Visualizing Distributed Memory Computations with Hive Plots

Sophie Engle
University of San Francisco
Harney Science Center, Room 532
2130 Fulton Street
San Francisco, CA 94117-1080
sjengle@cs.usfca.edu

Sean Whalen
Columbia University
450 Computer Science Building
1214 Amsterdam Avenue, Mailcode: 0401
New York, NY 10027-7003
swhalen@cs.columbia.edu

ABSTRACT

A *hive plot* is a network layout algorithm that uses a parallel coordinate plot in which axes are radially arranged and node position is based on structural properties of that node [8]. We apply hive plots to message-passing communication networks formed by different high performance computing applications at Lawrence Berkeley National Laboratory. Hive plots have advantages over common network visualization methods, abandoning the popular "hairballs" of force-directed network layouts and opting instead for a tunable, repeatable, and interpretable view of network-theoretic properties. Small multiples of these hive plots, called *hive panels*, are analyzed to suggest which properties contribute to accurate classification of application behavior for anomaly detection in high performance computing environments.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; E.1 [**Data**]: Data Structures—*Graphs and networks*; I.3.0 [**Computer Graphics**]: General

General Terms

Design, Experimentation, Security

Keywords

Hive plots, network visualization, high performance computing, anomaly detection

1. INTRODUCTION

Distributed computing systems, including clouds and high performance computing clusters, are attractive targets for misuse by both external attackers and insiders. An oft-neglected class of such systems are the supercomputers operated by government labs which consistently rank among the fastest in the world, yet are not known to have safeguards against cracking encryption keys or performing nuclear simulations for hostile entities, for example.

The de-facto standard for distributed memory parallel programming in high performance computing environments is Message Passing Interface (MPI). Programs written using MPI can scale to thousands of nodes, where each node performs local computation and requests non-local data from remote nodes using MPI calls. The pattern of MPI calls is highly structured and tied to the underlying algorithm; similar algorithms express similar patterns, and computational equivalence classes have been found [5].

Communication between nodes is modeled as a directed graph where ranks are nodes and MPI calls are edges. Such a graph may have edges scaling exponentially with the number of nodes, which themselves may number in the dozens, hundreds, thousands, or even tens of thousands for typical scientific computations. The analysis of such non-trivial graphs is the domain of network theory where they are termed *complex networks*.

Dozens of statistics exist for quantifying the structure of complex networks, but information overload is a pervasive problem with their visualization—drawing thousands of nodes with millions of edges in two or three dimensions is a challenging problem with no universal solution. Visualizations using popular force-directed layouts are helpful for small networks, but quickly lose their interpretability and often lack discussion of what insight (if any) was gained by their use. The problem is compounded when multiple properties are overlaid onto the same visualization. This poses a catch-22 for machine learning and its application to computer security in the form of anomaly detection: visualization can identify predictive features and reduce the dimensionality of both data and model, yet identifying relevant patterns can be difficult in the presence of visual complexity.

This paper explores a recent visualization technique called hive plots [8] that aim to resolve problems with current force-directed layouts and present a tunable, repeatable, and interpretable view of complex networks. We apply hive plots as an exploratory step to identify which network properties may lead to better classification results and more accurate anomaly detection, and demonstrate the advantages of hive plots over traditional network layouts for revealing communication patterns while remaining easily comparable across datasets. We first discuss details of our network data including relevant background on high performance computing and communication logging. We then review hive plots, discuss our implementation, and demonstrate how hive plots contribute to our understanding of predictive network characteristics.

2. DATASET

Our dataset consists of MPI calls made between the computational nodes of several high performance computing applications at Lawrence Berkeley National Laboratory. From these logs we generate directed multiple-edge graphs and compute per-node network measures using the `igraph` library [7] in the `R` statistical computing package. In this section we provide additional detail on this dataset and how it was collected, as well as describe the data processing steps taken prior to visualization.

2.1 Background

Parallel programming models can be broadly divided into shared memory models where computations share a common address space, and distributed memory models where computations are restricted to a local subset of global memory. Message Passing Interface (MPI) is the de-facto standard for distributed memory parallel programming in both industry and academia, with many cross-platform implementations optimized for different combinations of computational and networking hardware and software. The basic model involves computational nodes (referred to as *ranks* due to the unique identifier assigned to each node) performing computation on local data and requesting data from remote nodes when requiring access to non-local data. Communication between nodes is thus an essential characteristic of MPI, and implementing efficient communication for a given algorithm is often the subject of dedicated research areas.

