Similarity and Percolation on Networks

Vinay Bhaip

7177 words (TeXcount)

University of Oxford

A thesis submitted for the degree of

MSc in Mathematical Sciences

Trinity 2024

Abstract

Networks are a common way to understand structural relationships in the real world. One area of key interest is identifying nodes in a network that are similar to each other. Many existing similarity measures, such as the Jaccard Index and SimRank, primarily focus on neighbors and random walks from a node. This paper examines a new path-based metric known as percolation similarity, which measures the strength of connectivity between nodes as edges are randomly removed with some probability. To do so, this paper simulates the metric on different structured graphs and compares the results to popular existing metrics. Percolation similarity performs similarly to a metric based on the resistance distance, identifying nodes with high levels of information-flow between them. These path-based metrics find different communities in networks than those found by the random-walk and neighbors-based approaches by identifying a different feature in nodes instead: those that are more likely to be in the "core" of a network. One unique advantage of percolation similarity is its interpretability; the similarity value between any two nodes has a direct meaning in the context of networks. This paper opens the door to using percolation similarity for identifying core-like nodes in a network with a meaningful explanation as to what the metric's value is.

Contents

Chapter 1

Introduction

A common way to model objects and relationships in the real world is via networks. Formally, a network is just a graph $G = (V, E)$, which is a set of vertices and pairs of vertices known as edges. Edges can either be directed or undirected, depending on whether the order of the vertices in the edge (u, v) matters; in this paper, I will only focus on undirected graphs. I also use the terms networks and graphs interchangeably as well as vertices and nodes. Networks can and have been used to model different behaviors and phenomena such as social dynamics, citations of papers, and familial relationships.

1.1 Types of Networks

For a network with $|V|$ nodes, there are $\frac{|V|*(|V|-1)}{2}$ possible edges, which means there are $2^{\frac{|V|*(|V|-1)}{2}}$ resulting networks behind any given set of nodes. Networks with the number of edges $|E|$ being close to $\frac{|V|*(|V|-1)}{2}$ are categorized as dense and those with $|E| << \frac{|V|*(|V|-1)}{2}$ are categorized as sparse. A network with $|E| = \frac{|V|*(|V|-1)}{2}$ is called a complete graph on V, since there are no possible edges that are missing. This is also called a $|V|$-clique.

One interesting graph to examine is one that originates from a stochastic block model. The stochastic block model defines a set of groups of nodes and fixed probabilities that determine the likelihood of nodes being connected depending on the group that the node is in. If $f : V \to K$ is the node to group mapping, where K is the set of groups a node can be in, and $g : K \times K \to [0, 1]$ is the probability that two nodes in two groups are connected, then for nodes u, v, the probability that they are connected $\mathbb{P}\{(u, v) \in E\} = g(f(u), f(v))$. The stochastic block model can be represented as a $|K| \times |K|$ matrix M where $\mathbb{P}\{(u, v) \in E\} = M[f(u)][f(v))]$. Note, this matrix should be symmetric for undirected graphs.

A specific version of the stochastic block model is known as core-periphery models. In this model, we have a highly connected core, which is connected to other nodes outside of the core that themselves are sparsely connected. Using a stochastic block model with a matrix representation $M = \left(\begin{smallmatrix} a & b \\ b & c \end{smallmatrix}\right)$, we would have $a > b >> c$. Rombach et al. (2014) point out that this core-periphery model appears in real-life in numerous cases. This is because many graphs have "hubs," nodes with high degree, that connect to many otherwise unconnected nodes. One example of such a network that Rombach brings up is the World Wide Web, where there is a large tail of sites which are not connected to that many other sites, but some major sites with high degrees [14]. The presence of a core in many graphs makes it a useful problem to try to identify them.

1.2 Topics Related to Networks

One major problem in networks is community detection. The formal classification of this problem is to partition a set of vertices such that each subset of the partition has nodes that are similar to each other. For example, in a network with two cliques joined by one single edge between one node in each of the two cliques, it seems intuitive that the two communities or clusters are the cliques. In real-life networks, it is rarely this clear; for example, how does one decide how many communities to even subdivide till? As a rule of thumb, a node should be more connected to nodes within its community than to other nodes outside of its community.

There are many modern algorithms to identify communities in a network. Some methods successively subdivide nodes into more and more communities to greedily maximize a measure called modularity, which gives an understanding for how well separated the nodes are in each community. Other methods, such as the popular Louvain Method proposed by Blondel et al. (2008), build up communities considering each node as its own community and then successively joining nodes together until some heuristic for the community measure does not improve [1].

At the core of the question of community detection is a question of how similar nodes are. Similarity is a vague metric by definition, but Fouss et al. (2016) illuminate the concept more: nodes that share common features should be considered more similar, and when two nodes are the same, they should achieve the highest similarity [6].

There are three different ways to view similarity, as Hanneman and Riddle (2005) identify: structural equivalence, automorphic equivalence, and regular equivalence. Structural equivalence of nodes occurs when two nodes play the same role; swapping both nodes gives an indistinguishable graph. A less strict version of this is automorphic equivalence, which occurs when swapping both nodes leads to an isomorphic graph [8]. For example, consider a perfect binary tree. The nodes on any given level demonstrate automorphic equivalence because the graph structure is the same up to a relabeling of nodes. However, only the leaves of a binary tree that share the same immediate parent are structurally equivalent. Lastly, two nodes are regularly equivalent if the nodes they are connected are regularly equivalent to each other. As a rough ordering for the strictness of each definition, regular equivalence is less strict than automorphic equivalence which is less strict than structural equivalence [8].

One point to note is that equivalence is not the same as proximity in networks; equivalence captures structural patterns of nodes and/or the roles they play in a network whereas proximity is the path length distance between nodes. Similarity measures that attempt to approximate equivalence may use metrics that are tied to proximity as a heuristic, but the goal is to understand equivalence.

As Fouss et al. point out, knowing whether nodes are similar can not only reveal if they are in the same community, but also can be used for link prediction. If two nodes are highly similar, but there is no edge between the two, perhaps the collection of that edge was a false-negative. Moreover, if two nodes are highly similar, then maybe the neighbors of one node should be connected to the other node and vice versa [6]. This method of link-prediction should be taken with caution though, since, as observed above, the presence of an edge between nodes is not fully correlated to similarity.

Lastly, one concept related to networks is percolation. Two nodes u, v are said to percolate ($u \leftrightarrow v$) if there exists a path between the two nodes. Percolation is directly a measure of whether two nodes lie in the same connected component. Percolation between two nodes is a binary value, but it can be viewed as a probability when considering how likely two nodes percolate when the network structure is generated via a random model. Several different graph generation models loosely revolve around percolation, the most notable being the Potts Random Cluster Model proposed by Fortuin and Kasteleyn (1972) [5]. This model generates a graph based on parameters relating to percolation, and with a specific configuration, it devolves into the Ising Model, a famous model explaining magnetism. This is to say that percolation is at the core of many real-life networks.

1.3 Outline of Work

The goal of this work is to examine a new similarity measure called percolation similarity. In the second chapter, I introduce examples of existing, popular similarity measures and desirable properties of them. In the third chapter, I formalize percolation similarity and describe practical ways to measure it computationally. In the fourth chapter, I lay out the experiments of percolation similarity and other metrics on various graphs and analyze how percolation similarity performs. In the final chapter, I summarize the results with future steps.

Chapter 2

Similarity Measures

As the introduction points out, similarity is a vague metric, which can be taken to mean different things in different contexts. Many similarity measures try to approximate structural equivalence, and do so based on features of the nodes. Fouss et al. quantifies some of the features that similar nodes might align on: proximity (how close two nodes are using the path length as a distance), connection of the two nodes by indirect paths, influence of the network in similar ways, and sharing a common substructure. Many existing similarity measures use one or more of these features to motivate their metric. Fouss et al. list out four conditions (labels are my own) for the similarity of two nodes $s(u,v)$ [6]:

- **Symmetry:** $s(u,v) = s(v,u)$
- **Positivity**: $s(u,v) > 0$
- **Ordering**: If v_1 is more similar to u than v_2 is to u, then $s(v_1,u) > s(v_2,u)$
- **Maximum Self-Similarity**: For any nodes u, v, $s(u,v) \leq s(u,u)$

2.1 Examples of Similarity Measures

2.1.1 Local Similarity Measures

One simple similarity measure is the number of shared common neighbors. Fouss et al. says this is useful in the case of bibliographic counting, in the sense that two papers are similar if they cite many of the same papers [6]. One issue with this metric is that it arbitrarily sets a cut-off of length 1 for the depth of shared neighbors. Consider *k-neighbors* to be the set of neighbors that are of shortest distance k away from a given node. The shared common neighbors similarity metric only looks at *1-neighbors* but there is no reason to choose this; maybe two papers share a lot of papers that are of distance 2 away, but this metric ignores that information. This is also not a normalized metric; in dense graphs, the number of shared common neighbors would be higher than that of sparse graphs, so comparing this similarity metric across different networks is not very useful.

