LLNL-TR-677818

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Enhancing Community Detection By Affinity-based Edge Weighting Scheme

A. Yoo, G. Sanders, V. Henson, P. Vassilevski

October 5, 2015

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Enhancing Community Detection By Affinity-based Edge Weighting Scheme[*]

Andy Yoo, Geoffrey Sanders, Van Henson, and Panayot Vassilevski

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94550

Abstract

Community detection refers to an important graph analytics problem of finding a set of densely-connected subgraphs in a graph and has gained a great deal of interest recently. The performance of current community detection algorithms is limited by an inherent constraint of unweighted graphs that offer very little information on their internal community structures. In this paper, we propose a new scheme to address this issue that weights the edges in a given graph based on recently proposed vertex affinity. The vertex affinity quantifies the proximity between two vertices in terms of their clustering strength, and therefore, it is ideal for graph analytics applications such as community detection. We also demonstrate that the affinity-based edge weighting scheme can improve the performance of community detection algorithms significantly.

1 Introduction

Graphs offer an effective means to explore the relations between data objects and have been used as one of the most popular knowledge representation tools. Graph analytics has become increasingly important in recent years as the demand for the capability to mine useful information from an increasingly large volume of data has grown stronger. The graph analytics has found its uses in a wide range of applications. Among these graph analytics applications, community detection has received a great deal of attention in particular. Although there is no clear-cut consensus on what community is, the community in a graph is often loosely defined as a subgraph whose vertices are more densely connected to each other than to the rest of vertices in the graph. Community detection refers to a graph analytics problem to find a set of communities within the graph [13].

The strong interest in community detection is mainly due to the fact that the obscure structures in complex real-world graphs revealed by community detection algorithms may contain crucial information for solving target applications. For example, communities in a social network such as Facebook [12] may represent groups of friends or known associates and can be used

[*]This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

for solving various important problems in social sciences. A community in a protein-protein interaction network [19] in biology research may indicate a group of proteins that perform the same or similar functions and can provide valuable information for drug discovery. There are many other areas of interest where the community detection plays a key role, which include social media analysis, computer science, engineering, e-commerce, politics, national security, and web mining, just to name a few [11, 14, 21, 9, 16, 5, 10, 1, 4, 2, 15].

Algorithms to find the communities have been studied quite extensively as well, and numerous community detection algorithms have been proposed lately. Basically, these algorithms differ from each other in measuring and identifying the edges that connect the vertices in the same community. A divisive community detection method detects those edges connecting vertices in different communities and repeatedly remove them from the graph [6, 3]. With this method, the community structures surface after enough inter-community edges are removed. However, the computational cost is very high since usually a large number of edges need to be removed before the hidden community structures start to emerge. Agglomerative algorithms, on the other hand, find communities by combining densely linked vertices (and potentially partially identified communities) recursively. The popular Louvain algorithm [2] belongs to this class of community detection algorithms. Spectral clustering-based methods [22] use the eigenvalues and eigenvectors of the adjacency matrix (and its variants) of the target graph. Their computation is also very expensive, especially for large graphs, due to the high computational cost involved in calculating a large number of eigenpairs. There is also a class of community detection algorithms based on random walks [27, 24]. The basic idea of these algorithms is that a set of vertices in the same community can be found via multiple short random walks because if a random walk visit a vertex in a community, the next vertex it visits is likely to be in the same community. There are many other community detection algorithms reported in the literature, but we will not discuss them here due to space limitation. Interested readers should refer to an excellent survey on community detection algorithms by Fortunato [13] for more detailed discussion.

Most existing community detection algorithms typically take unweighted graphs as input. A major drawback of the unweighted graphs is that they do not provide any hint with respect to finding communities. That is, each edge in the unweighted graphs carries no information on the clustering of its endpoints and their neighbors, other than the fact two endpoints are related. This drawback of unweighted graphs can significantly limit the performance of community detection algorithms. To address this issue, we propose a new edge weighting scheme that transforms a given unweighted graph into a weighted graph whose edges contain information pertaining to vertex clustering.

The new edge weighting scheme is based on a recently proposed vertex proximity measure called *affinity* [28]. The affinity concept is motivated by an observation that using an improper proximity metric for solving a graph analytics problem often yields poor results [25], and therefore, for a proximity measure to be effective it must be defined within the context of the intended application. Vertex affinity is designed specifically for community detection. Unlike other existing metrics, the affinity quantifies the proximity between any two vertices in a graph in terms of their *clustering strength*, which is a strong indication of thether or not they are in the same community. This makes the affinity an ideal vertex proximity measure for community detection, since it enables a community detection algorithm to select the vertices that are most likely to be in the same community. A framework to efficiently compute the vertex affinity is also developed in [28].

