[f]
3. Quickly identify which are the highest degree nodes	Yes	Yes[a]	No[b]	No	Yes	Yes
4. Determine if the highest degree nodes are directly connected to other high-degree nodes	Yes	No	Yes[c]	No[b]	Yes	Yes
5. Determine whether the highest degree nodes are connected to each other indirectly via two hops	Yes	No	Yes	Yes[c]	Yes	Yes
6. Determine which lower-degree nodes are directly connected to the high-degree nodes	Yes	No	Yes	Yes	Yes	Yes
7. Provide visual cue of how much difference exists between the degree of the nodes, especially high-degree nodes	Yes	Yes	No	No	No	Yes
8. Determine if there is one central cluster or many clusters that contain the highest degree nodes	Yes	No	Yes	Yes	No	Yes
Representing large or directed networks, or with weighted links						
9. Provide log–log or semi–log representation for very large data sets	Yes	Yes	No	No	No	No
10. Can visualize both directed and undirected graphs	Yes	No	Yes[e]	Yes[e]	Yes	Yes
11. Determine which nodes connect to the highest weighted links	Yes	No	Yes[d]	Yes	Yes[g]	Yes[g]
Other centrality measures, standard format, low calculation cost						
12. Distribution of nodes by other centrality measures	Yes	Yes	No	No	No	No
13. Provide a canonical representation of the graph	Yes	Yes	No	No	Yes	No
14. Low calculation cost	Yes	Yes	No	No	Yes	No[h]

Notes: [a]If displayed or available via tooltip display; [b]except for very small data sets; [c]for small graphs or when edges are not occluded; [d]in some cases; [e]if link weights displayed, e.g., by color or width; [f]unless one can count number of node or links very carefully; [g]if link weights displayed as attributes of dots in the matrix; [h]order at least N^2 and most references state N^3.

number and identity of highest degree nodes or their direct connectivity to other high-degree nodes except in very small data sets. There are also no visual cues as to how much larger one node's degree is compared to another high-degree node. Moreover, they do not represent other measures of centrality, are not canonical, and require at least N^2 calculation cost.

As shown in Table 2, the only algorithms that can come close to the HB analytic triage approach in terms of computation time is the adjacency matrix and the node-degree distribution or histogram display. Even then, the histogram cannot address six of the features available in HB, while the adjacency matrix cannot address four. Overall, HB efficiently presents information about a graph in a single display that is not available in any other single display.

We also compared these approaches visually. Figure 24 illustrates five graphs using force-based layout, a histogram, an adjacency matrix, and HB. We have shown in previous sections the variety of questions that can be answered using HB. In contrast, there is little that can be determined directly from the force-based layout except for maybe the preferential attachment graph. While the histogram correlates well with the HB curve, the histogram provides no data whatsoever about the neighbors of the nodes. The adjacency matrix shows little in the way of patterns in these examples, except for the third and fourth graph, although this may be improved by using other node orderings such as those obtained in blockmodeling.

An interesting characteristic of HB not previously discussed is also shown in the proximity graph. It is clear from the histogram that the distribution is bimodal, and the adjacency matrix shows a blob in the lower left corner. HB shows not only the cluster in the upper-left corner where most of the high-degree nodes are interconnected, but also that this cluster is mostly disconnected from other parts of the graph; this could not be discovered from the histogram and is difficult to ascertain from the adjacency matrix.

Optional steps of the HB algorithm

Section 'HB approach' presented the six basic steps of the HB algorithm. This section describes four optional steps mentioned in the previous use cases:

1. If one requires the HB chart to be canonical, then rank nodes of the same degree in lexicographic ordering based on the node labels.
2. Display the inverse (the gaps) of links to the neighbors.
3. Display a log–log or semi–log chart.

