Extending Local Similarity Indexes with KNN for Link Prediction

G. Speegle[1], Y. Bai[2] and Y.-R. Cho[1]
[1]Department of Computer Science, Baylor University, Waco, TX, USA
[2]Amazon, Seattle WA, USA

Abstract— *One of the challenges in big data analytics is discovering previously unknown relationships between objects. Two common examples are suggesting friends in social media networks and predicting interactions between biological proteins. Both of these cases are examples of link prediction. Link prediction algorithms accept a graph and a pair of nodes and predict whether or not there should be an edge between those nodes. Local similarity indices are link prediction algorithms based on the assumption that if two nodes are structurally similar, there should be an edge between them. This concept can be extended by using the machine learning notion of k-nearest neighbor so that an edge from u to v is predicted if nodes similar to u have an edge to v, or nodes similar to v have an edge to u. It is straightforward to extend local similarity indices to k-nn versions of the algorithms, and with suitable selection of k accuracy is improved. Although there is additional computational cost, it can be amortized such that operations such as finding all predictions have similar computation time.*

Keywords: Link Prediction, k-nearest neighbor, Graphs

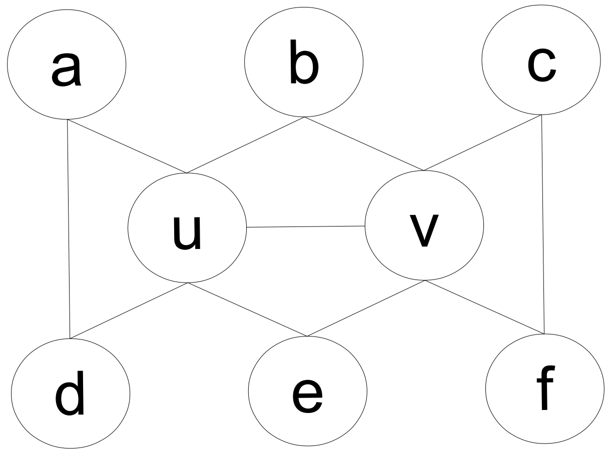


Fig. 1
A SIMPLE GRAPH WITH EIGHT NODES. THE LINK (u, v) IS OF INTEREST. NODES u AND v HAVE TWO COMMON NEIGHBORS, b AND e.

1. Introduction

Graphs are used to represent relationships between real world objects. For example, graphs can represent the distance between two cities, whether or not two people are friends in a social media network, or the interaction between two proteins. However, graphs do not always contain all the information from the real world. If two people are not friends on Facebook, it does not mean they are not friends in real life. Two proteins may interact in a way that has not yet been discovered. Thus, certain edges are "missing" in the graph. Suggesting missing edges is called *link prediction*.

The literature contains many link prediction algorithms. In [9], the algorithms are called indexes and are divided into categories. We are interested in similarity indices, and in particular, local similarity indices. Local similarity indices make a prediction on an edge (u, v) by using the properties of the nodes u and v. In theory, nodes with similar properties are more likely to have an edge than nodes that do not. Local similarity indices are computationally very efficient and reasonably good at predicting edges.

This work focuses on extending the theory behind local similarity indices in a natural way. The concept that two nodes are similar implies an edge between them only uses a portion of the information available. We consider a set of k nodes similar to u (and respectively, v). The more similar nodes that have an edge to v (or u), the more likely (u, v) is to exist. This technique is commonly known as k-nn. However, in order to avoid confusion between the similar nodes and the nodes adjacent in a graph, we use the term k-similarity to refer to the former case.

To see the difference between local similarity indices and k-similarity consider the simple graph in Figure 1. A key measurement for local similarity is the number of common neighbors between two nodes. Nodes u and v have two neighbors in common, specifically, nodes b and e.

Finding the k-similarity nodes to u and v is more complex. Table 1 shows how the similarity would be computed for the graph using common neighbors as the similarity criterion. Given a value for k, k-similarity can be computed from the table. For example, with $k = 1$, the most similar node to u is v and the most similar node to v is u. Since neither the edge (v, v) nor (u, u) is in the graph, the score for k-similarity using common neighbors and k=1 for (u, v) would be zero. Note that $k = 2$ results in a tie, which is arbitrarily broken. Assume a is selected as the second most similar node to u. Since (a, v) is not in the graph, the number of similar nodes to u that are neighbors of v is still zero. However, if a is

Table 1
THE k-SIMILARITY CALCULATIONS FOR FIGURE 1 USING COMMON NEIGHBORS AS THE SIMILARITY CRITERION. U SHARED IS THE NUMBER OF NEIGHBORS IN COMMON BETWEEN u AND THE NODE. V SHARED IS SIMILAR FOR v.