A more widely used metric in the literature is the Jaccard Index. If $N : V \to \mathcal{P}(V)$ is the neighbors of a node, then the Jaccard Index $J : V \times V \to \mathbb{R}$ is given by $J(u, v) = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$. In the extreme cases, if two nodes share the same neighbors, then this achieves a maximal value of 1 and if two nodes share no neighbors, then this achieves a minimal value of 0. This is essentially a normalized version of the shared common neighbors metric. The Jaccard index is used in many applications of network theory and has many extensions that spin off it centered around the same idea, such as those Costa (2021) describes [3].

The benefit of the shared common neighbors and Jaccard Index metrics discussed so far is that they are computationally efficient because they fall under a class of similarity measures known as local measures. To compute the similarity of two nodes with a local measure, there is no involvement of nodes outside of a local area, so larger information in the graph structure is ignored.

2.1.2 Global Similarity Measures

Alternatively, global similarity measures use the larger graph for seeing how similar any two nodes are. The simplest global similarity measure is the shortest path distance. To compute the shortest path distance between any two nodes, one must look at all the simple paths between the two nodes, which requires knowing the global graph structure. While it is arguably the easiest to understand and thus has high explainability, it often falls short. As Fouss et al. point out, this metric does not account for the degree of connectivity between nodes [6]. If two nodes in a graph have a single path of length 2 between them, whereas two separate nodes have many paths of length 2 between them, it might be reasonable that the latter group of nodes should be classified as more similar.

Many of the more complicated global similarity measures use the graph Laplacian, defined as the degree matrix minus the adjacency matrix. One example of this is effective resistance distance. The resistance distance treats the graph as an electrical network with edges having a unit resistance, and the distance between any two nodes being the effective resistance between the two points. The explicit computation for the resistance distance uses the graph Laplacian, and assigns a distance for any two points.

As Fouss et al. point out, there is another interpretation of the resistance distance; the resistance distance is proportional to the expected number of random walk traversals that would start at one node and end at the other, which naturally is proportional to the average commute time between the two nodes [6].

Effective resistance distance gives a distance measure on the nodes, but does not explicitly assign a similarity value. One way to remedy this is to take the inverse of the distance, which would assign a similarity of $+\infty$ to a node being compared to itself and smaller similarities to nodes that have larger distances between them.

One more popular global similarity measure is SimRank. SimRank measures how many other nodes reference the nodes being compared. The SimRank for any two nodes is proportional to the sum of the SimRank of each neighbor of the first node coupled with each neighbor of the second node. This leads to a recursive equation that puts nodes with similar reference patterns as having a high similarity. Fouss et al. explain that SimRank is especially useful in cases when measuring whether two nodes are co-cited by papers that themselves are very similar [6]. Jeh and Widom (2002) explain that they designed SimRank based off the notable PageRank algorithm from Google that ranks websites according to how many other websites refer to it [9].

At the start of the chapter, I identified conditions for the similarity of two nodes. There are a few additional desirable properties I would argue for what makes an effective similarity measure. First, we want a similarity measure to be computationally efficient; if there are large space/time complexity requirements, then it is not useful in practice. Second, we want a similarity measure to be explainable. The value a similarity measure outputs should be explainable as to what it means rather than being an abstract quantity. Lastly, similarity values should be normalized to be in the range $[0, 1]$; this allows us to compare different similarity metrics against each other.

Chapter 3

Percolation Similarity

Many local similarity measures miss out on a richness of information regarding the role that nodes play in the larger graph and relating that to their similarity. The popular Jaccard Index, while it gives a simple and fast heuristic for how close two nodes are to each other, does not capture more complex relationships between the nodes. Global measures like SimRank recursively look at neighbors to compute its value, which is akin to random walks. Another key feature of graphs that holds valuable information on the similarity of two nodes is the existence of paths between them.

The effective resistance distance does this well by treating the network as an electrical network, but arguably this translation to an arbitrary context is not as explainable as one would desire. For the similarity of any two nodes with this measure, it would be the reciprocal of the distance treating the graph as an electrical network, which does not give an intuitive understanding for the meaning of the value.

On the other hand, the shortest path between two nodes is very explainable, but often is not accurate and disregards information regarding the other paths between the two nodes. What is needed is a similarity measure that both works as well as the effective resistance distance by using all the path information while maintaining explainability like the shortest path metric.

3.1 Properties of Percolation Similarity

Lambiotte (preprint) introduces a new measure called percolation similarity. Recall that percolation is a measure for whether two nodes are connected. Percolation similarity is defined as follows: remove existing edges in the graph with some probability p and then measure whether the two nodes are still connected [4]. Assume that the two nodes to begin with are connected. At the extrema, with $p = 0$, the graph remains unperturbed, meaning the probability that the nodes are connected is 1. At $p = 1$, there are no edges in the graph, and the probability that the nodes are connected is 0.

Fix two nodes u, v in a fixed graph G. Assume that $f : [0, 1] \to [0, 1]$ takes in the edge-removal probability p and returns the probability that these two nodes are still connected.

Lemma. *f is decreasing.*

We can use a coupling-like argument on the edges of G using a common source of randomness. Consider a graph G and assign a uniform random number in the range [0,1] to each edge. Then consider two probabilities p_1 and p_2 such that $p_1 < p_2$. The perturbed graph G_{p_1} results from removing all the edges in G which have the random number that was assigned to them being less than p_1 and similarly for p_2. This means that any edge in G_{p_2} must be an edge of G_{p_1}. We can then say that $E(G_{p_2}) \subset E(G_{p_1})$. Now consider whether u and v are connected in G_{p_1} and G_{p_2}. Clearly, since the edges of G_{p_2} is a subset of the edges of G_{p_1}, if the nodes percolate in G_{p_2}, they must percolate in G_{p_1}.

Note that G_{p_1} is the graph for $f(p_1)$ and G_{p_2} is the graph for $f(p_2)$, the probability that the two nodes are connected. Thus, we've shown that for $p_1 < p_2$, $f(p_1) > f(p_2)$, which means f is decreasing.

For brevity, we will call the resulting output of f to be the "p-curve," which has the properties of being a decreasing function on $[0, 1]$ outputting a probability.

The next question is which value of p should we use. There is no clear answer to this. For sparse graphs, using a small p might be desirable because removing just one edge might remove connectivity of the nodes, but that same p might not be ideal for dense graphs because the nodes are highly likely to remain connected and vice versa for large values of p.

3.2 Versions of Percolation Similarity

This paper introduces two measures to capture percolation similarity: percolation similarity by integration and percolation similarity by threshold. Percolation similarity by integration numerically integrates the p-curve across p values from 0 to 1. The idea is to avoid missing out on the rich information that the p-curve provides. If the p-curve has large values even for large values of p, then the area underneath the curve

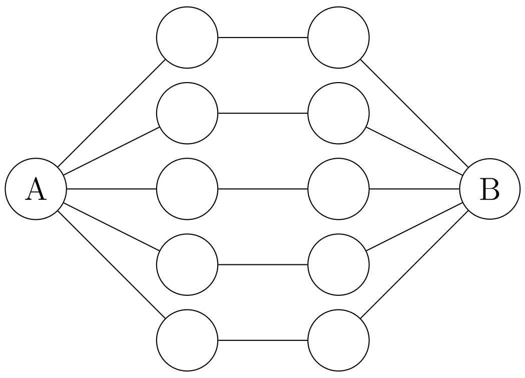

Figure 3.1: A sample network where similarity of the nodes A and B means different things depending on the context of distance or information flow. The Jaccard Index for nodes A and B would be 0 here.

should be large. Peel et al. (2018) describe this integration method in more detail in Equation 7 of their paper for a similar problem they encountered [12]. In practical computation, this looks like just averaging f over equally spaced values $k \in [0, 1]$. The higher the number of samples, the more accurate this approximation would be.