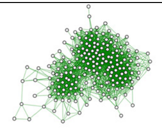
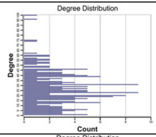
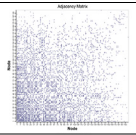
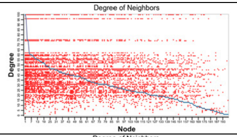
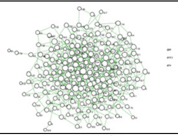
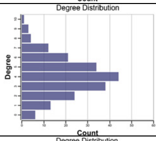
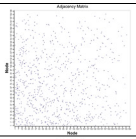
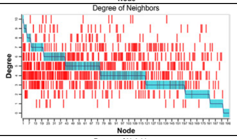
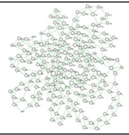
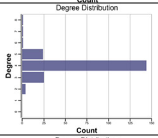
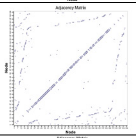
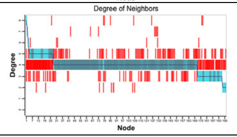
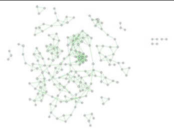
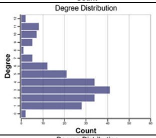
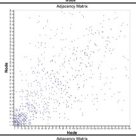
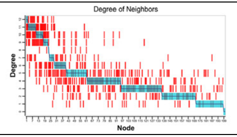
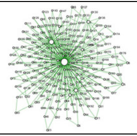
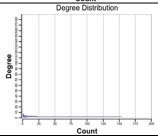
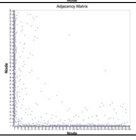
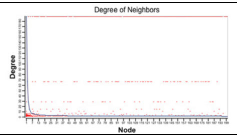
Graph Type	Force-Based Layout	Distribution	Adjacency Matrix	Neighbor-Metric Plot (HB)
Jazz N = 198, E = 2742				
Random Edges N = 200, E = 400				
Watts-Strogatz Random Graph N = 200, E = 400				
Proximity Graph N = 200, E = 401				
Preferential Attachment N = 200, E = 379				

Figure 24: Comparing different types of graphs and algorithms.

4. Display nodes with the same metric value using vertical offsets.

Displaying the inverse is performed at the time the chart is rendered. Rather than displaying the neighbor nodes at their specified X and Y coordinates, display dots where there is no neighbor node, as shown in Figures 10 and 11.

Displaying a log–log or semi–log chart benefits from offsetting the origin by 10,10 or 100,100 depending on the size of the original data set, as described in the Appendix.

Displaying multiple nodes with the same centrality measure using vertical offsets makes relationships easier to see. In this case, select the nodes of the same centrality that are of interest, as shown in Figures 21 and 22. Calculate the size of the offset based on the number of curve nodes with the same *Y* coordinate that need to fit within the space 0.5 above and 0.5 below the *Y*-value.

Limitations of HB

At the time of this writing, we have identified four limitations of the HB approach. First, if there are two or more nodes with the same degree (or other centrality measure), even though each node on the curve will have a different rank (*X*-coordinate), their neighbor nodes may land on top of each other. This reduces the ability of the HB display to allow an analyst to clearly see how the nodes and their neighbors relate to each other, as well as more difficult to identify which high-degree nodes are connected by two hops.

To address this limitation, the HB algorithm and code allows for vertical offsets for nodes that share the same degree, as described in Section 'optional steps of the HB algorithm.' Note that for most social networks, the high-degree nodes tend not to share the same degree, and when they do, usually only a small number share the same degree. For the low-degree nodes, there is little interest in identifying whether one-degree nodes are sharing the same row. If there is a dot above it, then that one-degree node is connected to that higher-degree node. If there is no dot about it, then that node is connected to another one-degree node and not connected to the rest of the graph and is an outlier.

The second limitation is that if the graph has multiple links per pair of nodes, as in a multirelational network (Zweig, 2016), then those will not appear on the HB chart. To address this, one could translate the number of links into a link weight and color-code the weights as shown in Figures 21 and 22. However, if the multiple links each have their own weights, most approaches – including HB, the adjacency matrix, and blockmodeling – would be unable to represent the weights.

Third, HB is not designed to represent loops, or nodes that connect to themselves. While this is not usually an issue for social network graphs, it is a limitation for the basic HB algorithm. One could apply workarounds, such as a vertical offset if not otherwise being used in this HB case, or one could extend the HB display to a third dimension.

Fourth, while Figure 9, for example, provides a clear indication of which nodes have the highest degree and whether they are highly connected to other (neighbor) nodes, it can be difficult to identify exactly which of the highest degree nodes are connected to each other due to the large number of nodes being displayed. To solve this display problem, we recommend taking the inverse and displaying the gaps when encountering situations with very large numbers of highly connected nodes, as well as displaying the top 1% of the highest-ranked nodes. (This requires no new calculations – just selecting the range of top nodes to zoom in on.) Figure 10 is an example of applying this solution, and clearly shows how few links are missing from the top 20 nodes to be fully connected.