Due to the link between data access and communication, the patterns of messages exchanged between compute nodes is closely tied to the underlying algorithm. Thus, different algorithms will result in different patterns of MPI calls, while algorithms with similar memory access patterns may exhibit similar communications. This is a subtle but important distinction from a security perspective: While we can obtain ground truth about communication patterns by capturing network traffic or logging calls using shared libraries, we cannot know what code is executing on a system by examining job submission logs or looking at binary names since such information is trivial to manipulate. An attacker could run an executable with the same name as a molecular dynamics simulator while the underlying computation brute forces an encryption key. However, if they wish to perform a certain kind of computation they must either make the necessary calls which will be patterned, or try to fool an observer by emulating other patterns at the cost of efficiency for their true goal.

2.2 Data Collection

In order to visualize communication patterns and identify useful attributes for anomaly detection, we first need a mechanism of observing MPI calls. To collect communications from running MPI programs we use a shared library called *Integrated Performance Monitoring* (IPM) [4]. The IPM library provides low-overhead logging of MPI call features such as the call name, the source and destination rank, the number of bytes sent, and aggregate performance counters such as the number of integer and floating point operations.

Our original dataset includes 1681 logs of 29 MPI-enabled scientific computations collected for Lawrence Berkeley National Laboratory by the National Energy Research Scientific Computing Center. Multiple logs exist for each program with varying ranks, parameters, architectures, and datasets when possible. Several simpler codes were logged by us; codes requiring significant domain knowledge or private datasets were logged from willing specialists on production systems. As a result, the inputs and parameters for some codes were not under our control. However, this dataset is several orders of magnitude larger than related efforts and we believe contains a representative sample of the application classes found in scientific computing.

Our analysis focuses on 8 of the 29 scientific computations collected in the original dataset: `cactus` (astrophysics), `ij` (algebraic multi-grid), `milc` (lattice gauge theory), `namd` (molecular dynamics), `paratec` (materials science), `superlu` (sparse linear algebra), `tgyro` (magnetic fusion), and `vasp` (materials science). These codes mostly span the basic classes of scientific computation outlined by Asanovic et al. [2] with some overlap to demonstrate how hive plots distinguish similar networks. To equalize visual comparison we use a single log from each code running on 64 computational nodes.

2.3 Preprocessing

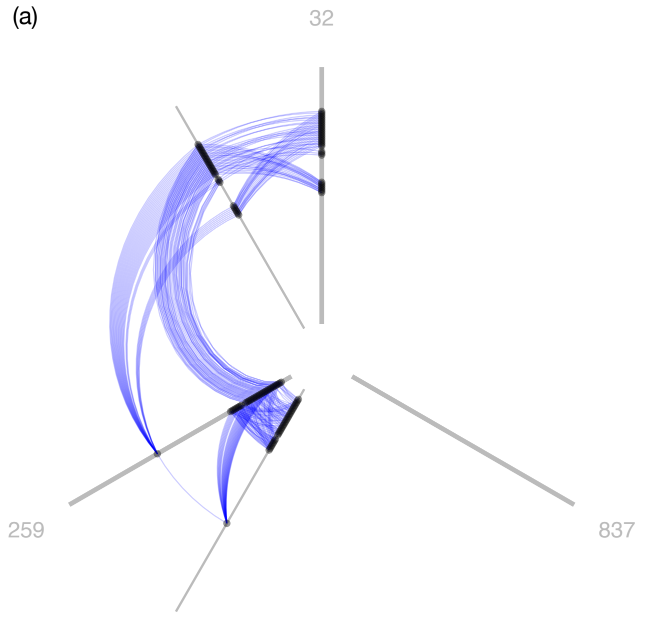
We first construct directed graphs from each log with MPI call names as edge labels. For greater interpretability, we focus on point-to-point MPI calls and filter out broadcast messages. We load these directed, multiple-edge graphs in the `R` statistical computing package and use the `igraph` library to calculate per-node network properties [7]. Since these graphs contain one directed edge per type of MPI call made, edge counts range from 1000 to over 10000. The `igraph` library was unable to calculate certain properties for extremely dense graphs such as `paratec` and `vasp`, so these codes were dropped from our final analysis.

We calculate the degree, betweenness, closeness, eccentricity, page rank, and transitivity for each node and distinguish between incoming and outgoing edges where appropriate. For example, we calculate closeness for all edges, only incoming edges, and only outgoing edges. Our final analysis focuses on degree, betweenness (undirected), page rank, and transitivity since these measures produced the most informative hive plots. The degree of a node is the number of edges connected to that node. The betweenness of a node indicates how many shortest paths travel through that node. Transitivity, also known as the clustering coefficient, measures the connectivity of a node's neighbors. Finally, page rank is an eigenvector centrality measure.