A simple affinity-based edge weighting scheme is developed in this work. Three simple methods to realize the clustering strength between two vertices are used in the edge weighting scheme. We applied the edge weighting scheme to the so-called college football graph, an extensively studied real-world graph with clearly defined ground truth, and evaluated the performance of a state-of-art community detection algorithm on the weighted graph. Our results show that the simple affinity-based weighting scheme can improve the performance of the community detection algorithm by as much as 30% in common modularity metric. We also show that the edge weighting scheme enhances the community detection quality even for graphs with fuzzy community structures.

The paper is organized as follows. We provide preliminaries in Section 2. Vertex affinity and its computation framework are described in Section 3. The use of the affinity-based edge weighting scheme with the popular Louvain algorithm is presented in Section 4. The results from the empirical study are discussed in Section 5. Conclusions are drawn in Section 6.

2 Preliminaries

Graph $G = (V, E)$ consists of a set of vertices and a set of edges, denoted by V and E, respectively. Given an edge $e = (u, v) \in E$, vertex v is said to be *adjacent* to vertex u and vertices u and v are said to be *incident* to the edge e (an edge $e = (u, v)$ is also denoted by $e(u, v)$). A set of vertices that are adjacent to vertex u is called the *neighbors* of u and denoted by N_u. If each edge represents an ordered pair of vertices, the graph is called *directed*. Otherwise, it is called *undirected*. A graph is *complete* if any two vertices in the graph are connected by an edge.

The vertices and edges also can have weights, where the weight of a vertex v and an edge $e(i, j)$ are denoted by w_v and w_{ij}, respectively. A graph that has weighted edges is called a *weighted graph*. An *unweighted* (or *binary*) graph is a graph in which all edges have the unit weight of 1.

A *path* from a *source* vertex s to a *destination* vertex t is a sequence $<v_0, v_1, \ldots, v_k>$ such that $s = v_0$, $t = v_k$ and $(v_{i-1}, v_i) \in E$, for $i = 1, 2, \ldots, k$ and is denoted by $p(s, t)$, The *length* of the path in an unweighted graph is the number of edges in the path (k). For a weighted graph, the length is the sum of the weight of the edges in the path. Two paths are *independent* if they do not share common vertices except the source and destination vertices. *Minimum independent shortest paths* between s and t refer to a set of all possible independent shortest paths between s and t that gives the minimum total weight sum.

Graphs, either weighted or unweighted, can be represented as a matrix. The *adjacency matrix* of a given graph $G = (V, E)$ is a $|V| \times |V|$ matrix $M = (m_{ij})$ such that $m_{ij} = w_{ij}$ if $(i, j) \in E$ and $m_{ij} = 0$ otherwise. By $d_i = \sum_{j=1}^{|V|} w_{ij}$, we denote the *degree* of a vertex i.

3 Vertex Affinity for Community Detection

A new vertex proximity concept called *affinity* for community detection was introduced recently [28]. The design of the affinity is motivated by an observation that using an improper proximity metric for solving a graph analytics problem often yields poor results [25] and therefore, for any proximity measure to be effective it must be defined within the context of the intended applications. For community detection, the proximity between vertices should be measures in terms of their *clustering strength*, which is a strong indication of those vertices belonging to the same community. Measuring the clustering strength accurately and efficiently is a key challenge.

The clustering strength between any two vertices in given graph should be high if they are connected by many short paths, because if any two vertices are in the same community, they are likely to be connected to many common vertices in the same community via short paths. Furthermore, the overall clustering strength between two vertices is affected by that of those vertices on the paths connecting them, because if two vertices are in the same community, the vertices on the paths connecting them are also likely to be in the same community and hence should also have high clustering strength. The affinity is defined in a way to capture these aspects of the clustering strength.