When to use HB

Given the strengths and limitations of the HB approach, when should an analyst use or not use HB? The authors recommend that HB be used as the initial algorithm to apply to a data set because of its information density and computational efficiency. As a triage method, HB can provide in the first pass the number of highest degree nodes, how they relate to each other, and how they relate to their neighbors. For graphs with large number of nodes that cannot be visually separated in the full HB plot, zooming in on the top nodes provides a computationally inexpensive way to get the same information.

The results of this triage can indicate areas of particular interest, such as gaps in the curve or neighbor nodes. Moreover, if a graph reference library (see Section 'future features and applications of HB') is available, the new, unknown data set can be quickly compared to its closest matches of known data sets, thereby suggesting likely underlying structures and algorithms to try next.

HB may be less useful for very small graphs, when the structure of the graph is already well-understood, when an analyst already knows exactly what metrics to compute, or, as indicated by the limitations above (Section 'limitations of HB'), when the graph structure includes multiple links per node pair or links that loop back to the same node.

Future features and applications of HB

The first planned future feature is to create a graph reference library (GRL) to compare new, unknown graphs to a set of graphs whose underlying structure is known. For example, curves generated by exponential random graph models (ERGMs) will appear different from known social media data curves. Once the known curves closest to the unknown curve have been identified, one can also display their neighbors and compare the neighbor distribution to the unknown graph neighbor distribution. This can provide a significant benefit to an analyst by quickly recommending known graphs to consider when analyzing a new graph to better understand its underlying structure. This comparison approach could also be automated or semi-automated by using convolutional neural networks to make these comparisons more thoroughly. Note that such a broad range of comparisons is possible because the HB representation is canonical. Creation of the GRL and the ability to display multiple curves on the same chart for purposes of comparison will be addressed in a subsequent paper.

One proposed visualization approach is to provide cross-highlighting or 'brushing' capabilities among different types of displays. For example, mousing over the HB display could not only provide additional information as tooltips, but by connecting to other display types such as backbone layout, also highlight the same nodes in other displays. This ability to cross-highlight selections in multiple displays would provide particular benefit in the examination of temporal displays, identifying which have changed positions in the curve and which have not.

An alternative visualization approach for highlighting similarities and differences in temporal graphs is to highlight which nodes have not changed rank by more than one or two, and so on for a selected number of bands identifying such changes. This method of triage will also help analysts quickly focus on similarities and differences in temporal displays.

Summary of advantages of HB

Hairball buster (or node-neighbor centrality) is an approach that provides a computationally efficient approach to graph analytic triage. HB provides a unique, canonical representation of any node-link data set. The ability of HB to provide a standard representation allows different node-link data sets, or different time slices of the same data set, to be compared to identify anomalies or large structural changes. The computational efficiency of HB is on the order of *M*, where *M* is the number of links, plus *N* log *N*, where *N* is the number of nodes.

Because of its computational efficiency, HB can act as a triage method to identify key features of a data set, including whether the curve appears more representative of a social network or a random graph. It can also be used to quickly identify how many high-degree nodes are in the graph, which are the highest ranked nodes, whether those nodes are connected to each other directly or by two hops, and how connected the higher ranked nodes are to the lower-ranked nodes.

In addition to degree, HB can visualize graphs using other centrality measures such as clique count, decay centrality, and betweenness. This flexibility of HB to represent a wide range of centrality measures is a significant benefit to analysts.

This paper also presented differences between HB and force-directed and backbone layout visualization algorithms. In each case, HB provides greater information density than other algorithms at lower or equal calculation cost. Overall, HB presents information about a graph in a single display that is not available in any other single display and can complement the analyst's existing toolkit.

Acknowledgments

No external funding was used to develop the Hairball Buster approach and code. The Johns Hopkins University Applied Physical Laboratory (JHU/APL) funded a small internal research and development seedling project to develop the initial code in Python (developed by co-author Mark Matties) and Java (developed by co-author Elisha Peterson). The authors would also like to thank the following for providing data sets (Cetin Savkli for the Jazz player data set, Matt Elder and Janis Butkevics for the Toaster data, Bobby Seng for the CodeDNA™ data, and Mark Matties for the Iranian Twitter™ data), Marc Johnson for complexity algorithm citations, and Roger Butler for recognizing that the HB approach was canonical.

