Investigating "Rankability" Using Distributed Graph Analytic Frameworks

COLLEGE of CHARLESTON
DEPARTMENTS OF COMPUTER SCIENCE AND MATHEMATICS

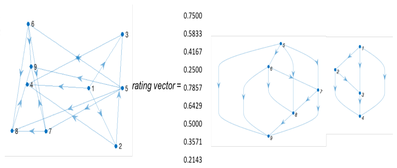
John Cobb, Austin Hunt, Amy Langville, and Paul Anderson

Departments of Mathematics and Computer Science, College of Charleston, Charleston SC

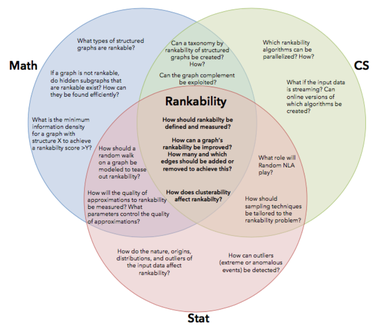
COLLEGE of CHARLESTON FOUNDED 1770

Introduction and Objective

Ranking is an essential data science task that is embedded in nearly every part of translating computational and algorithmic results into a form a human can use. Its applications include cybersecurity, web searches, machine learning, bioinformatics, and business decisions among others.



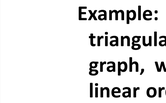
Rankability refers to the ability of a dataset to produce a meaningful ranking. While thousands of ranking algorithms have been developed, during the pursuit many foundational issues have been overlooked such as: Can this ranking be trusted? Are parts of the ranking too similar to be disambiguated and possibly meaningless? How can rankability be quantified? Can rankable subgraphs be identified? At what point is a dynamic, time-evolving graph rankable?



Our research focuses on developing lightweight and scalable measures of rankability using inherent graph structures using Apache Spark. Potentially useful statistics of a graph include the density of the adjacency matrix, the number of strongly connected components, the extent to which the adjacency matrix can be reordered to upper triangular form, the distribution of k-cycles, and the mean recurrence time of a random walk.

Matrix Structures and Rankability

Density



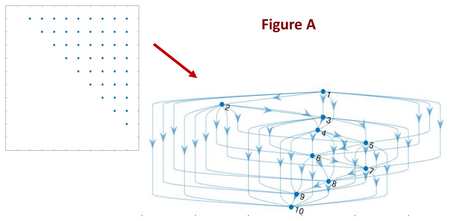
The density of a adjacency matrix could indicate the *potential* for rankability as a denser graph contains more information to support a ranking, although its link to rankability is less clear when these edges are not unidirectional.

Triangular Weightedness and Reordering

The ability of a dataset to be linearly ordered can be gauged via the observance of certain unique properties within the set's adjacency matrix, namely the weightedness of its elements toward the upper or lower triangle and, more indirectly, its density. Note that while the weightedness of the matrix mass toward the upper or lower triangle can be directly used as a metric for how rankable the corresponding data is, the *density* of the matrix is positively correlated with only the *potential* for a graph's rankability. The figures below will help to explain these correlations.

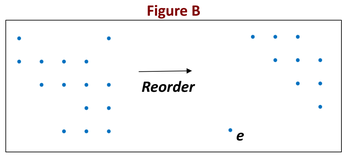
Figure A demonstrates that the data represented in the upper triangular adjacency matrix is very rankable, in that the edges in the corresponding directed graph are unidirectional (all downward), and thus closer to a one-dimensional ordering.

Figure A



Intuitively, the data represented by *any* triangular adjacency matrix exhibits a high degree of rankability because it provides both a completeness of information about pairwise comparisons (a completeness referred to as the matrix's density) and a nearly-linear graph structure. Now, even if a matrix is not perfectly triangular, if its rows can be reordered into this general structure, the rankability of the data can be approximated via the number and location of nonzero elements that prevent the formation of a perfect triangular structure (Figure B).

Figure B



Upon reordering into general upper triangular form, it can be seen that one edge (e) prevents a fully linear ordering of the dataset. In fact, the nonzero element e in this adjacency matrix is evidence of a k-cycle in the graph – that is, a cycle of length k from a node to itself.

Note the following correlations:

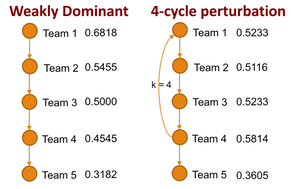
1) **As the number of nonzero elements preventing perfect triangular structure, or *the number of violations,* increases, the data's rankability decreases. (More violations implies more k-cycles, and more k-cycles imply that the dataset is more difficult to reduce to a single linear dimension)**
2) **As any nonzero element preventing the adjacency matrix's perfect triangular structure moves farther away from the mass of the matrix, the data's rankability decreases.**
   Example: edge e in Figure B is in the very bottom left of the *nearly* upper triangular adjacency matrix, meaning there is a cycle of maximum length 5 in the graph, which generally implies low rankability since cycles prevent consistent linear ordering of data within them, and cycles of greater length imply a greater degree of inconsistencies.