Node	**Neighbors**	**U Shared**	**V Shared**
u	a,b,d,e,v	5	2
v	b,c,e,f,u	2	5
a	d,u	1	1
b	u,v	1	1
c	v,f	1	1
d	a,u	1	1
e	u,v	1	1
f	c,v	1	1

selected as the second most similar node to v, since (a, u) is in the graph, the number of similar nodes to v that are neighbors of u is now one, and the reported score is one.

Using k-similarity with local similarity indices is very straightforward. Once the framework is in place, creating a k-similarity version of the index requires writing one method with typically no more than a few lines of code. As shown in Section 3, with appropriate selection of k, the k-similarity version can perform better than the native version for predicting links in biological graphs. However, sometimes the k-similarity version performs significantly worse, leading to speculation as to what properties of the indices can be exploited by k-similarity.

The notation in this paper extends the typical graph notation in order to simplify discussions. A graph G is defined as $G = (V, E)$ such that V is the set of vertices (or nodes) in the graph, and E is the set of edges. Let U be the set of all possible edges in G. The link prediction problem, as defined in [9], is to find the edges in $U - E$ that should be in G. Let the neighbors of a vertex $v \in V$ be denoted $\Gamma(v)$. The degree of a vertex v is d_v. When needed, the k-similarity vertices to v are denoted $K(v)$. The graph used in this work is modified from the BioGRID Interaction Database [14]. The graph consists of 6,186 nodes and 192,474 unique edges. The graph has been modified from [14] to remove redundant edges and self-loops.

This paper proceeds by providing background information on models for link prediction in graphs. Next, the paper describes the development of k-similarity algorithms and the experiments showing the impact of k-nn versus native applications of the local similarity indexes. Some surprising issues are discussed in Section 4. The conclusion and future work ends the paper.

2. Related Work

Link prediction is a popular research topic. In [9], the link prediction techniques are divided into similarity based algorithms, maximum likelihood methods and probabilistic models. Similarity based algorithms assume that an edge (u, v) is more likely if the nodes u and v are similar, based on some criteria. Clearly, k-nn methods are similarity based. Maximum likelihood methods assume the graph has an underlying structure, so that edges which contribute towards the structure are favored over edges that do not. Examples include the dendrogram in [4] and block models [2]. The probabilistic approaches attempt to model the underlying graph structure and predict the missing edges based on the probability of the link given the model. See [9] for more about maximum likelihood and probabilistic methods.

More recently, matrix factorization has been used for link prediction [10]. Matrix factorization is similar to k-similarity in that it incorporates other prediction models. As with k-similarity, matrix factorization does not ensure a better result than using the native version of the similarity index. Also, as with k-similarity, the matrix factorization model can be optimized for AUC, and this is done in [10]. Fundamentally, the matrix factorization model is a supervised learning technique which combines the graph topology with side information. It requires training linear in the number of possible edges, or quadratic in the number of vertices, which is similar to k-similarity for making all predictions.

The work by Lu and Zhou [9] further divides similarity methods into local, global and quasi-local indices. A local index uses structural information such as the number of common neighbors. Global similarity are typically based on properties of the entire graph, such as the number of moves two random walkers starting at u and v make before they meet. Quasi-local indexes perform trade-offs between the high computational complexity of global indexes versus the generally weaker predictive power of local similarity indices. For example, one quasi-local method considers not only the common neighbors, but also all common nodes within a distance of 2 from each node [15].

Within the hierarchy in [9], the extension of local similarity indexes to k-nn methods is best represented as a quasi-local index, in that more than local information is used, but with optimization it is possible to consider only information in a small portion of the graph.

2.1 Local Similarity Indices

Ten similarity measures (called indices in [9]) are used to test the application of k-similarity for link prediction. We describe each of the algorithms in detail in this section. The similarity measures are presented in alphabetical order for easier reference.