3. HIVE PLOTS

A hive plot is a network layout algorithm that uses a parallel coordinate plot in which axes are radially arranged and node position is based on structural properties of that node, introduced by Krzywinski et al. [8]. Each hive plot is created using several rules: axes assignment and node positioning rules, axes layout rules, and edge drawing rules. We achieve repeatable, comparable layouts by using consistent rules across datasets.

For example, to create a hive plot using the degree of each node, we must first create simple Boolean tests to assign each node to a single axis. The following is an example of such a test: "Is the node degree less than or equal to 32?" We then use the degree value to determine the node's position



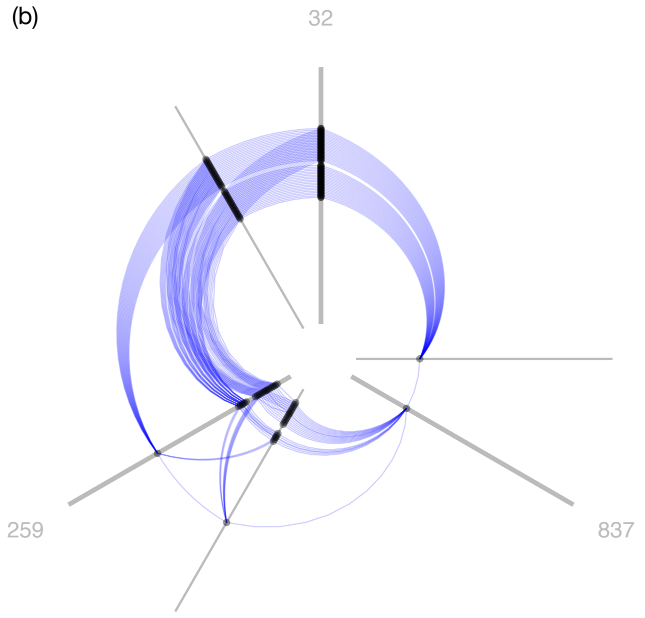


Figure 1: Panels (a) and (b) provide hive plots for degree of the `cactus` and `tgyro` datasets respectively.

on the assigned axis. For example, a node of degree 32 will be placed at the outer-most position on the axis since the maximum degree allowed by our example axis assignment test is 32. Edges are drawn as curves between nodes, with special rules governing how edges are drawn between nodes on the same axis. Example hive plots based on degree are given in Figure 1.

Several implementations of hive plots are available from the project's website at `hiveplot.net`. However, we found that these implementations did not meet our needs. We created a custom implementation of hive plots in the `R` using the `ggplot2` library [15]. We chose the `ggplot2` library for its adherence to well-established visual design principles [11, 16]. We followed many of the recommended axes assignment, layout, and edge drawing rules from the original paper [8], but did make several design modifications to maximize our specific goals. We discuss these decisions next.

3.1 Axes Assignment

Axes assignment rules determine the number of axes to use in each hive plot and how to assign nodes to those axes. We use three axes positioned every $2\pi/3$ radians to avoid edge crossings over axis lines. However, there may still be edges between nodes on the same axis. We discuss how these edges are drawn in Section 3.3.

Each hive plot visualizes a single attribute such as degree, betweenness, or other per-node network properties. We calculate the minimum and maximum value of each attribute across all datasets and divide that range between the three axes using 3-quantiles at probabilities 0.25 and 0.75. Each dataset uses the same range of values for each axis to allow for comparison across datasets.

For example, see panel (a) of Figure 1. This hive plot uses degree to assign nodes from the `cactus` dataset to different axes. The north axis contains nodes with degree less than or equal to 32, while the south-west axis contains all nodes with degrees greater than 32 but less than or equal to 259. The south-east axis contains all nodes with degree greater than 259 and less than or equal to the maximum value of 837, but no such nodes exist for the `cactus` dataset. Contrast this to the hive plot in panel (b) for the `tgyro` dataset, which uses the same cutoff values for each axis and contains one high-degree node on the south-east axis.

3.2 Axes Layout

Each node has an angle (theta) and distance (radial value) for plotting in polar coordinates. The axes assignment rules determine the theta value for the node position. The axes layout rules determine the radial value of a node along an axis. We normalized the axis length to 1 and explored different approaches for calculating node positions.

In order to address the case where many nodes share the same attribute value, we select from three different approaches to calculate node positions. For example, the `superlu` dataset has 25 nodes with degree 79. Figure 2 shows each approach.