Alternatively, percolation similarity by threshold sets a threshold t and finds the point where the p-curve intersects t, returning the value p such that $f(p) = t$. Since f is strictly decreasing on t, we can say $p = f^{-1}(t)$. This leads to two issues: first, what value of t should we use? This paper uses $t = 0.5$ for the explainability that the metric percolation similarity by threshold is the proportion of edges removed that marks the turning point for when the nodes would be more likely to not percolate than they would be to percolate. Second, while f is strictly decreasing, practically, we would not have oracle access to f. This means that there is some noise to what f is evaluated to be, meaning $f_{observed}$ might not be strictly decreasing. To remedy this, a practical way to determine where $f(p) = t$ is to find $\text{argmin}_p\, |f_{obs}(p) - t|$. This finds the closest value to t for $f_{obs}(p)$.

3.3 Justifications for Percolation Similarity

Percolation similarity captures information flow between the two nodes in the network. Consider the graph in 3.1. Some popular similarity measures, such as the Jaccard index, would give a similarity of zero for A and B. This might make sense in the context of distance, but does not make sense when considering the flow of the network. Any information starting at A would have many paths to take to reach B, even if they are not close to each other. A and B also play very similar roles in the graph, so having a similarity of zero seems misguided. The beauty of percolation similarity lies in the fact that it counts the number of paths as well as incorporating the lengths of these paths. If a path is long, then there are more edges in it that have the ability to be removed and stop percolation.

One additional benefit of percolation similarity is how it can generalize to find clusters. With most similarity measures, there can be a greedy algorithm that groups nodes with similar values for their similarity measure to recursively build up different communities. Percolation similarity allows for a further step; partitioning the graph into the best communities correlates to the min-cut of a graph. Consider a graph of nodes, partitioned into two node sets. Collapse all of the nodes in each partition group to one single node, thereby removing internal edges within the group, but maintaining any edges that go to the node representing the other set in the partition; this forms a multigraph with two nodes. Finding the partition that maximizes percolation similarity of these two nodes is precisely when there are the fewest edges possible between the two nodes, which is the min-cut of the graph. Similar logic extends to partitions of more than just two groups. This provides justification for a) the use of percolation similarity as a similarity measure and b) possibly using percolation similarity to find communities, one of the bigger goals of similarity measures in general.

Chapter 4

Simulations

Lambiotte provides justification for percolation similarity by providing the theoretical underpinnings of it [4]. This paper takes the approach of testing out the two formulations of percolation similarity, by integration and by threshold, on several graphs and examining the results.

4.1 Methodology

The similarity metrics I examined were percolation similarity by integration, percolation similarity by threshold, resistance distance, SimRank, and the Jaccard Index. The Jaccard Index, as a local and more simple measure, provides a baseline for the performance of the other global metrics. Resistance distance and SimRank are the main metrics that should provide understanding for how the percolation similarity metrics relatively perform.

The simulations were done using Python and extensive use of the NetworkX library, as published by Hagberg et al. (2008) [7][1]. The Jaccard Index and SimRank were calculated using NetworkX functions out of the box. The resistance distance also was calculated via NetworkX, but resistance distance by itself does not provide a similarity metric. I normalized the values to be a comparable metric by dividing all the resistance distances of pairs of nodes in a graph by the maximum resistance distance present, and then taking 1 minus that value, putting the range of the metric as [0, 1]. The key point to note with this transformation is that it maintains the ordering of the distances.

To compute the percolation similarity metrics, I compute f_{obs}, the noisy version of the p-curve I discuss in the previous chapter, by sampling f at various p values.

[1]The most relevant code I produced can be found in Appendix A.

Given a p and a graph G, I compute G', the perturbed version of G, by generating a random number in $[0, 1]$ for each edge and removing the edge if it falls below p. Then, for each pair of nodes I calculate whether the nodes still percolate by seeing if there exists a path between them.

For this to be an accurate probability, I repeat this process many times and compute the average over the samples. For percolation similarity by threshold, I then find the value of p for a given pair of nodes that has a probability closest to 0.5, the threshold I decided to use. For percolation similarity by integration, I average all values of f_{obs} for all the p values I tried. For all the percolation similarity simulations that follow I used a sample size of 100 and an interval size for p to be 0.01. This means I computed $f_{obs}(p) = \sum_{i=1}^{100} f_i(p)$ for $p \in [0, 0.01, 0.02, \ldots, 0.99]$, with $f_i(p)$ being the observed sample probability.

I tested these similarity metrics on graphs that capture different structures to see how robust each metric was. I tested on the following graphs:

- A clique
- A connected clique graph (ring of cliques), which is k cliques that are joined together by one edge in each clique to a neighboring clique
- A clique with a single edge missing
- A clique with an "outsider" – an additional node connected by one edge to a node in the clique
- Two types of core-periphery graphs
- Zachary's Karate Club network from Zachary (1977) [15]
- Florentine Family network from Breiger and Pattison (1986) [2]

The primary outputs of the simulations were two heat maps per metric. The first heatmap visualizes the absolute value of the similarity metric for each pair of nodes on a scale of [0,1]. The second heatmap visualizes the relative ranking of the similarity metric of two nodes relative to those of other pairs of nodes. In the case of ties when ranking two values, I take the minimum ranking for both of them, as implemented by the rankdata function in the Python package Scikit-learn from Pedregosa et al. (2011) [11].

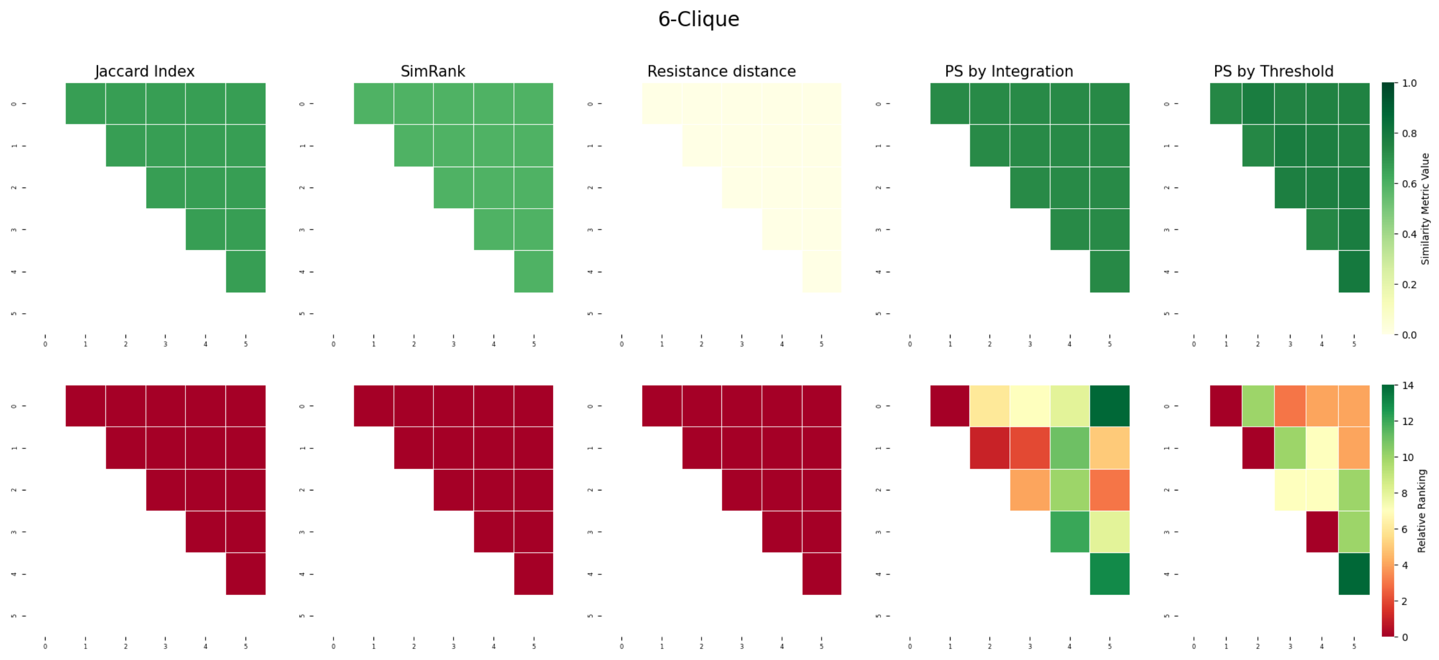

Figure 4.1: Heat maps for a clique with six nodes.

4.2 Results

4.2.1 Clique

For a clique of size 6, all of the metrics in 4.1 show similar values for the absolute value of the similarity metric, which makes sense. The ranked heatmaps are the same as well, but the percolation similarity matrices show seemingly random rankings. The probabilistic simulation nature of percolation similarity lends itself to introducing noise into the measurement.