2.1.1 Adamic-Adar

Abbreviated AA, this similarity measure originally presented in [1], is based on shared items on web pages. If two students have many items in common, they are more likely to be friends. Additionally, rare shared items, contribute more to the similarity score. The similarity score is modified slightly in [8] to consider common neighbors as the shared items. The modified formula is

$$s^{\mathrm{AA}}(x, y) = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{\log d_z}$$

2.1.2 Common Neighbors

Abbreviated CN, this similarity measure is one of the most basic, but performs very well. It is used in [11] to show that the probability of scientists collaborating increases with the number of collaborators they have in common. For nine of the ten local similarity measures in [9], the absence of common neighbors yields a similarity score of zero.

$$s^{\text{CN}}(x,y) = |\Gamma(x) \cap \Gamma(y)|$$

.

2.1.3 Cosine Similarity

Abbreviated cos, it is labeled as the Salton Index in [9]. The cosine similarity is based on the cosine of the angle between two vectors. By representing the neighbors as bit vectors of nodes, the cosine can be computed. Alternatively, the cosine similarity can be calculated directly from the properties of the nodes as

$$s^{\text{cos}}(x,y) = \frac{|\Gamma(x) \cap \Gamma(y)|}{\sqrt{d_x * d_y}}$$

2.1.4 Hub Depressed Index

Abbreviated HDI, this similarity measure is new in [9]. It is analogous to HPI, except hubs are depressed due to their large degree. The similarity formula is

$$s^{\text{HDI}}(x,y) = \frac{|\Gamma(x) \cap \Gamma(y)|}{\max(d_x, d_y)}$$

2.1.5 Hub Promoted Index

Abbreviated HPI, this similarity appears in [13] as the topological overlap between two nodes. Collections of nodes with high topological overlap tend to represent biologically interesting modules. The similarity formula is

$$s^{\text{HPI}}(x,y) = \frac{|\Gamma(x) \cap \Gamma(y)|}{\min(d_x, d_y)}$$

2.1.6 Jaccard Index

Abbreviated J, this similarity measure was defined by Paul Jaccard over 100 years ago. It is a statistic for comparing the similarity of two sets, and it is applied here by comparing the sets of neighbors between two nodes. The formula is

$$s^{J}(x,y) = \frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}$$

2.1.7 Leicht-Holme-Newman Index

Abbreviated LHN, this similarity measure in [7] can be considered a near inverse of k-similarity. Vertices u and v are similar if either has a neighbor w that is similar to the other. Consider 1-NN, in which if the most similar node to u is a neighbor of v, then u and v are similar. The significant distinction is that LHN reports similarity if any neighbor is similar, while k-similarity reports similarity if any similar node is a neighbor. Since link prediction is defined by similarity between two vertices, the simplified formula in [7] is used in which the number of common neighbors is divided by the expected number of neighbors. It is proportional to the formula:

$$s^{\text{LHN}}(x,y) = \frac{|\Gamma(x) \cap \Gamma(y)|}{d_x * d_y}$$

2.1.8 Preferential Attachment

Abbreviated PA, this local similarity method does not use common neighbors. Over ten years ago, the concept was used to note that new edges tended to be incident on high density nodes more often than low density nodes [3]. A similarity measure based on this concept is used in several applications (see [9] for a listing) and is defined as

$$s^{\text{PA}} = d_x * d_y$$

2.1.9 Resource Allocation

Abbreviated RA, this similarity measure is based on the flow of resources in a graph [12]. Resources leaving one node flow into all of its neighbors. The amount of the resource that flows into the target represents the resource allocation. Specifically, the similarity formula is [9]

$$s^{\text{RA}}(x,y) = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{d_z}$$

2.1.10 Sorensen

Abbreviated S, [9] states this similarity measure is primarily used for ecological community data. The formula is

$$s^{S}(x,y) = \frac{2 * |\Gamma(x) \cap \Gamma(y)|}{d_x + d_y}$$

3. K-Similarity Extensions to Local Similarity Indexes

Adapting a local similarity index to a k-similarity approach is straightforward. The prediction of an edge (x,y) is the greater number of elements in the k-nn of each node that are neighbors of the other node in G.

$$\begin{aligned} s^{K}(x,y) = \quad & \max(|\{v | v \in K(x) \wedge (v,y) \in E\}|, \\ & |\{v | v \in K(y) \wedge (v,x) \in E\}|) \end{aligned}$$

Making a single prediction with k-similarity is significantly slower than using the local index alone. Finding the k-similarity of a node requires $O(|V|)$ local similarity calculations, while just using the index requires only one. However, since the k-similarity of a node can be calculated once and saved, making all of the predictions for a graph requires $O(|V|^2)$ local similarity calculations, identical to the number when using the index alone. Thus, for finding the edges most likely to be missing from a graph, the k-similarity approach is not prohibitively slow.