The first method, illustrated by panel (a), orders nodes by attribute value and evenly spaces nodes along the axis. While this approach maximizes the space used along the axis, the actual range of attribute values are obscured. For example, it appears that the degree values for `superlu` on the south-west axis range from 32 to 259 in panel (a), whereas in reality the values range from 79 to 175.

Our second method uses linear interpolation from the minimum to maximum possible value for an axis. This approach provides a better estimation of the actual range of values on a given axis, but nodes with the same value are plotted at the same location. As a result, only one node or edge appears where several occur. For example, there are 25 nodes with degree 79, but only one node appears with that value in panel (b) of Figure 2.

To address this issue, we combine linear interpolation with

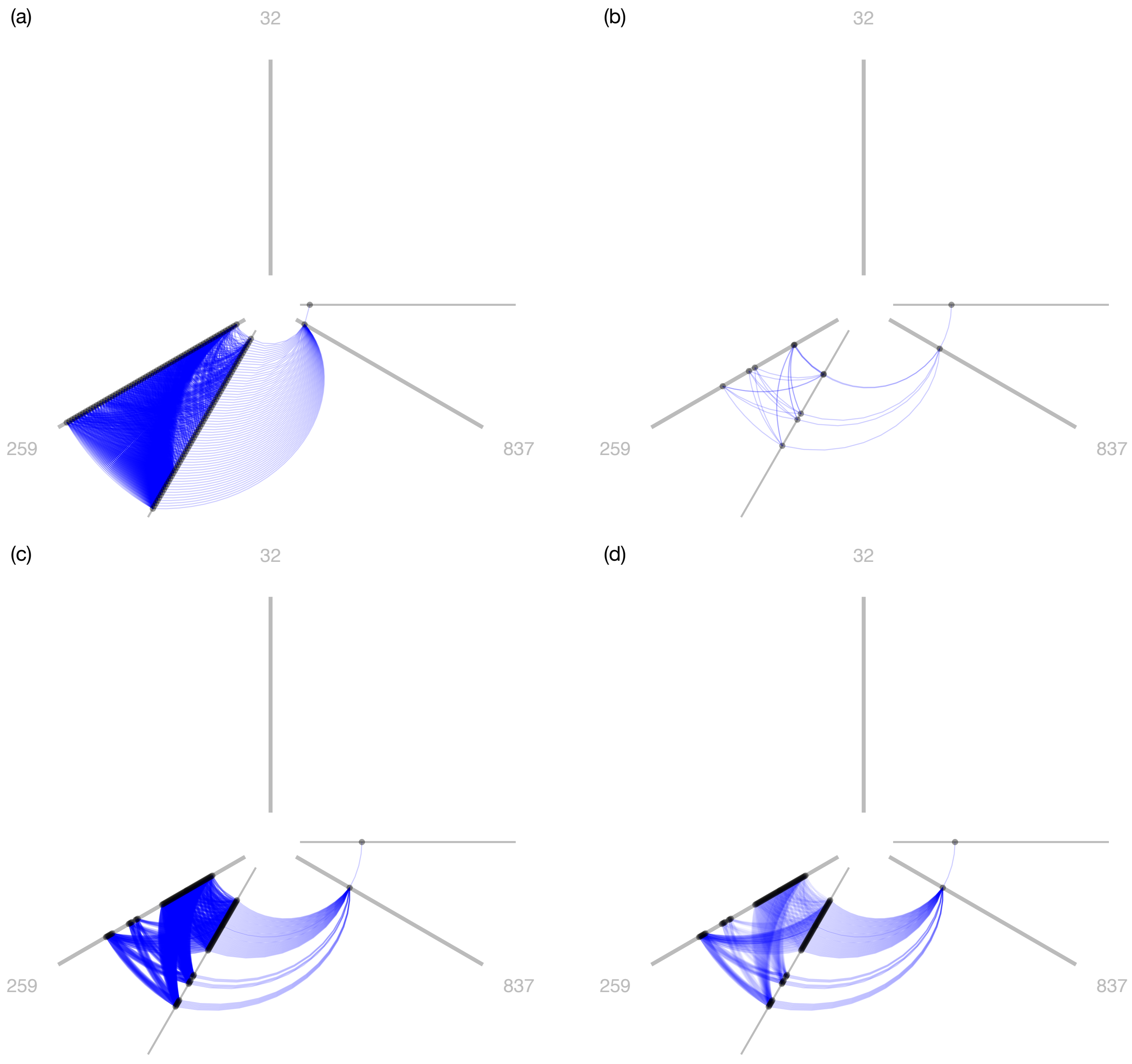


Figure 2: Hive plots of degree for the `superlu` dataset using different node layout and edge drawing rules. Panel (a) uses evenly-spaced node positions and fixed edge alpha values. Panel (b) uses linear interpolation for node positions and fixed edge alpha values. Panels (c) and (d) uses linear interpolation plus a repelling scheme for node positions and fixed versus variable edge alpha values, respectively.

a repelling scheme so nodes with the same value appear clustered next to each other. We can still visually estimate the range of values on an axis, but are also able to estimate the number of nodes on an axis and how many of those nodes share the same value. This approach is illustrated in panel (c) of Figure 2.