4.2.2 Connected cliques

For six cliques of size 6, the Jaccard Index, resistance distance, and SimRank metrics result in similar matrices that all rank nodes in the same clique as having similar scores as seen in 4.2. The one node in each clique that connects to other cliques has a lower score to nodes in its clique, but also has a positive similarity for the node it connects to outside of its clique. The percolation similarity heatmaps for both integration and threshold show a similar heatmap, but with noise added. The percolation similarity by integration map appears to have much less noise than the threshold map, which makes sense because the integration metric is using the data

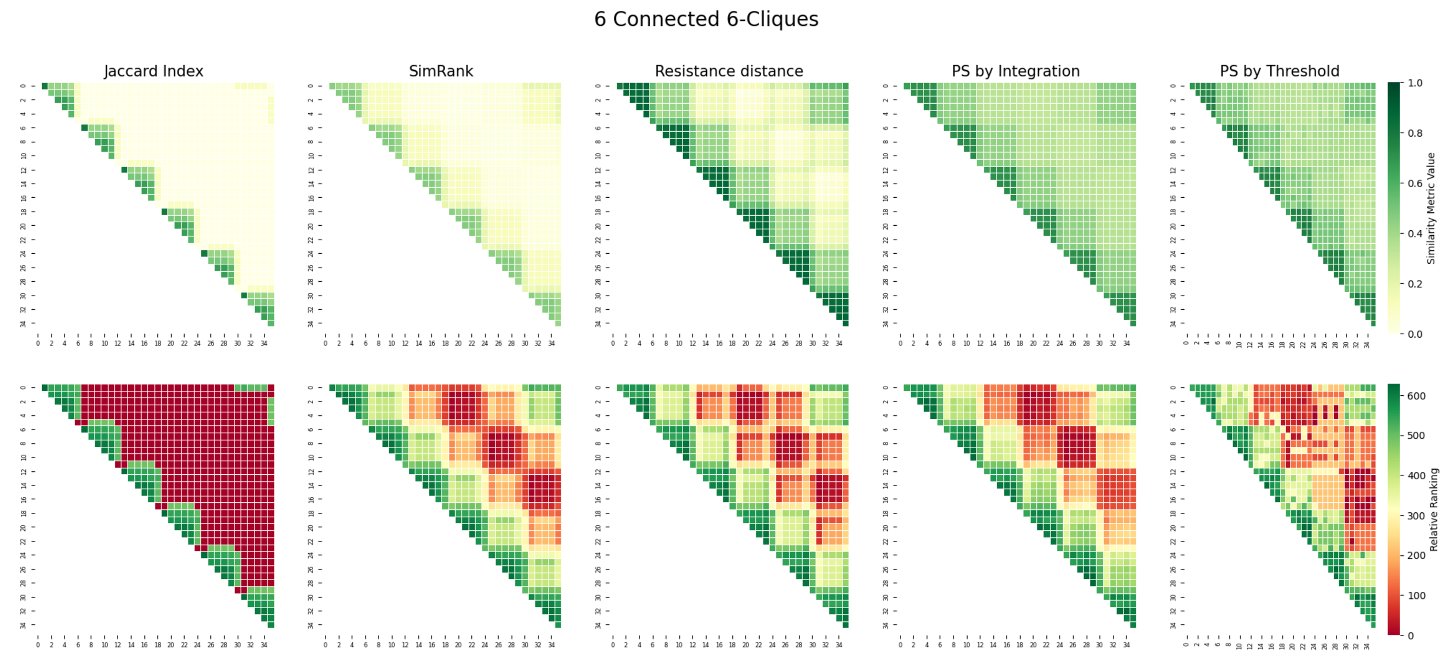

Figure 4.2: Heat maps for a graph with six cliques, each of size six and connected to a neighboring clique with one edge to look like a "ring of cliques." Here, node 5 is connected to node 6, node 11 to node 12, node 17 to node 18, node 23 to node 24, and node 29 to node 30.

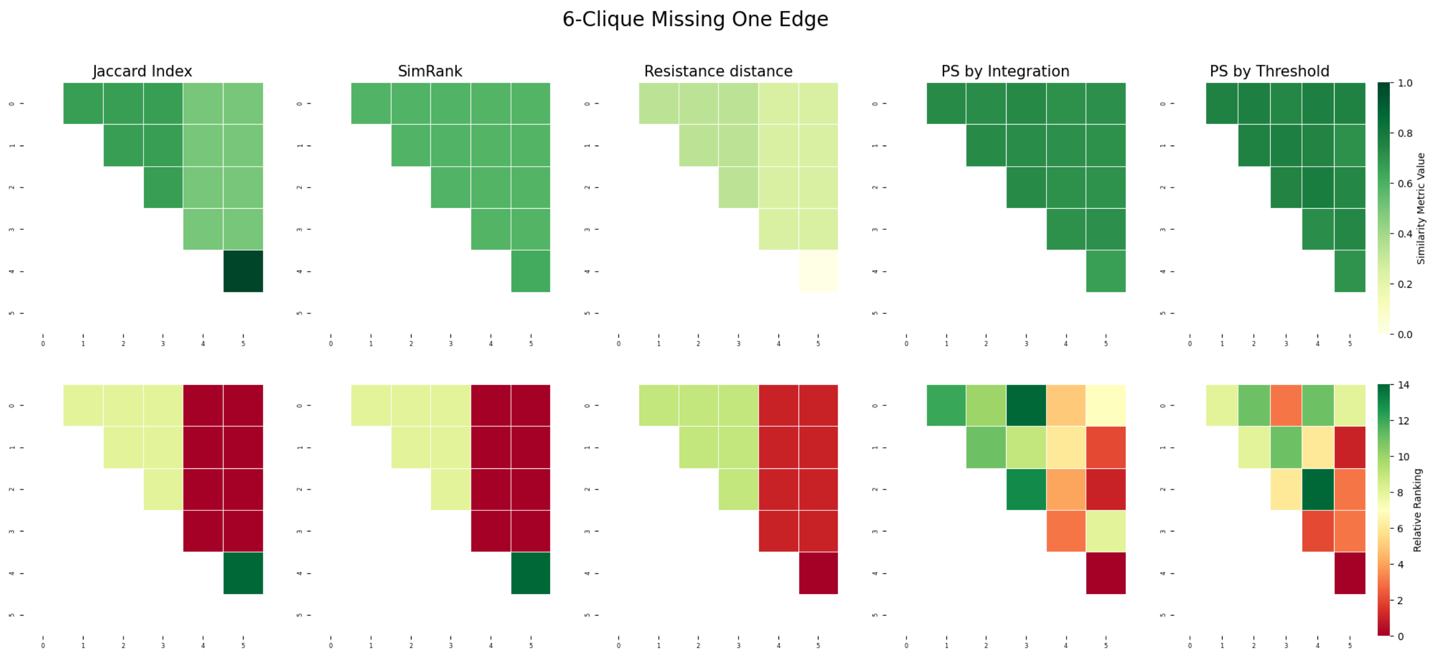

Figure 4.3: Heat maps for a clique with six nodes that is missing one edge between nodes 4 and 5.

from the whole function, thereby reducing sensitivity to outliers. In these highly structured graphs, there is essentially a ground truth for what the heatmaps should look like, and the percolation similarity metrics model it with some noise.

4.2.3 Clique with single edge missing

In this graph, we have a clique with an edge missing between node 4 and node 5. In 4.3, the Jaccard Index and SimRank heat maps produce the exact same ranked heatmaps, ranking nodes 4 and 5 as the most similar, and all the other neighbors with either node 4 or 5 as being the least similar. This is interesting because nodes 4 and 5 have automorphic equivalence and structural equivalence and play the same unique role in the graph, but are the only nodes that do not have a path of length one between them. The resistance distance metric puts these nodes as having the least similarity, reflecting how it captures the distance between nodes. The percolation similarity measurements again are noisy, but it aligns with the resistance distance metric by putting the similarity of nodes 4 and 5 at the bottom.