3.1 K Selection

One of the challenges for using k-similarity is the selection of k. The general wisdom is selecting k does not significantly alter the effectiveness of the algorithm, so long as k is not too small. In [6] optimal values for k can be efficiently determined as long as the error can be incrementally calculated. For link prediction, AUC is used as the scoring factor, enabling a $O(|V|^2)$ algorithm to determine the optimal k.

The maximum reasonable value for k is $\sqrt{|V|}$ [6], [5]. Call this value maxK. The algorithm for calculating the best k uses a maxK $\times$ maxK matrix M. M_{ij} is the percentage of edges with a predicted score of j or less when using the i-nearest neighbors. Thus, it is a cumulative distribution and when $j \geq i, M_{ij} = 1.0$. To efficiently generate M, the algorithm iterates over the edges in the graph. The maxK nodes are found for each node incident on the edge and the appropriate M_{ij} is incremented. In one pass over the matrix, the cumulative values are computed and each cell is divided by the total number of edges.

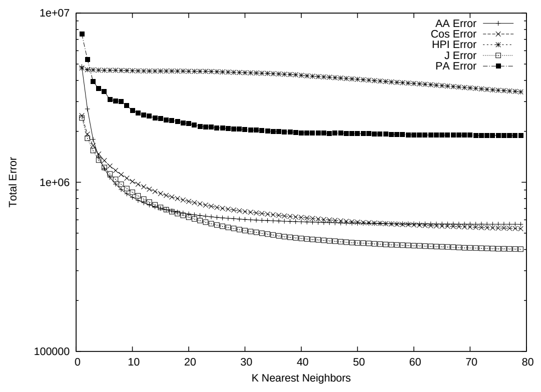
The error for a particular k can be computed by considering all of the non-edges ($(u, v) \in U - E$). The k-similarity score j is calculated for each nearest neighbor set of size $1 \leq i \leq$ maxK. The error for (u, v) with i-nearest neighbors is the number of actual edges with a lower score ($M_{i(j-1)}$) plus one-half the number of actual edges with the same score (M_{ij}). The results are in Figure 2. The indexes are split into two charts for clarity.

From Figure 2, it is clear that the error improves with increasing k. In fact, for nine of the ten indexes, the lowest error occurs with $k = 79$. However, it is also clear that each error improves only marginally after some point. We use the least k such that the error improves by less than 1% as worthy of investigation.

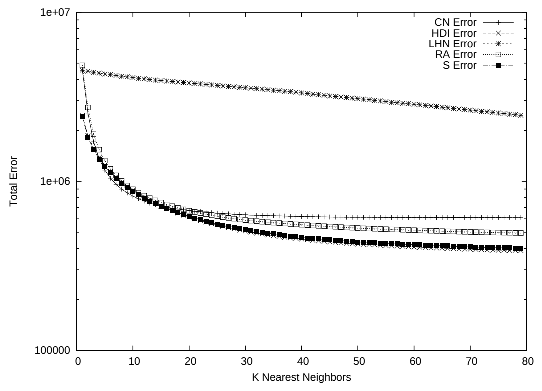
3.2 Area Under the Curve

Our next experiments show the impact of different values of k within the k-similarity framework on AUC, a common measurement of link prediction algorithms. Experiment 1 calculates the AUC using only the native local similarity index. The second through fourth experiments use different values of k. Experiment 2 uses an arbitrarily chosen value of 10. Experiment 3 uses the first k such that the error never improves by more than 1%. Experiment 4 uses the k with the least error. Table 2 shows the results. It should be noted that although increasing k increases the work done, it does not incur any additional index computations, which are the most expensive operations. Therefore, the runtime for each index is similar for different values of k.