3.3 Edge Drawing

All edges are drawn as curves between the polar coordinates associated with each node. If there are any edges between nodes on the same axis, the axis line and associated nodes are duplicated $\pi/6$ radians next to the original axis. Edges between nodes on the same axis are drawn between the original and duplicate axis lines. For example, the edge between the node on the south-east axis and its duplicate in Figure 2 indicates that node has a self loop.

We also address the visual density of edges between axes in high-degree datasets such as `ij` and `namd`. In some cases, aggregate edge density obscured individual edge lines and the resulting communication patterns. For example, the density of edges between the original and duplicate axis in panel (b) of Figure 2 results in a nearly solid block of color.

The visual density of these edges is not an accurate reflection of actual edge density in the original graphs, especially since the hive plots do not convey multiple edges or directionality. However, lowering the overall alpha (opacity) value for the entire graph forces edges such as the one between the south-east axis and its duplicate axis to disappear. Instead, we calculate a per-edge alpha value that approximates nearby edge density. The result of this variable per-edge alpha is shown in panel (d) of Figure 2, which better visualizes individual edges between the axis lines and the banding structure caused by the communication between these nodes.

3.4 Improvements

There are several directions for improving our hive plot implementation. First, we use linear scaling to position nodes along axes. Future implementations would allow for additional scales. Second, we use a normalized axis length that creates the appearance of each axis covering an equal range of values. We address this by including labels indicating the range of each axis, but believe non-normalized axis length is also informative. Lastly, we did not experiment with approaches for conveying the directionality and multiple edges present in our communication networks with hive plots. This is a limitation of hive plots in general and is not specific to our implementation, though it may be possible through a combination of color, alpha values, and edge decoration.

4. RESULTS

Our primary goal is not to use visualization directly for anomaly detection, but instead to use exploratory data visualization to help identify which network properties are likely to improve our classification results for anomaly detection. Towards this goal, visualization must reveal communication patterns between nodes while remaining easily comparable across datasets. This section discusses how our approach meets this goal.

4.1 Communication Patterns

Plotting the complementary cumulative distribution function (CCDF) of network properties is a standard visualization technique. For example, CAIDA provides CCDF plots to compare Internet topologies for network properties including degree, betweenness, eccentricity, and transitivity [10]. We show the CCDF of degree for our datasets in Figure 3. From this plot we are able to compare the degree distributions of different datasets. However, the communication patterns exhibited by these networks are lost in the CCDF plot.

Notice in Figure 3 that the CCDF plots for `cactus` and `tgyro` are similar despite the codes having very different functionality (astrophysics vs. magnetic fusion). Compared to their hive plots in Figure 1 we can still see that both datasets have similar degree ranges, but the communication patterns between these nodes exhibit different shapes and slightly different banding patterns between axes. Hive plots allow us to visually differentiate between these datasets.

Instead of CCDF and hive plots, one may use other network layout algorithms to visualize communication patterns present within these networks. Many of these algorithms are unsuitable for comparison of networks, as we discuss next.

4.2 Comparability

The result of force-based network layout algorithms for medium and large networks are sometimes referred to as "hairballs" as the number of nodes and edges obscure any discernible structure. The unpredictability inherent in such algorithms makes it difficult to compare different networks, or even compare the the same network with minimal structural changes such as adding or removing a single node.

Hive plots address the underlying issues that make force-based network layout algorithms unsuitable for network comparison [8]. This is especially true of hive panels, which are small multiple visualizations of hive plots across attributes and datasets using consistent assignment, layout, and edge drawing rules. Using hive panels we are able to directly compare the "quantitative visual signature" of each plot across datasets.

A hive panel for degree, undirected betweenness, transitivity, and page rank is shown in Figure 4. From the panel we see `ij` and `namd` share similar patterns for degree and page rank. We expect similarities in these datasets since both have nearly fully-connected topologies. However, we can visually distinguish between them by looking at their betweenness and transitivity properties instead. The codes `cactus` and `milc` have similar topologies as well, and as a result the betweenness plots are quite similar except for two high-valued nodes in `cactus`. However, they exhibit very different degree plots due to the diversity of MPI calls made between nodes.