Measuring Graph Structure

K-cycles

A cycle of length *k* from a node to itself is called a *k*-cycle. The longer the dominance chain that loops back onto itself, the worse the inconsistency.

Right: Rating vectors reveals the impact of a 4 cycle on an otherwise rankable weakly dominant graph.



Graph rankability could be measured from the histogram of the counts of *k*-cycles for all values of *k* for all nodes in the graph. Identification of *k*-cycles is NP-hard, and thus a very expensive measure of rankability. The *k*-cycle measure serves as a standard against which the other less expensive rankability measurements are created.

Mean Recurrence Times

Mean Recurrence Time (MRT) for a node is the average number of steps, upon leaving node *i*, it takes to return to node *i*. It is one of many statistics that is derived from a random walk. The MRT can be thought of as approximating the *k*-cycle histogram.

Right: Color-mapped MRT distribution of a graph alongside histogram.

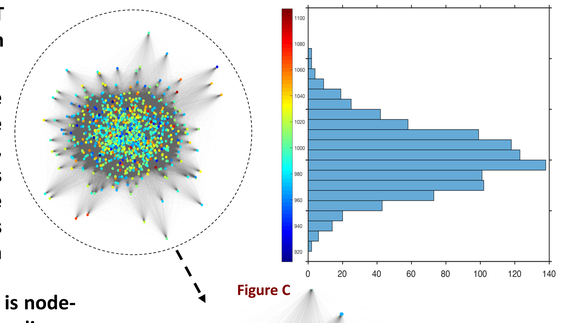
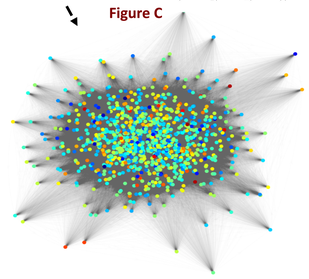


Figure C



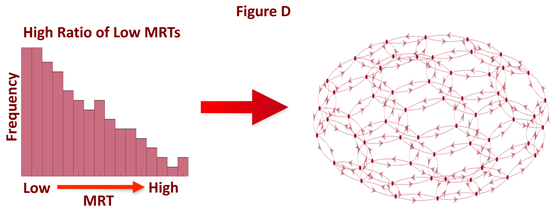
Mean Recurrence Time (MRT) for a node is the average number of steps, upon leaving node *i*, it takes to return to node *i*. It is one of many statistics that is derived from a random walk.

Since mean recurrence time is node-specific and developing a linear ranking is dependent upon pairwise comparisons (i.e. edges) it can be difficult to draw any conclusions about a dataset's rankability on a global scale without understanding the overall distribution of the MRTs among the data.

Interpreting MRT Distributions

In order to draw a connection between these distributions and the rankability of a dataset, it's important to note the correlation between MRT and k-cycles in the graph. Since MRT is defined as the average number of steps it takes to return to node i upon leaving it, then a low MRT for node i (ex: MRT of node i = 2 or 3 in a graph of 100 nodes) suggests that node i is contained in more small k-cycles than in large ones. On the other hand, if the MRT of node i is larger (ex: MRT of node i = 95 in same 100-node graph), it is contained in a higher number of large cycles (approximately 95 edges in length). Now, if the global distribution of MRTs is weighted toward a lower value, this suggests that the graph contains a greater proportion of small k-cycles (see Figure D).

Figure D



On the other hand, if the distribution is more weighted toward higher MRTs, then there are more nodes contained within large k-cycles. On a local level, if node i carries a high MRT, this might suggest that the neighborhood of node i is comparatively *more linear,* and thus *more rankable*, than it would be if node i had a low MRT. However, as mentioned in the previous sections, high MRTs are evidence of long dominance cycles, which of course have poor implications for the rankability of any dataset. This raises quite a rich question which has yet to be definitely answered: does a greater number of small k-cycles have a stronger negative correlation with a dataset's rankability than a small, fixed number of large k-cycles?

Improving Rankability via Sensitivity Analysis

By observing which edges and nodes impact the ranking most (impact can be quantified by Euclidean distance between initial and final rating vectors upon removal of sensitive node/edge), a graph's rankability may be improved simply by gathering more pairwise comparison information surrounding the highly sensitive element(s).

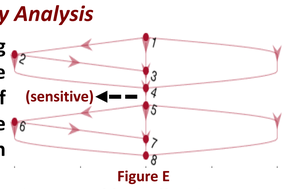


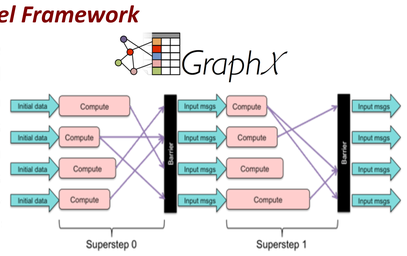
Figure E

For example, the rankability of the graph in Figure E could be improved by collecting more comparison data between the two clusters (i.e. cluster [1,4] and cluster [5,8]) to augment the single edge between nodes 4 and 5, because this single intercluster link has the greatest impact on the global ranking.