It is interesting to note that while using k-similarity is generally helpful, it does not always improve the performance of the index. In particular, HPI performs very badly as a k-similarity index (see Section 4.1 for an explanation) . Conversely, HDI is exceptionally powerful as a k-similarity index, outperforming all other techniques. Also, note that the improvement between Experiment 3 and Experiment 4



(a) The k-nn error for five of the local indexes: AA, cos, HPI, J and PA.



(b) The k-nn error for five of the local indexes: CN, HDI, LHN, RA and S.

Fig. 2
THE ERROR FOR THE LOCAL SIMILARITY INDEXES IN [9].

Table 2
THE AUC FOR THE TEN LOCAL SIMILARITY INDEXES IN [9] UNDER FOUR EXPERIMENTS. THE FIRST USES THE SIMILARITY INDEX ALONE. THE NEXT THREE USE THE k-SIMILARITY APPROACH. IN EXPERIMENT 2, $k = 10$. IN EXPERIMENT 3, THE k IS INCLUDED IN THE RESULTS. IN EXPERIMENT 4, THE k IS 79 EXCEPT FOR COMMON NEIGHBOR WHERE THE BEST k VALUE IS 68. THE VALUES ARE THE AVERAGES OF 5 EXECUTIONS. THE BEST RESULTS FOR EACH EXECUTION IS IN BOLD.

Index	Exp 1	Exp 2	Exp 3	Exp 4
AA	0.906	0.912	0.934 (22)	0.937
CN	0.903	**0.917**	0.928 (20)	0.933
Cos	0.839	0.895	0.934 (31)	0.944
HDI	0.824	0.907	0.951 (38)	**0.958**
HPI	0.819	0.520	0.512 (10)	0.646
J	0.833	0.911	**0.952** (38)	0.957
LHN	0.669	0.569	0.565 (10)	0.745
PA	0.895	0.720	0.772 (23)	0.797
RA	**0.910**	0.908	0.941 (30)	0.953
S	0.836	0.913	0.948 (38)	**0.958**

is typically small, especially under the better performing techniques – HDI, J, RA and S.

3.3 LOOCV

Cross validation is a commonly used technique to measure the effectiveness of machine learning algorithms. In general, the data is divided into two sets; one containing the training data and one containing the test data. The algorithm is trained on the training data and then attempts to predict the test data. The experiment is repeated such that every data item is used in a test set exactly once. In leave-one-out cross-validation, the test set consists of exactly one element. For link prediction, the elements are the edges in the graph. Thus, we execute each of the local similarity indexes on a graph with an edge removed and see if it predicts the existence of the edge.

Natively, nine of the local indexes found 184,357 of the 192,474 edges (95.8%) from the BioGrid yeast PPI network version 3.1.84. The lone exception is Preferential Attachment, which found 192,110 (99.8%). This is to be expected since all of the local indexes (except Preferential Attachment) will return a score of 0 exactly when two nodes do not have any common neighbors. Preferential Attachment will return a score of 0 exactly when at least one node does not have any neighbors. This occurs when a node of degree one is part of the edge removed for the cross validation. There are 364 nodes in the graph with degree 1. Likewise, each local similarity index performed the LOOCV in well under a minute, typically in a few seconds.

For k-similarity the runtime is significantly worse with indexes requiring 24-48 hours. Since the graph changes for each edge removed, the optimization of computing the k-similarity of each node once and saving it for later use is not possible. As a simple example, consider again the graph in Figure 1. Removing the edge (u, v) changes the common neighbors. The three most similar nodes to u are now v, a and d. None of these nodes have an edge to v. Similarly, the three most similar nodes to v are u, c and f. None of these nodes have an edge to u. Therefore, under LOOCV, the score for the k-similarity using common neighbors would be zero, indicating the node is not found. Recall the score for the node with (u, v) included is likely to be above zero, depending on the tie breaking process.

Using the k-similarity version of the local similarity index resulted in better LOOCV performance for three of the indexes (AA, RA and CN), slightly worse performance for four of the indexes (Cosine, HDI, Jaccard, PA and Sorensen) and terrible performance for HPI and LHN. Table 3 has the results of the LOOCV experiments.