The properties shown in Figure 4 were selected for their ability to visually distinguish codes. Other properties resulted in informative but less distinct hive plots, and we expect will be less informative for building models of normal behavior for anomaly detection. The evaluation of this hypothesis is left for future work.

5. RELATED WORK

There are many applications of visualization to intrusion detection and specifically anomaly detection, often focusing on post-hoc analysis independent from system design. Closest in spirit to our approach are papers by Xin et al. [17] and Laskov et al. [9] which use visualization to guide feature selection during the initial design and refinement of intrusion detection systems as opposed to the more common post-hoc analyses of log data, for example.

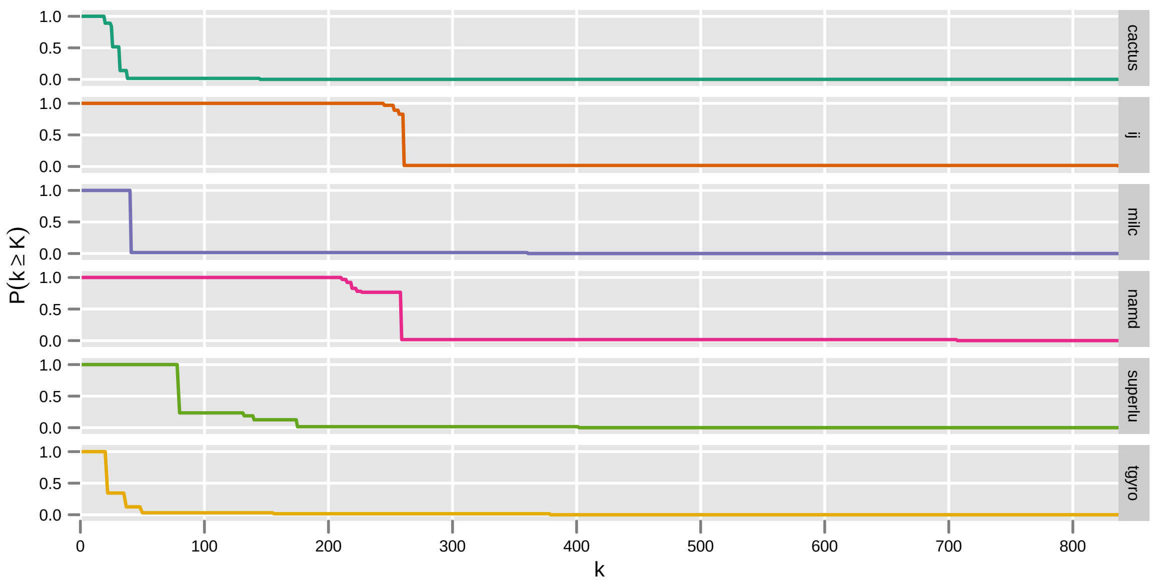


Figure 3: A complementary cumulative distribution function (CCDF) plot for degree. The shape of the CCDF plots for `cactus` and `tgyro` are similar despite the codes having different functionality.

Parallel coordinates [12] are a well-established method for visualizing multivariate data and have been previously applied in a security context [3, 6]. A hive plot is essentially a parallel coordinate plot with a radial axis layout where nodes are placed using the projection of structural properties of a graph. However, as discussed by Krzywinski et al. [8], these differences lead to more compact and interpretable representations over related techniques such as parallel coordinates or semantic substrates [1]. Previous applications of hive plots focus on biological networks such as gene regulation and gene interaction networks.

Previous work using this dataset found that MPI communication patterns reveal the underlying computation with high accuracy [13, 14]. This result was motivated by earlier work on computational equivalence classes called *dwarves* [5] defined as "a pattern of communication and computation common across a set of applications" [2]. We aim to use exploratory visualization with hive plots to select features for a purely complex networks-based approach to anomaly detection in high performance computing.

6. FUTURE WORK

This work represents an initial step towards improving classification for anomaly detection. Specifically, we use hive panels to reveal communication patterns and identify which network properties lead to distinct hive plots for distinct data sets. The next step is to examine whether hive plots for multiple runs of the same code with different run-time parameters are similar. For example, the hive plots of the `cactus` code when run on 64 compute nodes should be visually similar to the hive plots of `cactus` when run on 128 compute nodes. Finally, we must explore whether the network properties identified during this exploratory data visualization step actually improve classification results and the accuracy of anomaly detection.