References

Bender-deMoll, S. and McFarland, D. A. 2006. The art and science of dynamic network visualization. *Journal of Social Structure*, 7.

Brandes, U. and Corman, S. R. 2003. Visual unrolling of network evolution and the analysis of dynamic discourse. *Information Visualization*, 2(1): 40–50, available at: https://doi.org/10.1057/palgrave.ivs.9500037.

Freeman, L. C. 2000. Visualizing social networks. *Journal of Social Structure*, 1(1): 4, available at: https://www.researchgate.net/profile/Linton_Freeman/publication/242008428_Social_Network_Visualization_Methods_of/links/57516bfc08ae02ac12759651.pdf.

Fruchterman, T. M. J. and Reingold, E. M. 1991. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11): 1129–1164, available at: https://doi.org/10.1002/spe.4380211102.

Ghoniem, M., Fekete, J.-D. and Castagliola, P. 2005. On the readability of graphs using node-link and matrix-based representations: a controlled experiment and statistical analysis. *Information Visualization*, 4(2): 114–135, available at: https://doi.org/10.1057/palgrave.ivs.9500092.

Girvan, M. and Newman, M. E. J. 2002. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12): 7821–7826, available at: https://doi.org/10.1073/pnas.122653799.

Gleiser, P. and Danon, L. 2003. Adv. Complex Syst.6, 565, available at: http://deim.urv.cat/~alexandre.arenas/data/welcome.htm as cited on the Konect website: http://konect.uni-koblenz.de/networks/arenas-jazz.

Gleiser, P. M. and Danon, L. 2003. Community structure in jazz. *Advances in Complex Systems* 6(4): 565–573, available at: https://www.worldscientific.com/doi/abs/10.1142/S0219525903001067.

Gopalan, P. K., Gerrish, S., Freedman, M., Blei, D. M. and Mimno, D. M. 2012. Scalable inference of overlapping communities. In Pereira, F., Burges, C. J. C., Bottou, L. and Weinberger, K. Q. (Eds), *Advances in Neural Information Processing Systems*. MIT Press, Cambridge MA, pp. 2249–2257, available at: http://papers.nips.cc/paper/4573-scalable-inference-of-overlapping-communities.pdf.

Jackson, M. O. 2010. *Social and Economic Networks*, Princeton University Press, Princeton and Oxford.

Kamada, T. and Kawai, S. 1989. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1): 7–15, available at: https://doi.org/10.1016/0020-0190(89)90102-6.

Lehmann, K. A. and Kottler, S. 2007. Visualizing large and clustered networks. In Kaufmann, M. and Wagner, D. (Eds), *Graph Drawing.* Springer, Berlin and Heidelberg, pp. 240–251.

Maughan, D. and Carlsten, N. 2018. *Transition to Practice Technology Guide.* Department of Homeland Security Washington DC, available at: https://www.dhs.gov/sites/default/files/publications/CSD_TTP_Guide_2018_webversion_06262018_508%20Final.pdf.

Moody, J., McFarland, D. and Bender-deMoll, S. 2005. Dynamic network visualization. *American Journal of Sociology*, 110(4): 1206–1241, https://doi.org/10.1086/421509.

Nocaj, A., Ortmann, M. and Brandes, U. 2014. Untangling hairballs. In Duncan, C. and Symvonis, A. (Eds), *Graph Drawing.* Springer, Berlin and Heidelberg, pp. 101–112.

Nocaj, A., Ortmann, M. and Brandes, U. 2015. Untangling the hairballs of multi-centered, small-world online social media networks. *Journal of Graph Algorithms and Applications* 19(2): 595–618, available at: https://doi.org/10.7155/jgaa.00370.

Peterson, E. 2011. Time spring layout for visualization of dynamic social networks. *2011 IEEE Network Science Workshop*, pp. 98–104, available at: https://doi.org/10.1109/NSW.2011.6004630.

Sheny, Z. and Maz, K.-L. 2007. Path visualization for adjacency matrices. *Proceedings of the 9th Joint Eurographics/IEEE VGTC Conference on Visualization*, 83–90, available at: https://doi.org/10.2312/VisSym/EuroVis07/083-090.