3.4 Testing Predictions

The genome information is constantly evolving, creating the need for new link predictions. We can exploit this to provide another mechanism for testing the local similarity indexes and the impact of using k-similarity. For this experiment, we use version 3.1.73 of the yeast PPI data from BioGRID (last modified 2011-01-31). Between version 3.1.73 and version 3.1.84, 28,974 edges were added to the yeast protein interaction graph.

Table 3
THE NUMBER OF EDGES FOUND DURING LEAVE ONE OUT CROSS-VALIDATION FOR BOTH LOCAL SIMILARITY INDEXES AND THE k-SIMILARITY VERSION OF THE INDEX. NINE OF THE INDEXES FOUND 184,357 (95.8%) OF THE EDGES IN THE BIOGRID YEAST PPI NETWORK VERSION 3.1.84.

Index	**LOOCV**	**LOOCV with KNN**
AA	184357	191539 (99.5%)
CN	184357	191815 (99.7%)
Cosine	184357	168267 (87.4%)
HDI	184357	173017 (89.9%)
HPI	184357	19401 (10.1%)
Jaccard	184357	173226 (90.0%)
LHN	184357	5579 (2.9%)
PA	192110 (99.8%)	191969 (99.7%)
RA	184357	191815 (99.7%)
Sorensen	184357	173226 (90.0%)

The best predictions for each index, both used natively and with k-similarity are generated. Since the k-similarity approach does not differentiate between ties, all of the perfect scores are included in the experiment. For most indexes, relatively few links received a perfect score, so the top 100 predictions are used. However, Common Neighbors produced 271 ”perfect” scores, while Adamic-Adar yielded 157. We then count the number of predicted links that are present in the newer data. The results are in Table 4. It is interesting to note that the local similarity indexes by themselves tend to be extremely successful or fail miserably with this experiment. Six of the native implementations did not predict any found links, while four predicted a number of links extremely unlikely to be found by chance. Only three of indexes did not make a successful prediction when combined with k-similarity but for many of the indexes, random chance could find the same number of links. It is interesting to note that the native Adamic-Adar index performs the best.

4. Issues with Extending Local Similarity Indexes with KNN

The results from Section 3 indicate using local similarity indexes within a k-similarity approach can be beneficial. However, there are some obvious issues. First, the surprisingly bad performance of the HPI index must be examined. Second, optimization of the running time of the approaches must be considered. Finally, the ease and power of creating new k-similarity approaches is demonstrated.

4.1 Hub Promoted Index Performance

The Hub Promoted Index (HPI) [13] is an effective local similarity index. Under our experiments in Section 3, HPI scored 0.819 on the AUC experiment (see Table 2). Although that is not particularly strong compared to the other local

Table 4
THE NUMBER OF PREDICTIONS FROM BIOGRID VERSION 3.1.73 THAT WERE DISCOVERED AS OF BIOGRID VERSION 3.1.84. THE NATIVE CHANCE AND KNN CHANCE COLUMNS REPRESENT THE PROBABILITY THE NUMBER OF EDGES COULD BE GUESSED RANDOMLY.

Index	Predictions	Native Found	Native Chance	KNN Found	KNN Chance
AA	157	14	7.8×10^{-15}	5	2.5×10^{-7}
CN	271	18	1.3×10^{-14}	6	1.3×10^{-7}
Cosine	100	0	1	3	8.5×10^{-5}
HDI	100	0	1	3	8.5×10^{-5}
HPI	100	0	1	0	1
Jaccard	100	0	1	1	0.08
LHN	100	0	1	1	0.08
PA	100	6	3.5×10^{-10}	0	1
RA	100	8	4.1×10^{-14}	0	1
Sorensen	100	0	1	1	0.08

similarity indexes, it is well above random chance and the index performed much better on the data in [9].

However, using HPI within our k-similarity framework performed exceptionally poorly. Using a k of 10, HPI scored 0.520 for AUC, barely better than random chance. Increasing k to the maximum (78) yields a score of 0.646, which is better, but still below any of the local similarity indexes by themselves. Likewise, the error score for HPI (Figure 2) improves very slowly. In fact, the first time the error improves by less than 1% is at $k = 3$.

Comparing HPI to the basic common neighbors approach sheds light on the reason it does not work well within a k-similarity framework. Recall the formula for HPI as

$$s^{\text{HPI}}(x, y) = \frac{|\Gamma(x) \cap \Gamma(y)|}{\min(k_x, k_y)}$$

where k_x is the degree of node x. Therefore, the best possible score for HPI is 1, and is achieved when either $\Gamma(x) \subseteq \Gamma(y)$ or $\Gamma(y) \subseteq \Gamma(x)$.