Our visualization approach may also be improved. While interactivity was not necessary for our immediate classification goal, adding interactivity may improve our understanding of the communication patterns for each type of computational dwarf. We may also explore including more information from the original data set, such as the number of bytes communicated between nodes and the different types of MPI calls made between nodes.

7. CONCLUSIONS

High performance computing environments are overlooked by existing anomaly detection systems, but regularities in the patterns of messages exchanged between compute nodes make them a suitable target for such systems. These patterns are captured by multiple-edge directed graphs, and ensembles of statistical properties from the field of complex network theory can distinguish graphs that otherwise appear similar using traditional approaches.

Selecting the properties most relevant for machine learning models results in faster, more general anomaly detectors. Exploratory visualization can be a powerful tool for such feature selection. However, traditional force-directed network layout algorithms suffer from lack of reproducibility and an inability to compare across networks or their statistical properties. We use a recent network visualization technique called hive plots which address these weaknesses by modifying parallel coordinate plots to use radial axes with special rules for assigning nodes to axes and the position of nodes along each axis. Hive panels are formed using small multiples of many hive plots for visualizing different network properties and datasets.

Our initial results are promising. Using a hive panel we are able to determine which network properties generate distinct visual signatures in a repeatable and comparable way. We also gain more intuition for the communication patterns exhibited by different scientific computing applications than provided by traditional CCDF plots. We find that different codes with certain topological similarities, while having visually similar hive plots for certain properties, are consistently distinguished by multiple other properties. We also find that, in agreement with our hypothesis, certain properties fail to visually distinguish different codes and are unlikely to contribute significantly to an accurate classifier.

We plan to extend this work by examining hive plots for the same code across different numbers of compute nodes and observing if the resulting visualizations remain similar. While this work focused primarily on the implementation and evaluation of hive plots for visualizing the communication patterns resulting from distributed memory computations, we plan to design and evaluate a new anomaly detection system modified from previous work [13, 14] and will determine if the features suggested by the hive plots do indeed lead to more accurate anomaly detection.

8. ACKNOWLEDGMENTS

Thanks to Scott Campbell and David Skinner for capturing IPM data at NERSC. This research was supported in part by the Director, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy, under contract number DE-AC02-05CH11231. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under contract number DE-AC02-05CH11231. The views and conclusions contained in this document are those of the authors and not necessarily those of its sponsors.

9. REFERENCES

[1] A. Aris and B. Shneiderman. Designing Semantic Substrates for Visual Network Exploration. *Information Visualization*, 6(4):281–300, 2007.

[2] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, and K. Yelick. The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, University of California, Berkeley, 2006.

[3] S. Axelsson. Visualization for Intrusion Detection: Hooking the Worm. In *Proceedings of the 8th European Symposium on Research in Computer Security*, pages 309–325, 2003.

[4] J. Borrill, J. Carter, L. Oliker, D. Skinner, and R. Biswas. Integrated Performance Monitoring of a Cosmology Application on Leading HEC Platforms. In *Proceedings of the 2005 International Conference on Parallel Processing*, pages 119–128, 2005.

[5] P. Colella. Defining Software Requirements for Scientific Computing. Technical report, DARPA High Productivity Computing Systems, 2004.

[6] G. Conti. *Security Data Visualization: Graphical Techniques for Network Analysis.* No Starch Press, 2007.

[7] G. Csardi and T. Nepusz. The igraph Software Package for Complex Network Research. *InterJournal Complex Systems*, 2006.

[8] M. Krzywinski, I. Birol, S. J. M. Jones, and M. A. Marra. Hive Plots – Rational Approach to Visualizing Networks. *Briefings in Bioinformatics*, 2011.

[9] P. Laskov, K. Rieck, C. Schäfer, and K.-R. Müller. Visualization of Anomaly Detection Using Prediction Sensitivity. In *Proceedings of Sicherheit 2005*, pages 197–208, 2005.

[10] P. Mahadevan, D. Krioukov, M. Fomenkov, B. Huffaker, X. Dimitropoulos, K. Claffy, and A. Vahdat. Lessons from Three Views of the Internet Topology. Technical report, Coorperative Association for Internet Data Analysis, 2005.

[11] E. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, 2nd edition, 2001.

[12] E. J. Wegman. Hyperdimensional Data Analysis Using Parallel Coordinates. *Journal of the American Statistical Association*, 85(411):664–675, 1990.

[13] S. Whalen, S. Engle, S. Peisert, and M. Bishop. Network-Theoretic Classification of Parallel Computation Patterns. *International Journal of High Performance Computing Applications*, 26(2):159–169, 2012.