Spark Implementation

Graph Parallel Framework

Graphx, like most graph-parallel systems, use the bulk synchronous execution model (*Right*), in which all vertex programs run concurrently in a sequence of supersteps operating on the adjacent vertex-program state or on messages from the previous super-step. The abstraction is sufficiently expressive to support a wide range of algorithms on large clusters.



Winning percentage

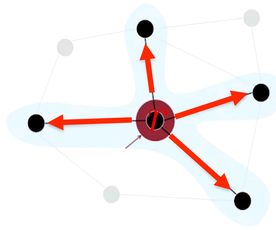
$$r = \frac{1 + w_i}{2 + t_i}$$

Strength of schedule correction

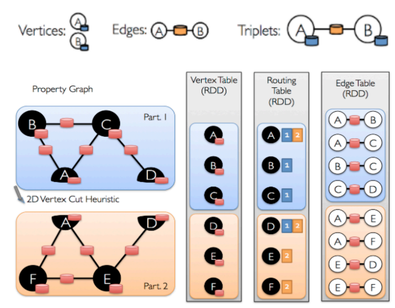
$$w_i^{cor} = \frac{w_i + l_i}{2} + \sum_{j \in O_i} r_j$$

Colley iterative scheme

$$r_i = \frac{1 + (w_i + l_i)/2 + \sum_{j \in O_i} r_j}{2 + t_i}$$



***Top*: A schematic demonstrating the vertex-central Colley ranking algorithm. The vertex-program for a vertex *v* begins by receiving messages that were send along edges from the previous iteration and computing the sum pairwise to complete the part of Colley in the red box. Then, the value is funneled into formula to get a point estimate for *v*.. This whole process is reiterated with the new ranking found until convergence.**



***Left*: Visualizations of GraphX datatypes (*top*) and partitioning (*bottom*).**

The same restrictions that enable graph-parallel systems to our perform contemporary data-parallel systems when applied to graph computation also limit their applicability to many of the operations found in a typical graph analytics pipeline. In a sense, linear algebra would need to be rewritten with respect to these partitioning schemes to support scalability.

Creating a Rankability Toolbox

```
class rankGraph() {
  // imports
  def buildTennisGraph(filepath: String): Graph[String, Double] = {} //builds graph based off tennis database
  def colley(graph: Graph[String, Double]): Graph[String, Double], Double] =  {} //runs colley until convergence, returns colley ratings as vertex attribute
  def removeNode(identifier: Long, graph: Graph[String,Double]): Graph[String,Double] = {} //removes a given node from graph
  def sampleGraph(): Graph[String, Double] = {} //returns a graph with known attributes to run tests on
  def singleNodeSensitivity(node: VertexId, graph: Graph[String, Double], initRank: Graph[(String, Double), Double] ): Double = {} //finds sensitivity of single node
  def sampleEdges(graph: Graph[String, Double], fraction: Double): Graph[String, Double] = {} //sample given percentage of edges from graph
  def sampleNodes(graph: Graph[String, Double], fraction: Double): Graph[String, Double] = {} //sample given percentage of nodes from graph
  def replaceNode(inputGraph: Graph[(String, Double), Double], newNode: (VertexId,(String, Double)) ): Graph[(String, Double), Double] = {} // replaces given node from input graph
  def buildGraphFromMatrix(matrix: Array[Array[Int]]): Graph[String,Int] = {} // creates a graph object from an adjacency matrix
  def toGexf(graph: Graph[String, Double]) = {} // creates a .gexf file based on graph to be read by popular graph visualization software
}//end class
```

***Above*: Method signatures of rankGraph class used to explore rankability questions on large scale graphs.**

The colley GraphX implementation follows the message passing pattern described above, while methods such as single node sensitivity uses optimized slice methods to generate a subgraph excluding the input nodes to pass into the colley method.

Conclusions and Future Directions

- **Although a pure count of *k*-cycles remains the most promising metric, further investigation is needed before trusting "approximations" such as Mean Recurrence Time.**
- **If mean recurrence time could be more faithfully linked to global rankability, then sensitivity analysis could be redeveloped to use MRTs rather than choosing an arbitrary ranking method. For example, removing a sample of the graph and observing the ratio of MRTs which then go to infinity could provide global rankability information.**
- **Although our measures follow an intuitive path, a more formal definition of ranking would help standardize otherwise wandering conceptions of rankability.**
- **A graph's Laplacian *L* may provide interesting insight into cluster: intercluster link ratio. The dimension of *L* is the number of connected components while the second smallest eigenvalue is the Fiedler value (connectivity).**
- **Although we have left clustering methods relatively unexplored, their results or optimization routes are likely to hold information about graph structure. In particular, David Gleich at Purdue has developed a clustering method that clusters by motif- or small graph structures. In our case, it might be helpful to cluster by *k*-cycles of varying length.**

Acknowledgements

Acknowledgment is made to the Dean's Summer Fund for financial support of this research. Further thanks to Dr. Tim Chartier of Davidson College, the IGARDS team, AMPlab at UC Berkeley, and Tresata for the support and materials necessary for the making of this poster.