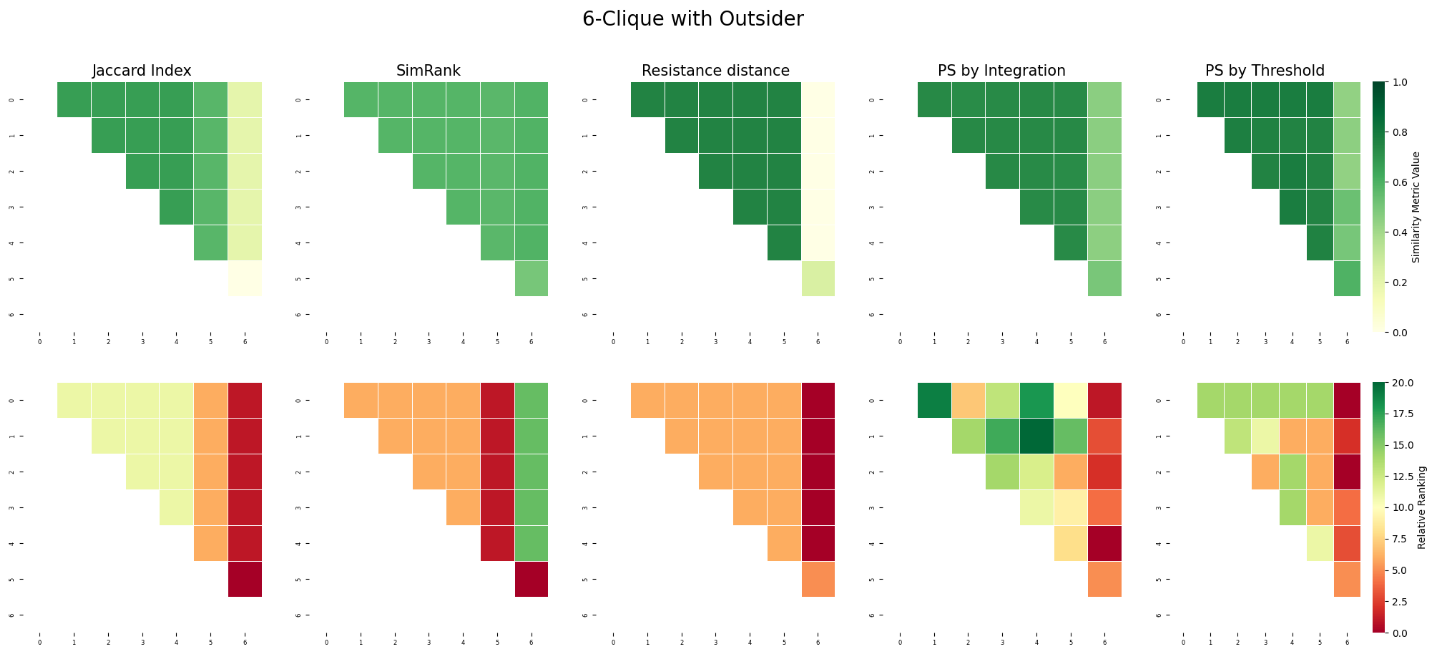

Figure 4.4: Heat maps for a clique with six nodes that has an additional node, node 6, that is connected to only one node in the clique, node 5.

4.2.4 Clique with an "outsider" node

Similar to the previous example, in 4.4, the Jaccard Index and SimRank rank the additional node with its one connection node as having the lowest similarity. The Jaccard Index and SimRank differ on the similarity ranking of the additional node with the rest of the clique though; the Jaccard Index marks it as not similar as well, but SimRank puts it as the most similar. The resistance distance puts nodes within the clique as having the most similarity and the additional node with all of the non-neighboring nodes in the clique as having the least similarity. Both percolation similarity metrics generally capture that the last node is not similar to the other nodes, but beyond that it is hard to discern meaningful information.

The graphs so far use highly structured dense graphs, and the percolation similarity matrix loosely reflects the resistance distance. There is a significant amount of noise in the ranked heat maps for the percolation similarity metrics though, which makes sense because the approximated p-curve graphs might differ slightly. For example, for percolation similarity by threshold, the p that reaches the threshold for 50% of the time the nodes would not percolate would differ slightly, so the noise in sampling would affect the observed value.

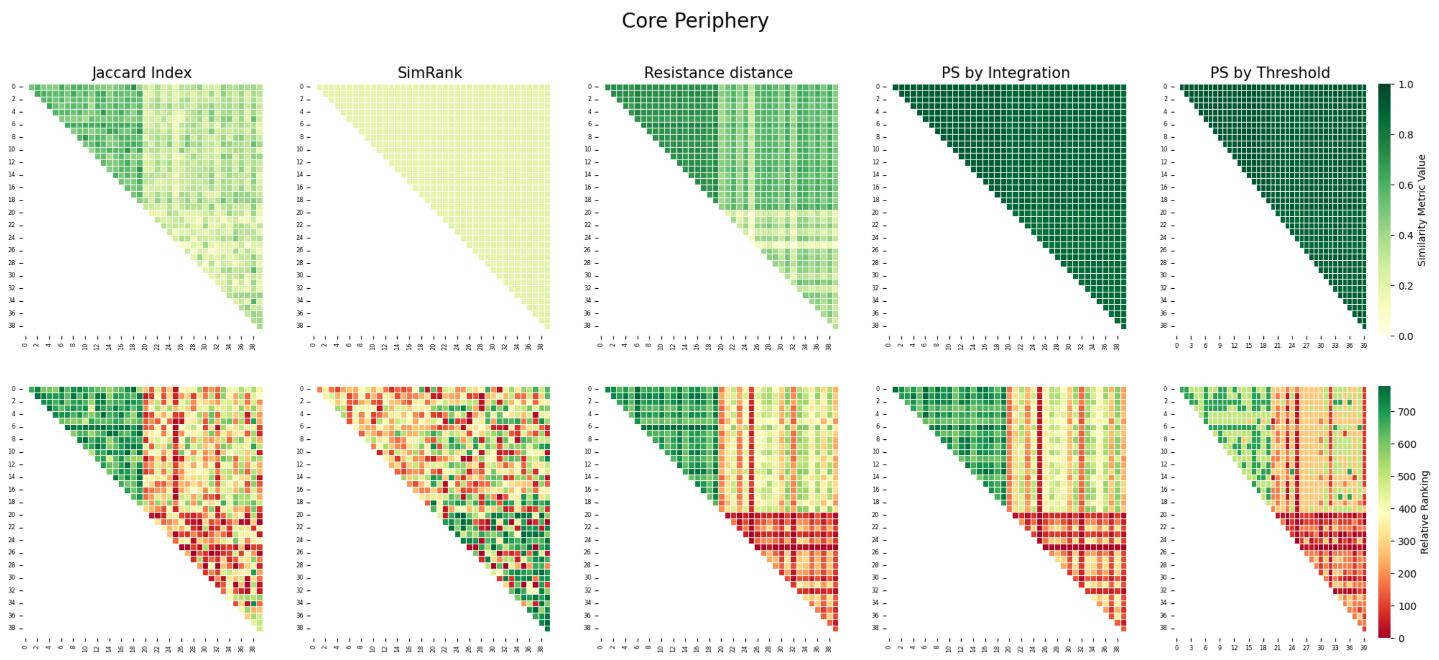

Figure 4.5: Heat maps for a Core Periphery graph with two communities of 20 nodes. The model used $\begin{pmatrix} 0.9 & 0.5 \\ 0.5 & 0.1 \end{pmatrix}$ for the stochastic probabilities of an edge existing.

4.2.5 Core-Periphery

4.2.5.1 2-Level Core-Periphery

The resistance distance and percolation similarity metrics in 4.5 do a fairly good job at picking out the communities that arose from the stochastic block model; the integration version does better than the threshold version notably. The Jaccard Index does a good job, but introduces a decent bit of noise, possibly due to the stochastic nature of how the core periphery graph was created to begin with. The SimRank metric, while it does an acceptable job at roughly visualizing the different communities, has much more noise in the metric.

While this analysis is done by observing the heat maps, a straightforward way to recover the core periphery groups would be to group nodes with similar metric rankings. The goal with these simulations was to build an intuition for when the metrics would perform the best.