[14] S. Whalen, S. Peisert, and M. Bishop. Network-Theoretic Classification of Parallel Computation Patterns. In *Proceedings of the 1st International Workshop on Characterizing Applications for Heterogeneous Exascale Systems*, 2011.

[15] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis.* Springer New York, 2009.

[16] L. Wilkinson. *The Grammar of Graphics.* Springer, 2nd edition, 2005.

[17] J. Xin, J. E. Dickerson, and J. A. Dickerson. Fuzzy Feature Extraction and Visualization for Intrusion Detection. In *Proceedings of the 12th IEEE International Conference on Fuzzy Systems*, pages 1249–1254, 2003.

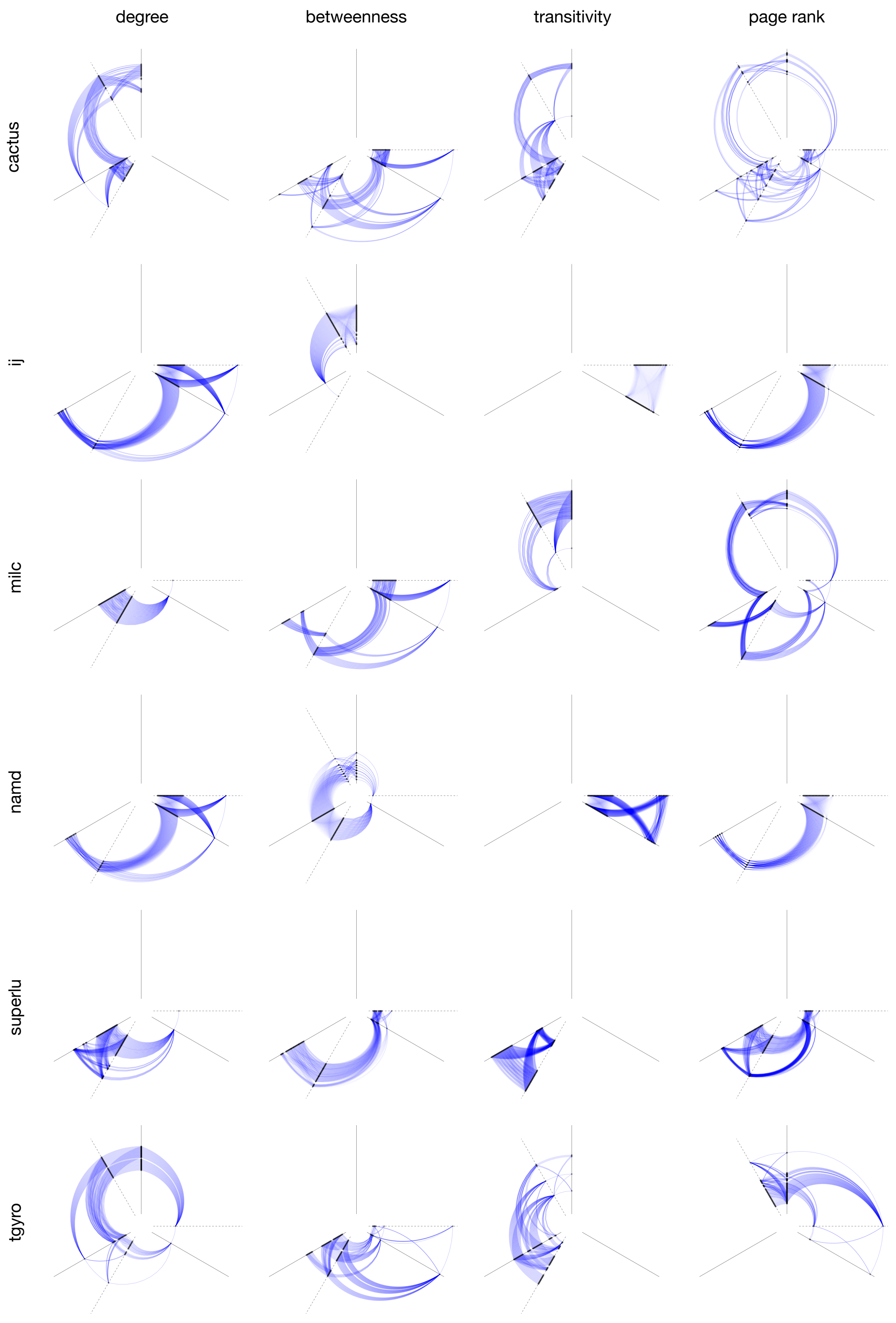


Figure 4: A hive panel showing total degree, undirected betweenness, transitivity, and page rank for each dataset using linear interpolation plus a repelling scheme for node positions and variable edge alpha values.