Now consider the score for the edge (u, v) within the k-similarity framework. We consider only u, as the case for v is identical. The most similar nodes to u would be those nodes x such that $\Gamma(x) \subseteq \Gamma(u)$. This is much more likely in cases where the degree of x is very low. In particular, if the degree of x is 1, then if the neighbor of x is also a neighbor of u, then the HPI score of (u, x) is 1. Given the existence of hubs within the yeast PPI network, it is very likely for a sufficiently large set of such nodes to exist. Note that since these nodes have a degree of 1, and their only neighbor is the hub, these nodes cannot have an edge to v, and thus fail to add to the score.

As a comparison, consider the common neighbors similarity index under k-similarity. A node of degree 1 is less likely to be the most similar because a node of high degree would be more likely to have two or more common neighbors. Of course, a node of high degree would also have many non-common neighbors, but in the basic approach, there is no penalty for uncommon neighbors.

4.2 Common Neighbor Optimization

Nine of the ten local similarity indexes (all but Preferential Attachment) use the number of common neighbors as a significant portion of the score calculation. In seven of these approaches (all but Adamic-Adar and Resource Allocation), the number of common neighbors serves as the numerator of the score function. In AA and RA, each common neighbor contributes to the score.

For the k-similarity framework, finding the k most similar nodes is a significant consumption of resources. In the naïve case, all other vertices must be checked. However, for the common neighbor based indexes, given a node u, only nodes with a distance of 2 or less may have a common node with u. Thus, only those nodes need to be considered. Given an adjacency matrix A, $A \cup A^2$ (where $a_{ij} \in A \cup B$ is true if the entry is true in A or B) can be precomputed to hold the possible common neighbors. Thus, we can find the k-similarity of a node without considering all possible nodes, potentially saving significant time. Table 5 shows the speedup to be 25-75%.

4.3 Extension from Native to KNN

The framework established allows very rapid development of both native and k-similarity local similarity indexes. For example, consider an unusual local similarity index developed for this paper called Asymmetric. This index is designed to work in both directed and undirected graphs, so the similarity for edge (u, v) is allowed to be different from the similarity for edge (v, u). Specifically, the score for (u, v) is defined as

$$1.0 - \frac{|\Gamma(u) - \Gamma(v)|}{|\Gamma(u)|}$$

.

For undirected graphs, the score of (u, v) is the maximum of the directed scores for (u, v) and (v, u).

To implement the AUC test for Asymmetric requires creating the class and implementing the prediction method. In this case, five new lines of code have to be added to

Table 5
Comparison of the time required to perform k-similarity predictions when the graph is pre-processed such that only possible nodes are searched as opposed to the entire graph. Preferential Attachment is not included since the optimization is not applicable. The k from Experiment 3 in Table 2 is used. The time is in milliseconds.

	KNN		**Optimized**		
Index	**Score**	**Time**	**Score**	**Time**	**Speedup**
AA	0.934	1208440	0.934	970389	1.25
CN	0.928	166743	0.928	105703	1.58
cos	0.934	760410	0.934	430137	1.77
HDI	0.951	766940	0.951	434552	1.77
HPI	0.512	786422	0.513	444833	1.77
J	0.952	1326770	0.951	824168	1.61
LHN	0.565	766476	0.566	428172	1.79
RA	0.941	1200200	0.939	973598	1.23
S	0.948	763862	0.950	429719	1.78

implement Asymmetric (most of the local similarity indexes from [9] only needed one additional line of code). The test suite can then be executed on a given graph by simply passing parameters to the class. For example, the parameter -m indicates finding the best K, while -e finds the AUC for both the native and k-similarity implementation. It took less time to create the class than to run the experiments.

For the curious, the Asymmetric index has an AUC of 0.829 run natively, 0.952 with $k = 38$ and 0.960 with $k = 78$. After $k = 38$, the error improves by less than 1%, which is the cut-off used in Experiment 3 reported in Table 2. The Asymmetric index using k-similarity with $k = 78$ outperformed all of the other local similarity indexes under any conditions, *even though the native implementation was unimpressive.*. Thus, using the k-similarity framework can lead to new and improved algorithms for link prediction.