4.2.5.2 3-Level Core-Periphery

This is a similar case to the previous core-periphery graph above, but with different probabilities and more clusters. In 4.6, the emergence of the communities appears to

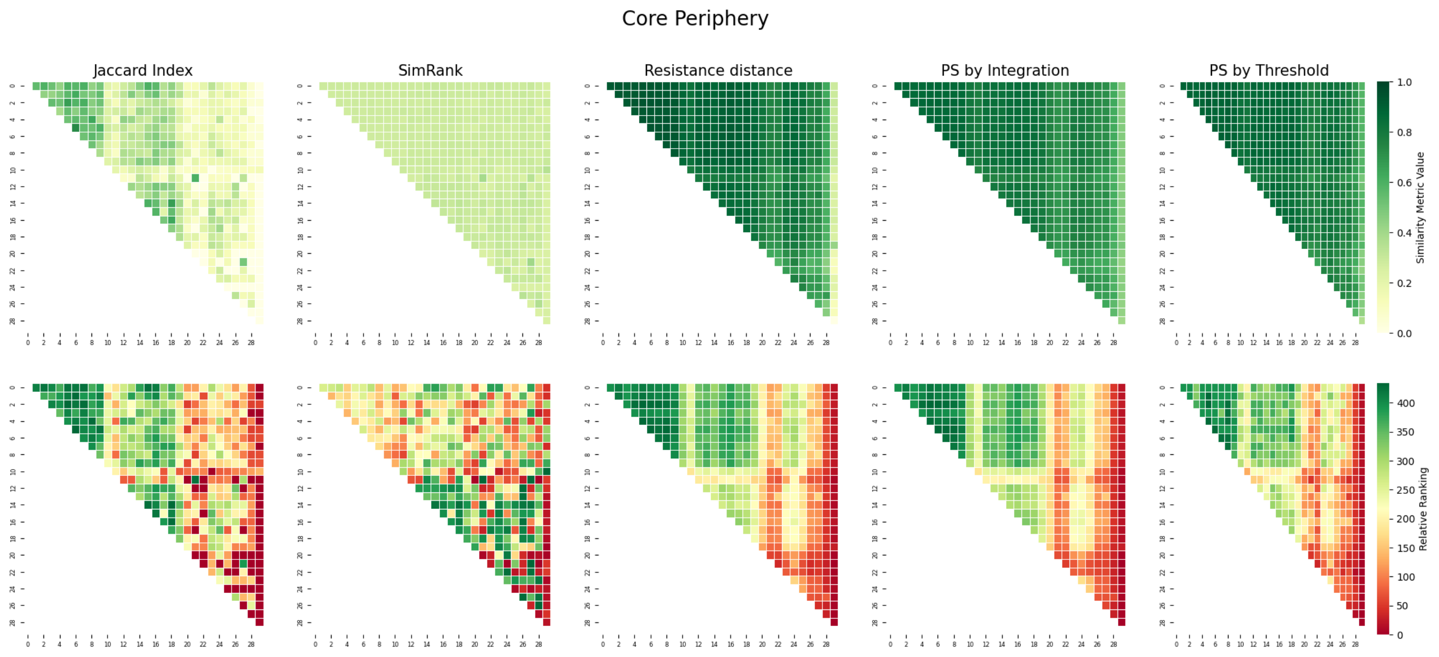

Figure 4.6: Heat maps for a Core Periphery graph with three communities of 10 nodes. The model used $\begin{pmatrix} 0.95 & 0.5 & 0.3 \\ 0.5 & 0.1 & 0.1 \\ 0.3 & 0.1 & 0.05 \end{pmatrix}$ for the stochastic probabilities of an edge existing.

be the same with both the resistance distance and the percolation similarity metrics, but a lot more poor with the other two non-path based metrics.

4.2.6 Zachary's Karate Club

In 4.7, the Jaccard Index graphs place any nodes with a shortest path distance between them greater than 2 as having no similarity. This does a decent job with roughly identifying communities, but there is a lack of nuance in the dissimilarity between any nodes that differ by a distance more than 2. SimRank does an especially good job at identifying these communities as the heatmap ordered by communities shows.

The percolation similarity metrics and the resistance distance metric both produce similar results that do not divide the communities as well. For example, nodes 0 and 33 are very similar, but these are the two "leaders" of the separate factions in real life. The difference between this group of metrics and SimRank and Jaccard Index is that they appear to be measuring two separate qualities. The Jaccard Index and SimRank focuses more on relationships of neighbors of nodes whereas these metrics look at the existence of paths.

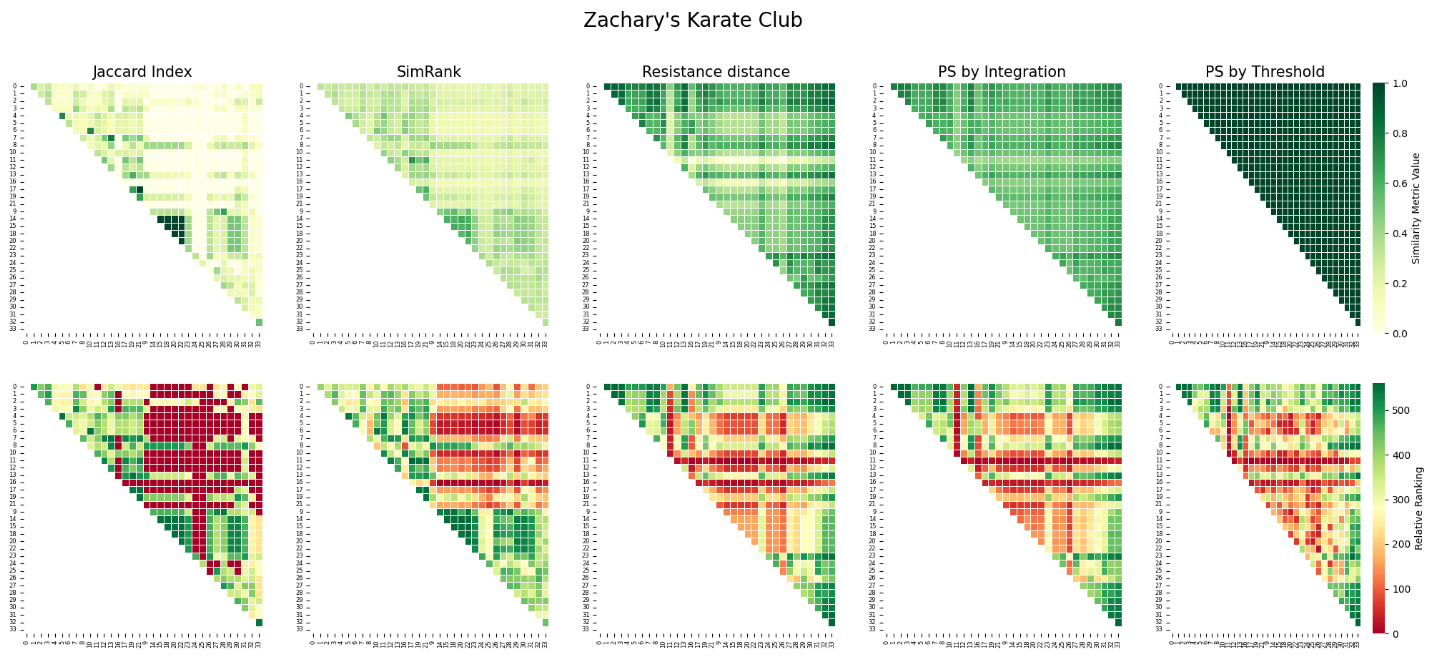

Figure 4.7: Heat maps for Zachary's Karate Club Network. Note the labels on the heatmaps are zero-indexed and organized by community – the nodes up till and including node 9 were found to be in one group, and the nodes after that to another group.

The justification for nodes 0 and 33 being similar according to the resistance distance might be that there are a lot of parallel paths between them due to their high degree and connectivity, which leads to a lower effective resistance. The justification for the percolation similarity metrics is a lot more clear; there are a lot of paths that connect the two nodes such that they are very likely to be connected even when many edges are removed.

Viewing the graph in the context of core periphery models helps to explain why the resistance distance and percolation similarity metrics picked certain nodes as being the most similar to each other. Rombach et al. looked at this graph as well and determined the nodes that are most likely to be in the core via their algorithm, finding nodes 0, 33, 2, 32, 1 as being the top five most likely nodes[2] to be in the core [14]. These nodes are the ones that have the highest similarity with each other and with other nodes – for percolation similarity metrics and resistance metrics, the top 20 ranked pairs of nodes all involve at least one of these nodes, while that is only true for 10 of the top 20 for the Jaccard Index, and 0 for SimRank. This provides support that percolation similarity and resistance distance are capturing a very different meaning of similarity, one that is tied to the definition of a core.

Peel et al. (2017) provide additional justification that the metadata for what is seen as the "ground-truth" for the communities in Zachary's Karate Club might not even be the best division for communities. The authors point out that within the partition space of a graph, there may be multiple effective divisions for communities, specifically pointing out an example with Zachary's Karate Club in Figure 1 of their paper [13]. This suggests that the path-based similarity metrics still might be finding realistic communities, though not the same ones as what is seen in the meta-data.

This example illustrates the strength of percolation similarity as an explanatory tool for how nodes are similar with respect to the paths between them as well as how it ties into identifying nodes in a core.

4.2.7 Florentine Families

One interesting thing to note in 4.8 is that the percolation similarity metrics for their absolute value produce quantities with a much varied range than they did for the other graphs. The biggest difference in rankings between either of the percolation similarity metrics and the resistance distance rankings is 25, out of 120 possible rankings, which puts a reasonable cap on the variance between the two rankings.

[2]There is an offset in indexing in their paper compared to the labels I used, which I account for.