Squartini, T., Mastrandrea, R. and Garlaschelli, D. 2015. Unbiased sampling of network ensembles. *New Journal of Physics*, 17(2): 023052, available at: https://doi.org/10.1088/1367-2630/17/2/023052.

Twitter™ 2018. Twitter™ website, data set regarding election integrity, PERISCOPE, SCOPE and the Periscope logo are trademarks of Twitter, Inc. or its affiliates, available at: https://about.twitter.com/en_us/values/elections-integrity.html#data (accessed November 8, 2018).

Ware, C. 2010. *Visual Thinking: for Design*. Elsevier, Amsterdam.

Wasserman, S. and Faust, K. 1994. Social network analysis by Stanley Wasserman, Cambridge University Press, available at: https://doi.org/10.1017/CBO9780511815478 (accessed October 29, 2019).

White, H. C., Boorman, S. A. and Breiger, R. L. 1976. Social structure from multiple networks. I. blockmodels of roles and positions. *American Journal of Sociology*, 81(4): 730–780, available at: https://doi.org/10.1086/226141.

Zweig, K. A. 2016. *Network Analysis Literacy: a Practical Approach to the Analysis of Networks.* Springer Science & Business Media, Wein, p. 115.

Appendix. Semi–log and log–log displays for the hairball buster approach

For very large data sets, a Cartesian representation of the HB algorithm may not be sufficient to encompass the whole data set. Although the HB approach has been successfully applied to data sets with over 600,000 nodes displayed on Cartesian coordinates, there exist much larger data sets for which a semi–log or log–log display would be needed to represent all of the data in a single HB chart. (In this appendix, we will always be referring to log-base-10.)

When simply taking the semi–log or log–log of a data set, we immediately discovered that the display is dominated by the first few nodes, leaving little visual benefit in the remaining part of the chart. See Figure A1 for an example of a log–log display based on the jazz player data set. This does not present particularly useful information to the analyst.

However, there is an easy solution to this problem. By adding either 10 or 100 to all of the data points, we are essentially creating an 'offset' of the origin to point 10,10 or point 100,100. Figure A2 shows an offset of the origin to the point at coordinates 10,10 for the Jazz Player data set. Although a relatively small data set,

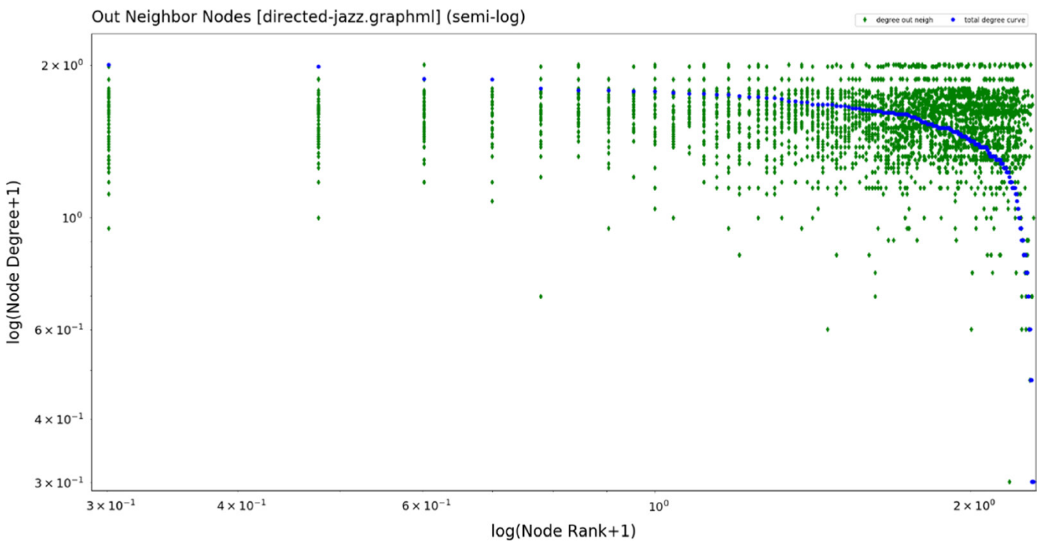


Figure A1: Sample Log_{10}–$\log_{10}$ plot of jazz player data set with no offset.