5. Conclusion

Local similarity indexes (see [9]) are quick tools for predicting links in graphs. Typically, these tools look at only the portion of the graph immediately connected to a node or to properties of the node. This allows large graphs to be processed efficiently and predictions to be made when more robust techniques would be too computationally expensive.

The machine learning technique k-nn can be applied by assuming that if the k most similar nodes to v have a link to u, then v should have a link to u. The challenge is to find the k most similar nodes. However, local similarity indexes are exactly intended to find similar nodes, leading to a natural integration between the local similarity indexes and k-nn.

We propose a framework in which given a local similarity index s, a prediction for edge (x, y) in graph G can be made. Based on s, the k most similar neighbors to x and y are found. The score of the link is

$$\begin{aligned} s^K(x,y) = \quad & \max(|\{v|v \in K(x) \wedge (v,y) \in E\}|, \\ & |\{v|v \in K(y) \wedge (v,x) \in E\}|) \end{aligned}$$

For each index, the best k is found by following the techniques in [6].

Using an index in a k-similarity framework as opposed to native application produces unpredictable results. In some cases, the k-similarity version of the index performs considerably better than the native application, but in some cases the k-similarity version is far worse (see Table 2). For testing purposes, the k-similarity versions are significantly slower under AUC and unreasonable under LOOCV. However, when finding all possible predictions, the k-similarity versions required only twice as much time, indicating reasonable performance for some applications.

This work needs to be extended in several ways. There are additional graphs to be considered, such as co-authorship or social media. Social media applications with hundreds of millions of nodes would require the optimizations in Section 4.2 to run efficiently. Also, the use of k-similarity as a framework with reasonable performance opens the opportunity for similarly indexes which may perform poorly when applied natively. Thus, new similarity indexes can be developed. In particular, semantic similarity can be considered, as well as structural similarity.

References

[1] Lada Adamic and Eytan Adar. Friends and neighbors on the web. *Social Networks*, 25(3):211–230, 2003.

[2] Edoardo M. Airoldi, David M. Blei, Stephen E. Fienberg, and Eric P. Xing. Mixed membership stochastic blockmodels. *J. Mach. Learn. Res.*, 9:1981–2014, June 2008.

[3] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[4] Aaron Clauset, Christopher Moore, and M.E.J. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453:98–101, 2008.

[5] Anil K. Ghosh. On nearest neighbor classification using adaptive choice of k. *Journal of Computational and Graphical Statistics*, 16(2):482–502, 2007.

[6] G. Hamerly and G. Speegle. Best k for knn. In *Proceedings of the 27th British National Conference on Databases, BNCOD 27*, pages 37–54, June 2010.

[7] E. A. Leicht, Petter Holme, and M. E. J. Newman. Vertex similarity in networks. *Phys. Rev. E*, 73:026120, Feb 2006.

[8] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 556–559, 2003.

[9] L. Lu and T. Zhou. Link prediction in complex networks: A survey. *Physcia A: Statistical Mechanics and it Applications*, 390(6):1150–1170, 2011.

[10] A. Menon and C. Elkan. Link prediction via matrix factorization. *Machine Learning and Knowledge Discovery in Databases*, pages 437–452, 2011.

[11] M. E. J. Newman. Clustering and preferential attachment in growing networks. *Phys. Rev. E*, 64:025102, Jul 2001.

[12] Qing Ou, Ying-Di Jin, Tao Zhou, Bing-Hong Wang, and Bao-Qun Yin. Power-law strength-degree correlation from resource-allocation dynamics on weighted networks. *Phys. Rev. E*, 75:021102, Feb 2007.

[13] E. Ravasz, A. Somera, L. D. A. Mongru, Z. N. Oltvai, and A.-L. Barabasi. Hierarchical organization of modularity in metabolic networks. *Science*, 297:1551–1555, August 2002.

[14] C. Stark, B.J. Breitkreutz, A. Chatr-Aryamontri, and et al. The biogrid interaction database: 2011 update. *Nucleic Acids Research*, 39:D698–D704, 2011.

[15] T. Zhou, L. Linyuan, and Y.-C. Zhang. Predicting missing links via local information. *The European Physical Jounral B*, 71:623–630, 2009.