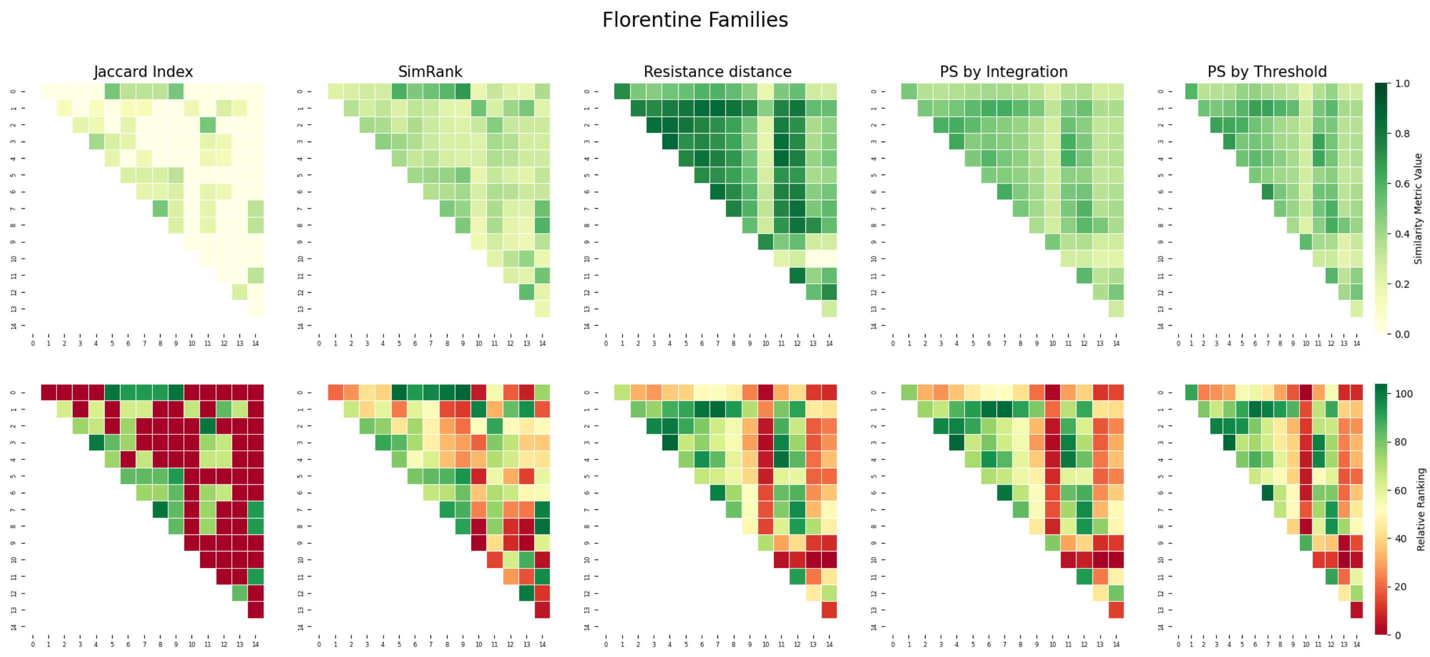

Figure 4.8: Heat maps for the Florentine Families Network.

The ranked heatmaps produce fairly similar results, though they differ a decent bit compared to the SimRank heatmaps. The Jaccard Index heat map has many nodes with a similarity of zero.

The difference between this network and the other ones examined is the sparsity and size of the graph. The Florentine network has an edge density of $\frac{20}{120} \approx 0.17$ with only 15 nodes in total. Since percolation similarity more or less modeled the resistance distance, which itself is a good measure, without much noise, the sparsity of the graph suggests a scenario where it proves to be an effective metric.

4.3 Discussion

The percolation similarity metrics seem to be effective at capturing similarity of nodes in the same way that resistance distance does. The benefit of percolation similarity is that it has a more interpretable and explainable value that does not require an arbitrary context switch to electrical networks. The value of the percolation similarity metric has a meaning in the context of graph theory itself.

Since resistance distance is very similar to percolation similarity, we can further say that resistance distance also captures the idea of information flow, the concept that percolation similarity was motivated by. This has some upsides and downsides

	# Nodes	# Edges	Jaccard Index	SimRank	Resistance Distance	PS Integration	PS Threshold
6-Clique	6	15	0.001	0.040	0.038	1.190	1.198
Connected 6-Cliques	36	96	0.014	3.433	2.040	61.124	61.483
Zachary Karate Club	34	78	0.014	3.211	1.871	34.210	34.339
Florentine Families	16	20	0.003	0.435	0.284	5.866	5.935

Table 4.1: Run-times in terms of seconds for computation of similarity metric values for every pair of nodes in the graph as done on a 2021 M1 Macbook Pro.

depending on the context; for example, with Zachary's Karate Club Network, these information flow based metrics did not find the communities as well as SimRank and (to a lesser extent) the Jaccard Index did. They did, however, explain how and where information would be more likely to flow.

One additional metric to point out is the run-time of these algorithms. Note that the Jaccard Index, SimRank, and resistance distance metrics all use matrix based operations to compute the result for all pairs of nodes, whereas the simulations I computed used sampling to get the result. The main concern with this paper was to find the accuracy of percolation similarity, so I used a high number of samples and smaller intervals for p, which led to both percolation similarity results being fairly slow. The run-time data for the simulations are shown in 4.1.

Note that this shows the performance when running the program on a standard laptop, but the time for the algorithms would vary significantly based off the device it is being run on. The main takeaway is the order of magnitude difference in the percolation similarity metrics compared to the other metrics.

The percolation similarity metrics are slow even for these fairly small graphs, which raises a concern about their effectiveness in practical situations. The benefit of the percolation similarity metrics is that the speed can be adjusted based off the time complexity constraints; if time is not an issue, the simulations can use many samples at small p intervals to gain a fairly accurate result. If time is a constraint, the simulation can use fewer samples trading off accuracy for speed.

The benefit of this type of sample-based approach to computing similarity is that parallelization is straight-forward; splitting up the work onto different CPUs is a natural way to improve the speed by orders of magnitude.

Finally, percolation by integration seemed to have much less noise than percolation by threshold. They both capture the same data, but percolation by integration modeled the resistance distance graph with much less noise in networks than percolation similarity by threshold. This makes sense considering how percolation by integration is using more datapoints from the p-curve and reveals a preference for which percolation similarity metric to use in practice.

Chapter 5

Conclusion

This paper examines valuable qualities in similarity metrics and simulates two implementations of the percolation similarity metric proposed by Lambiotte compared against various other metrics. Percolation similarity attempts to correlate nodes being similar to each other based off how easily information can flow from one node to the other. The simulations reveal that percolation similarity is roughly as effective as resistance distance at being a similarity metric with more interpretability at the trade-off of being slower.

SimRank and the Jaccard Index seem to be more random-walk/neighbor-based metrics, whereas resistance distance and percolation similarity are more path-based. One way this has a real impact on quantifying similarity of nodes is in the task of partitioning communities. For example, in Zachary's Karate Club network, SimRank and Jaccard do a much better job at dividing nodes into the communities that the groups eventually split off into. Percolation similarity and resistance distance make more sense in the context of learning where information can spread by identifying core-like nodes, which is not necessarily the same as nodes that would be in the same social circle.

5.1 Future Directions

5.1.1 Adjusting Parameters

This paper opens the door to using percolation similarity as a metric for better understanding various networks. There are numerous directions where future work could go. First, tweaking the practical implementations and details of percolation similarity seems to be a priority. One area that would be especially important to look into is what threshold would perform the best for percolation similarity by threshold.

This paper used a threshold of 0.5 somewhat arbitrarily, but different thresholds would produce different results that might be more fruitful. For example, Page et al. (1998) explain they use a damping factor of 0.85 in PageRank since that is what they found works best in practice [10]. A thoughtfully selected threshold that is supported by simulations would work best for future implementations of percolation similarity and might reveal more meaningful differences between percolation similarity by integration and percolation similarity by threshold.

5.1.2 Improving Runtime Complexity

Moreover, the current runtime of percolation similarity metrics is not ideal; the suggestions brought up in the previous section would work well for allowing it to be run in much larger settings. Parallelization is the most straightforward way to increase the speed of the metric while maintaining the accuracy. One other method that could be fruitful is having a better method for finding the best p value that is closest to the threshold value. In this paper's implementation, the algorithm tests the probability of percolation for each p value and chooses the p value that is closest to the threshold. Since the function is decreasing, one could use a binary search instead. The issue is that the function's actual values are not known and we have a noisy, not necessarily decreasing, function that results from sampling. This means a modified algorithm for binary search would have to be used, but it would reduce the computational complexity significantly.