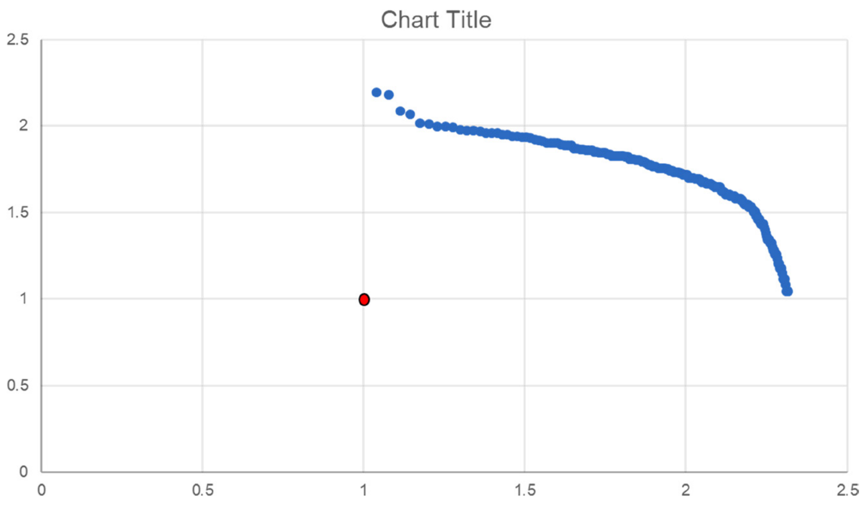


Figure A2: Sample offset of origin to 10,10 for Log_{10}–$\log_{10}$ plot of jazz player data set.

this example shows how the offset of the origin allows a much smoother and continuous representation of the curve compared to Figure A1. (One needs to remember that when reading the chart, the origin has been offset.)

Figure A3 shows the Toaster data set in semi–log format. No offset was needed because the log of one is 0. Since most of the connections are of degree 1, the origin of zero–zero works. However, the display tool the authors applied would not display anything at coordinate value of 0 for the *Y*-axis. Therefore, we placed the degree one nodes at the lowest *Y*-value on the semi–log display.

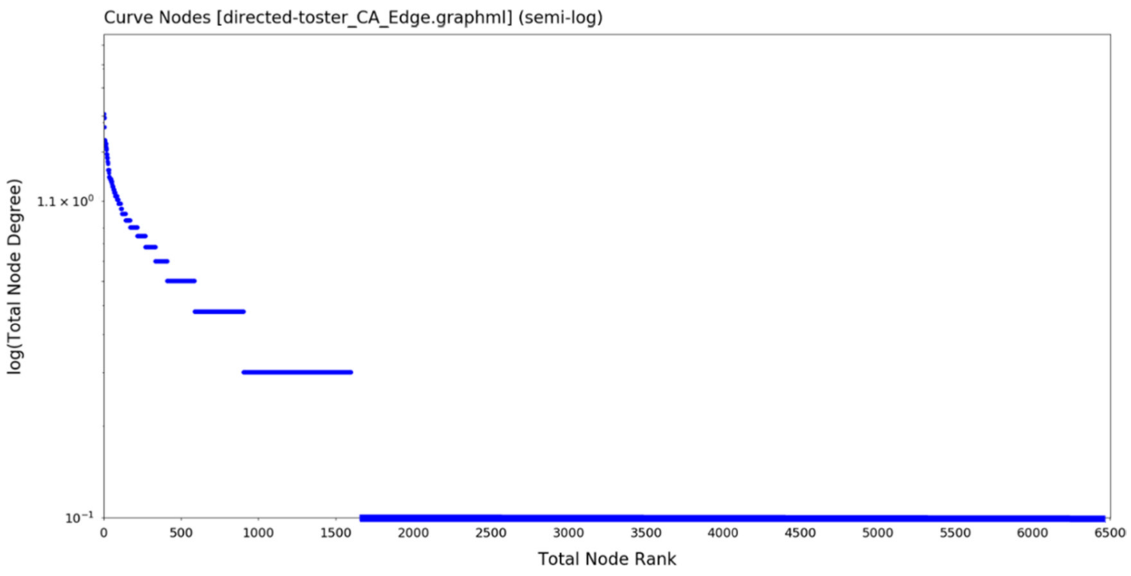


Figure A3: Sample offset of origin to 10,10 for semi–log plot of Toaster data set.