Vertex affinity is defined as follows. Denoting the clustering strength of an edge as $ec(e(u, v))$, the the *resistance* of an edge e, denoted by $r(e)$, is defined as

$$r(e) = 1/ec(e). \tag{1}$$

Given a path $p = <v_0, v_1, \cdots, v_{k-1}>$, the *path resistance* for a path p is defined as

$$R(p) = \sum_{i=1}^{k-1} r(e(v_{i-1}, v_i)). \tag{2}$$

Let $P(s, t)$ denote the set of minimum independent shortest paths between s and t. The *affinity* between vertices s and t, denoted by $A(s, t)$, is defined as

$$A(s, t) = \frac{1}{\sum_{p \in P(s,t)} \frac{1}{R(p)}}. \tag{3}$$

Readers should note that we omit detailed discussion of the edge clustering strength from the definition of the affinity. That is because the definition and computation of the clustering strength of an edge vary greatly depending on data and intended applications. Here, we adopt three simple methods to quantify the clustering strength of an edge and refer to the quantified clustering strength of the edge as the *edge clustering coefficient*. These methods are designed to be simple, ease to compute, and accurate. Furthermore, they only consider locally available information to compute the edge clustering coefficients. This not only reduces the computational overhead, but can improve the accuracy of a community detection algorithm, as vertices in a community typically have very short geodesic distance.

The first method is based on the standard vertex clustering coefficient, which is a measure of the clustering strength of vertex [26]. We chose the vertex clustering coefficient as basis because it is easy to compute and can measure the clustering strength of a vertex with its neighbors fairly accurately. The clustering coefficient of vertex v, $cc(v)$, is defined as

$$cc(v) = \frac{|\{e(u, w) : u, w \in N_v, e(u, w) \in E\}|}{d_v \cdot (d_v - 1)}. \tag{4}$$

We simply take the average of the vertex clustering coefficients of the endpoints of an edge as its edge clustering coefficient. That is, the clustering coefficient of an edge $e(u, v)$, $ecc(e(u, v))$ $= \frac{cc(u)+cc(v)}{2}$. We call this approach the `Average` method.

The next edge clustering coefficient calculation method is called the `Joint` method. Here, the edge clustering coefficient $e(u, v)$ is defined as

$$ecc(e(u,v)) = \frac{|\{(x,y)|x,y \in J, e(x,y) \in E\}|}{|J| \cdot (|J|-1)}, \quad (5)$$

where $J = N_u \cup N_v$ is the union of the vertices adjacent to either u or v. That is, in the `Joint` method the edge clustering coefficient of an edge $e(u, v)$ is defined as the density of a subgraph formed by the vertices that are adjacent to either u or v.

The last approach we adopt for the computation of the edge clustering coefficient is what we call `Triangular` method and was proposed in [20]. Basically, the `Triangular` method calculates the clustering coefficient of an edge as the ratio of the number of triangles that contain the given edge to the number of all possible triangles that can be constructed on given edge. More formally, the edge clustering coefficient $ecc(e(u, v))$ is

$$ecc(e(u,v)) = \frac{z_{u,v}+1}{min(d_u-1, d_v-1)}, \quad (6)$$

where $z_{u,v}$ is the number of triangles that contain the edge e. We discuss the computation of the affinity in more detail in following.

Assuming that each edge in given graph is assigned its edge clustering coefficient, its inverse, called edge *resistance* is assigned to each edge as its weight. This is necessary to ensure that the affinity between two vertices decreases numerically as the clustering strength between them increases. Once the resistance of each edge is determined, then the given graph is reduced to a network of resistors, each of which is a measure of the affinity. The affinity between any two vertices can be modeled as the total resistance between them as shown in Equation (3). This can be computed using Kirchoff's circuit laws, but in our framework a heuristic approach is adopted for its computation, because solving the Kirchoff's laws is computationally expensive, and more importantly, this heuristic approach tends to limit the ill effect of remote affinity values.

Vertex affinity is computed by using the minimum independent shortest paths. Finding minimum independent shortest paths is NP-hard, so a greedy method is used. In this heuristic, given two vertices s and t, the shortest path with minimum path resistance is found using the Dijkstra's algorithm [7]. Then, all the vertices on this path except s and t are marked invalid. Dijkstra's algorithm is invoked again to find next shortest path that consists of only valid vertices. This process is repeated until there exists no path connecting s and t. This set of independent shortest paths conceptually forms a combination of series and parallel resistance circuits. Its total resistance can be computed by simple circuit theory formulas as specified in Equations (2) and (3) and represents $A(s, t)$. Pseudo code for the framework is given in Algorithm 1.

The strengths of the vertex affinity and its computation framework are as follows.