For scaling percolation similarity to larger graphs, if the algorithm only has to look at two nodes, one could run the algorithm on an induced subgraph of the neighborhood surrounding the nodes in question. Paths that are very long, walking around the entire network, are likely to be broken up as p increases anyways, so the main paths that determine percolation are the shorter graphs in the nearby neighborhood. This gives a method for extending percolation similarity to larger graphs.

5.1.3 Extending to More and Different Types of Networks

One other extension to this paper could be running percolation similarity on more graphs. This paper ran the similarity metrics on basic cliques and extensions of cliques, core periphery graphs, and famous networks such as Zachary's Karate Club to get an understanding of the metric in the simulation setting. Testing percolation similarity on a network of larger scale with the speed optimizations above would reveal how percolation similarity would translate over to real-life, larger networks.

This paper conjectured that percolation similarity might have more variance in its values with sparser networks compared to denser networks. This was shown with just the Florence Families network, so it might be fruitful to try it on other networks as well.

Other directions of percolation similarity could look at how it could extend to weighted networks and directed networks. One easy way to extend percolation similarity to weighted networks would be to just treat the network as unweighted, but this seems to lose out on the richness of the data. One possible extension that could be looked into is to use the same routine, but measure the strength of the connectivity to be the maximum flow between the two nodes ($\max\{\min_{\text{edge-weight}} z : z \in$ paths from node A to B$\}$). For example, we currently calculate the expectation of the connectivity of the graph using nodes in the perturbed graph being connected as 1 and not connected as 0. With a weighted network, we could use the maximum flow instead of just 1. The benefit of this approach is that it still maintains the explainability property that we desired in a similarity measure.

For directed networks, the extension of percolation similarity is a little less obvious. The benefit of using undirected graphs was that the similarity of nodes A, B is the same as the similarity of nodes B, A—the symmetric property of nodes is easily maintained. For example, now, the existence of a path A to B does not imply the existence of a path from B to A. This presents a bit of an issue because modifying it such that it is symmetric loses the information flow intuition behind percolation similarity but keeping it as such means it is not really a similarity measure. Future work could look into how to maintain both of these properties or to just investigate any new results that may arise treating percolation similarity as just an asymmetric information flow measure.

One more interesting area to examine in percolation similarity would be to look at an unconnected graph. If there are nodes that are not connected to begin with, an alternative construction of percolation similarity could *add* edges with probability q to see the point when the nodes connect, and perform a similar analysis to what this paper completed.

This paper reveals that percolation similarity has the potential to be a valuable metric that captures information flow while maintaining explainability. Future work is needed to explore the feasibility in larger-scale settings, but the metric seems promising and provides its own niche compared to other currently used similarity measures.

Appendix A

Selected Code

A.1 Relevant Helper Functions

```python
# edge_removal_rate is measure of whether we should keep edge \in [0, 1]
# returns 0-1 matrix whether nodes are connected after removing edges
def perturbed_graph_connectivity(graph, edge_removal_rate):
        nodes = list(graph)
        total_nodes = len(nodes)

        if edge_removal_rate < 0 or edge_removal_rate > 1:
                raise Exception(f"edge_removal_rate={edge_removal_rate} is not within [0, 1]")

        perturbed_graph = graph.copy()

        for edge in graph.edges():
                if random() > edge_removal_rate:
                        perturbed_graph.remove_edge(edge[0], edge[1])

        connected = np.zeros((total_nodes, total_nodes))

        for i, source_node in enumerate(list(graph)):
                for j, target_node in enumerate(list(graph)):
                        if i > j:
                                continue
                        if nx.has_path(perturbed_graph, source_node, target_node):
                                connected[i][j] = 1
                        else:
                                connected[i][j] = 0

        return connected
```

```python
# gets the average over n trials of perturbed_graph_connectivity
def avg_perturbed_graph_connectivity(graph, edge_removal_rate, n):
        samples = []

        for i in range(n):
                samples.append(perturbed_graph_connectivity(graph,
                ↪  edge_removal_rate))

        return np.array(samples).mean(axis=0)
```

```python
# computes avg_perturbed_graph_connectivity for all p values
# returns 3d matrix of shape (1/p_interval, len(nodes), len(nodes))
def percolation_similarity_samples(graph, sample_size, p_interval):
        nodes = list(graph)
        total_nodes = len(nodes)

        samples = []

        p = p_interval
        while p < 1:
                samples.append(avg_perturbed_graph_connectivity(graph,
                ↪  p, sample_size))
                p += p_interval

        return np.array(samples)
```

A.2 Percolation Similarity Metrics

```python
# returns numerical integration of "p-curve"
def percolation_similarity_integration(graph, sample_size, p_interval):
        nodes = list(graph)
        total_nodes = len(nodes)

        percolation_similarity = np.zeros((total_nodes, total_nodes))

        p = p_interval
        while p < 1:
                percolation_similarity +=
                ↪  p_interval*avg_perturbed_graph_connectivity(graph,
                ↪  p, sample_size)
                p += p_interval

        return percolation_similarity
```

```python
# return p that is closest to get half nodes connected, using a linear search
def percolation_similarity_threshold(graph, n, p_interval):
        samples = percolation_similarity_samples(graph, n, p_interval)

        # maps [0, 0.5, 1] -> [-0.5, 0, 0.5] -> [0.5, 0, 0.5] -> [0, 1, 0]
        # highest value is closets to 0
        normalized = 1-2*abs(samples - 0.5)
        indices = np.argmax(normalized, axis=0)

        p_closest_to_half = indices*p_interval

        total_nodes = len(list(graph))

        # this sets the p value for nodes that aren't connected at all to have p=1 (still unconnected) by default
        for i, source_node in enumerate(list(graph)):
                for j, target_node in enumerate(list(graph)):
                        if i > j:
                                continue
                        if not nx.has_path(graph, source_node, target_node):
                                p_closest_to_half[i][j] = 1


        # similarity measure, if connected as p=0, then very strong connectivity so strong similarity
        return 1-p_closest_to_half
```

A.3 Comparing Results

```python
# visualize p curve
def generate_p_curve(graph, n, indices, p_interval):
        samples = percolation_similarity_samples(graph, n, p_interval)

        for idx1, idx2 in indices:
                connectivity_along_p = samples[:, idx1, idx2]
                plt.plot([p_interval*k for k in range(1, len(connectivity_along_p)+1)], connectivity_along_p, '-o')
                plt.title(f"Indices {idx1} and {idx2}")
                plt.show()
```

```python
# given a 2d-matrix, returns 2d-matrix with rank of items for ranked heatmap
def get_rank_matrix(matrix):
        # set values below diagonal to be -1, so it doesn't mess up other entries which are 0 in the ranking
        matrix = matrix - np.tril(matrix+1, -1)

        rank_matrix = (rankdata(matrix.ravel(), method='min')-1).reshape(matrix.shape)

        #values under diagonal will be ranked the lowest, so we want to correct for that
        n = len(rank_matrix)
        normalized_rank_matrix = rank_matrix - (n-1)*n/2

        return normalized_rank_matrix
```

Bibliography

[1] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, October 2008.

[2] Ronald L Breiger and Philippa E Pattison. Cumulated social roles: The duality of persons and their algebras. *Social Networks*, 8(3):215–256, 1986.

[3] Luciano da F. Costa. Further generalizations of the jaccard index, 2021.

[4] Renaud Lambiotte et al. Percolation similarity in networks, 2024 (Preprint).

[5] C.M. Fortuin and P.W. Kasteleyn. On the random-cluster model: I. introduction and relation to other models. *Physica*, 57(4):536–564, 1972.

[6] François Fouss, Marco Saerens, and Masashi Shimbo. *Algorithms and Models for Network Data and Link Analysis.* Cambridge University Press, 2016.

[7] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[8] Robert A. Hanneman and Mark Riddle. *Introduction to social network methods.* University of California, Riverside, 2005.

[9] Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, page 538–543, New York, NY, USA, 2002. Association for Computing Machinery.

[10] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[12] Leto Peel, Jean-Charles Delvenne, and Renaud Lambiotte. Multiscale mixing patterns in networks. *Proceedings of the National Academy of Sciences*, 115(16):4057–4062, 2018.

[13] Leto Peel, Daniel B. Larremore, and Aaron Clauset. The ground truth about metadata and community detection in networks. 2016.

[14] M. Puck Rombach, Mason A. Porter, James H. Fowler, and Peter J. Mucha. Core-periphery structure in networks. *SIAM Journal on Applied Mathematics*, 74(1):167–190, January 2014.

[15] Wayne W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):452–473, 1977.