1. Since vertex affinity measures the proximity between vertices in terms of their clustering strength, it provides proximity values that are closely related to community detection, in contrast to other existing metrics. Therefore, any community detection algorithms

Algorithm 1 Algorithm for Affinity Computation

```
1: Input: Graph $G = (V, E)$, where $V$ and $E$ are a set of vertices and edges, and two vertices
   $s$ and $t$ such that $s$, $t \in V$
2: Output: Affinity between $s$ and $t$, $A(s, t)$

3: for $\forall e = (u, v) \in E$ do
     4.1: Compute edge clustering coefficient $ecc(e)$
     4.2: Compute edge resistance $r(e) = ecc(e)^{-1}$
   end for
4: $P = \emptyset$
5: Find the shortest path $p = <s, v_1, \ldots, v_{k-2}, t>$ using Dijkstra's algorithm
6: while $p \neq$ null do
     7.1: $P = P \cup p$
     7.2: $\forall v \in p$ such that $v \neq s, t$, $V = V - v$
     7.3: Find the shortest path $p = <s, v_1, \ldots, v_{k-2}, t>$ using Dijkstra's algorithm
   end while
7: $l = 0$
8: for each $p_i \in P$ do
     9.1: Let $p_i = <v_0, v_1, \ldots, v_{k-2}, v_{k-1}>$, where $v_0 = s$ and $v_{k-1} = t$
     9.2: Compute $R(p_i) = \sum_{j=1}^{k-1} r((v_{j-1}, v_j))$, where $(v_{j-1}, v_j) \in E$
     9.3: $l = l + R(p_i)^{-1}$
   end while
9: $A(s, t) = l^{-1}$
```

that takes an input graph with affinity-based weights can find community structures more accurately.

2. The affinity is extensible and flexible in that any measures of edge clustering strengths can be used, depending on the intended graphs and specific applications, in computing the affinity values. Due to such extensibility and flexibility, the affinity measures can be applied to solving a wide range of graph analytics problems.

3. The computation of the affinity values is relatively inexpensive compared to other metrics, as they can be computed via simple heuristic approach.

4 Community Detection by Louvain Algorithm with Affinity-based Edge Weighting

We claim that the performance of many existing community detection algorithms can be enhanced substantially by assigning each edge a weight derived from the vertex affinity. For validation, we conducted comprehensive empirical research on the performance impact of the affinity-based weighting.

Community detection algorithm needs to be able to process weighted graphs in order to take advantage of the weighting scheme. Many community detection algorithms that were originally designed for binary graphs, however, can be easily modified to handle weighted graphs. For example, algorithms based on random walks, such as recently proposed *DEMON* algorithm [8], can be modified for weighted graphs by adjusting the random walk probability to be proportional to the edge weights.

We selected a state-of-art community detection algorithm, commonly known as the *Louvain* algorithm [2] as baseline algorithm in our experiments. We chose this algorithm mainly because it is simple, fast, and accurate. Due to these advantages, the Louvain algorithm has been accepted as a de facto standard algorithm for community detection in the research community and widely used for solving various applications. Moreover, due to the highly accurate results obtainable with the Louvain algorithm, it is difficult to even marginally outperform this algorithm.

The Louvain algorithm is basically a greedy method that finds communities by finding a local maxima of an objective function based on the *modularity* [14, 18], which provides a density-based measure on the quality of a community. It is also agglomerate, as it finds final communities by merging vertices. The Louvain algorithm works as follows.

Given a graph, the algorithm starts with singleton communities, each of which contains a single vertex as its member. The algorithm proceeds through a series of passes, within which the algorithm iterates through two phases repeatedly. In the first phase, for each vertex the algorithm considers all of its neighbors and computes the modularity gain that can be obtained by moving the vertex from its current community to the community of one of its neighbors. The vertex is moved to a community that maximizes the modularity gain, but only when the gain is positive. During each move, the vertex is removed from its current community and merged to the new community. This process is repeated for all vertices in sequence until no vertex can be moved. In the second phase, the algorithm constructs a new graph whose vertices are the communities computed in the first phase. The edges are assigned new weights that are the sum of the weights of the edges between vertices in the computed communities. Once the new graph is constructed, then the first phase of the algorithm can be applied to the newly built graph in the next pass. The passes are iterated until the maximum modularity obtained cannot be improved by more than a given threshold value.

In our affinity-based edge weighting scheme, we assign each edge a new weight that is derived from the affinity between its two endpoints. The edge weighting is performed on the unweighted input graph only once before the Louvain algorithms is invoked. Since the Louvain algorithm finds community structures by maximizing its objective function (i.e., modularity), an edge with higher clustering strength, which translates into a smaller affinity value, should be given a larger weight. We adopt a straightforward approach to compute the edge weights: taking the inverse of the edge's affinity value. More formally, we compute the weight of an edge $e(u,v)$, $w(e(u,v))$, as

$$w(e(u,v)) = \frac{\beta}{A(u,v)^{\alpha}}, \tag{7}$$

where $A(u,v)$ denotes the affinity between endpoints u and v, and α and β are given parameters. In this method, the parameters α and β control the effect of an edge weight. In this study, however, we simply use the reciprocal of affinity values (i.e., $\alpha = \beta = 1$), because finding proper α and β values for optimal performance is highly data-dependent.

5 Experimental Results

We demonstrate that the performance of community detection algorithms can be enhanced substantially by the simple affinity-based edge weighting scheme discussed in Section 4 through an empirical study, where we evaluate the performance of the Louvain algorithm with the edge weighting scheme. In this evaluation, we use what is commonly known as the *NCAA Division I-A Football* graph, a well-studied real-world graph. We chose this graph mainly because it offers clearly defined ground truth communities, and therefore, we can objectively evaluate the quality of communities computed by community detection algorithms. The graph consists of 115 vertices (teams) and 616 undirected edges. Each edge in this graph indicates that the two teams represented by its endpoints have played a game with each other. The teams and the conferences they belong to are described in Table 1, where the teams and the conferences are identified by unique IDs.

Tables 2 to 5 report the results from the comparison of the computed communities and the ground truth for different affinity calculation schemes. The accuracy of the computed communities is measured by standard F-score [23]. For clarity, we only report positive F-score values in the tables. An F-score value of 1.0 indicates a perfect match between the computed communities and the ground truth communities. A set of communities computed by any community detection algorithms is considered ideal if it satisfies certain criteria. First, the closer the number of communities computed is to the number of communities in the ground truth, the higher the accuracy of the computation is. We also consider the quality of computed communities to be high if the comparison of the computed communities to the ground truth communities yields high statistical similarity measure. Finally, if the input graph contains a community of outliers, then the comparison of this community to the ground truth communities should return low statistical similarity values.

We first evaluate the performance of the Louvain algorithm with the original unweighted football graph and report the results in Table 2. Overall, the communities identified by the Louvain algorithm are relatively accurate, as indicated by high F-score values, with the exception of the ones in the last row of the table. The last community (11) in the ground truth is comprised of 8 independent teams. Since these teams do not belong to any conferences, any subgraphs they form lack strong community structure, and therefore, it is extremely difficult to group them together as a separate community. In fact, it can be considered that the Louvain algorithm correctly labels the independent teams as outliers by assigning them relatively low F-scores. The fact that such high accuracy is obtainable with the Louvain algorithm makes it very hard to even marginally outperform this algorithm.

However, the Louvain algorithm also has its limitations. As shown in the table, the Louvain algorithm identified 10 communities. Although this number is close to the actual number of communities in the ground truth, only six of them are perfect matches. Furthermore, some of the computed communities do not have high F-score numbers when compared to the ground truth communities, indicating that they do not have strong community structures. For example, the fourth computed community (3) is similar to two ground truth communities, 4 and 7, as both entries have relatively high F-score values, 0.60 and 0.73 respectively. Also, the table shows that the independent conference (11) is similar to one of the computed communities (5) with the F-score number of 0.40. Since this computed community is likely to be the ground truth conference 9 (Southeastern Conference) as indicated by high F-score value of 0.83, the F-score for the ground truth conference 11 should be very low.

We repeat the above experiment with the football graph with edge weighting. With the edge weighting, each edge in the original graph is assigned a new weight computed by the method described in Equation (7). This implies that the weighted graph can have edges that are not present in the original graph (that is, the weighted graph becomes a complete graph) as we can measure the affinity between any nonadjacent vertices using our framework. The results are reported in Tables 3 to 5.

The accuracy of the communities detected by the `Average` affinity computation method is evaluated in Table 3. Like the unweighted case, the Louvain algorithm found 10 communities, six of which are perfect matches, for the weighted graph. Also, the comparison of the community of independent teams (community 11) exhibits low F-scores, indicating that none of the computed communities match to these outliers. However, with the weighted graph the F-score values for the last row (for the independent teams) are smaller than those for the unweighted graph. This indicates that the Louvain algorithm can find better-structured communities with the weighted graph. The F-score obtained by using the `Joint` affinity calculation method is almost identical to those from the `Average` method. This is probably due to the very strong community structures in the football graph which results in these two affinity calculation methods returning affinity values that are proportional to each other.

Table 5 shows the F-score results for the weighted football graph, when the `Triangular` affinity computation method is used. The results in the table show that this edge weighting enhances the performance of the Louvain algorithm considerably. First, the number of communities detected is in perfect accordance with the ground truth, since the Louvain algorithm found 11 communities (considering the independent teams as outliers). Furthermore, the communities detected are highly accurate and close to the ground truth. The Louvain algorithm identified eight communities correctly as indicated by F-score values of 1.0. Even the imperfect communities are very close to real-world communities as they has very high F-scores. Just like in the previous experiment, the outlier community of independent teams (11) have low F-scores with the computed communities.

Table 6, in which the global modularity of the graph formed by the final communities [2] are listed, provides strong evidence that the affinity-based edge weighting improves the performance of the Louvain algorithm considerably. As can be seen in the table, the modularity of final communities improves by 29% when the graph is weighted with the `Triangular` method, compared to the unweighted case. Measuring the quality of communities using the modularity measure is not always valid, since the modularity tends to increase as the number of communities decreases. In this case, however, the number of communities detected for the weighted graph is greater than that for unweighted graph, validating the modularity results.

As stated earlier, the college football graph has been widely studied in the literature mainly due to the availability of ground truth. With this graph, however, it is challenging to evaluate the effectiveness of community detection schemes, because the graph has very well-defined community structure that is easier to detect than that in other real-world graphs. In following experiments we perturb the community structures in this graph by adding and/or removing some random edges in order to make the communities harder to find, so that we can evaluate how the affinity-based edge weighting scheme performs in more realistic settings. The results are presented in Figures 1 to 3.

Figure 1 presents the changes in modularity of the computed communities as more edges are added to the original graph. Adding random edges to a graph dilutes the community structure in the graph and hence, makes it harder to detect. This is clearly shown in the Figure 1, where the overall modularity values decrease as more edges are added. The performance of the Louvain algorithm without weighting suffers most. As shown in the figure, the modularity of computed communities decreases from 0.6 to 0.38 (roughly 38% reduction) for unweighted graphs when 616 new edges (exactly 50% of the number of edges in original graph) are added. With the weighting, on the other hand, the Louvain algorithm obtains higher overall modularity. For the graph weighted with affinity computed by `Average` method, the Louvain algorithm maintains higher modularity compared to the unweighted graph. Edge weighting by the `Joint` method exhibits the similar performance as the one with `Average` method.

Edge weighting by the `Triangular` method achieves the best performance. The modularity obtained by this weighting scheme is as much as 35% higher than the unweighted case as shown in the figure.

Similarly, Figure 2 shows the modularity of the computed communities as more edges are removed from the original graph. Removing edges from a graph does not have as significant impact on the modularity as does adding edges, especially for well-structured graphs like the football graph, since quite a significant number of edges need to be removed in order to destroy the existing community structures [17]. This can be clearly seen in Figure 2, where the modularity values do not vary as much as in Figure 1. For the unweighted graph, removing edges actually improves the modularity in some cases, probably because the removal of some edges reduces noise while maintaining the community structure. The figure shows that the affinity-based weighting via the `Triangular` method still achieves the best performance. With this weighting

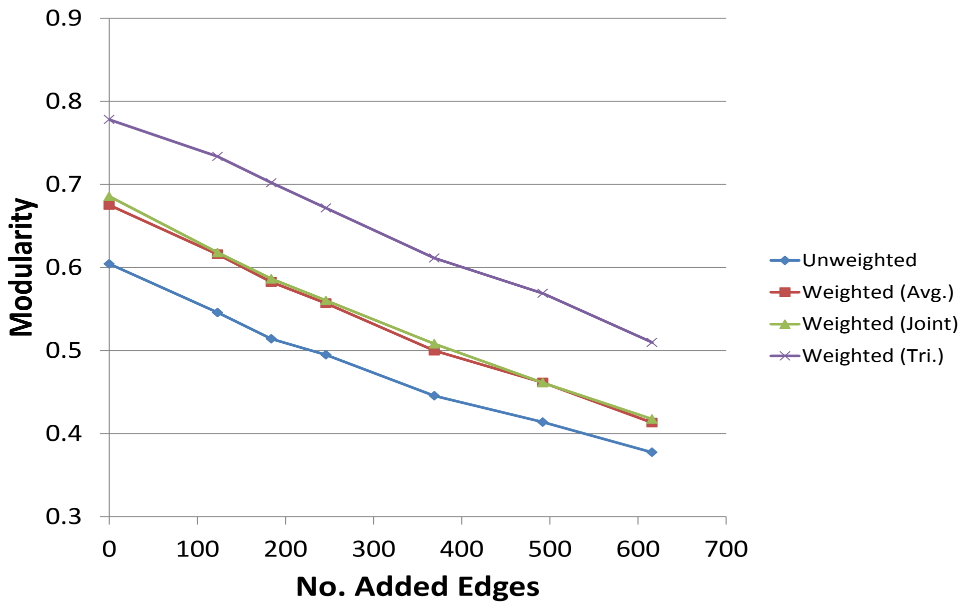

Figure 1: The modularity of communities computed by the Louvain algorithm for unweighted and weighted football graphs to which a certain number of edges are added randomly. The graph is then fully weighted, resulting in a complete weighted graph.

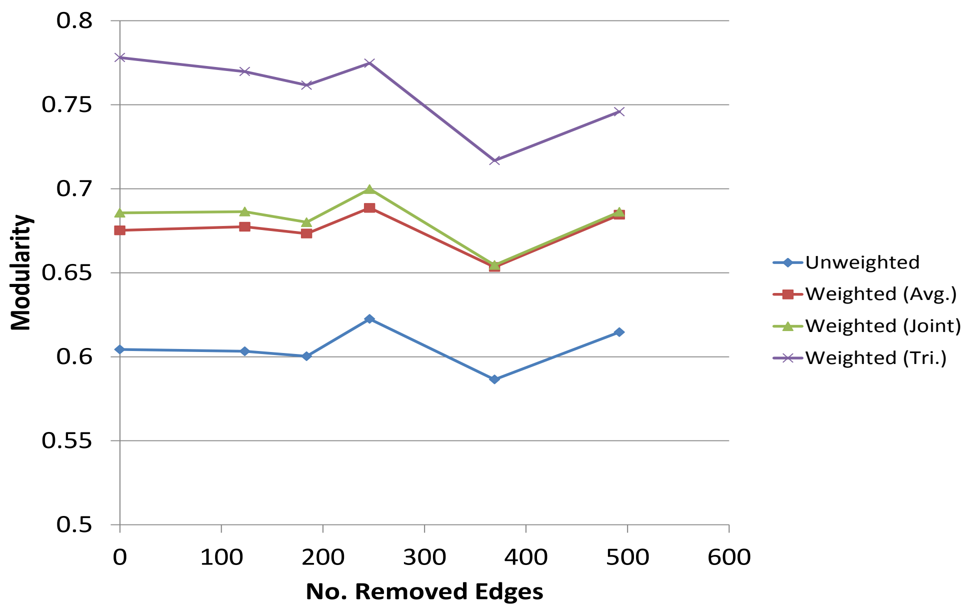

Figure 2: The modularity of communities computed by the Louvain algorithm for the unweighted and weighted football graphs from which a certain number of edges are removed randomly. The graph is the fully weighted, resulting in a complete weighted graph.

scheme, the Louvain algorithm improves the overall modularity by at least 20%.

In the next experiment, we modify the graph by adding new random edges, while removing the exactly same number of edges so at the end the graph has the same number of edges as the original graph. The combination of addition and removal of edges expedites the deconstruction of the community structures in the graph and hence should result in poorer modularity results. This can be clearly seen in Figure 3, where the modularity curves decline more rapidly than the case where edges are only added. However, the Louvain algorithm with the edge weighting scheme still outperforms the one without the edge weighting. The edge weighting by the `Triangular` method still achieves the best performance, improving the modularity by as much as 34% compared to the unweighted graph.

6 Conclusions

The performance of community detection algorithms is limited by the inherent constraint of the fact that unweighted graphs offer very little information on their internal community structures. In this paper, we propose a new scheme that weights the edges in a given graph based on the recently proposed vertex affinity concept to overcome this limitation. Unlike other existing

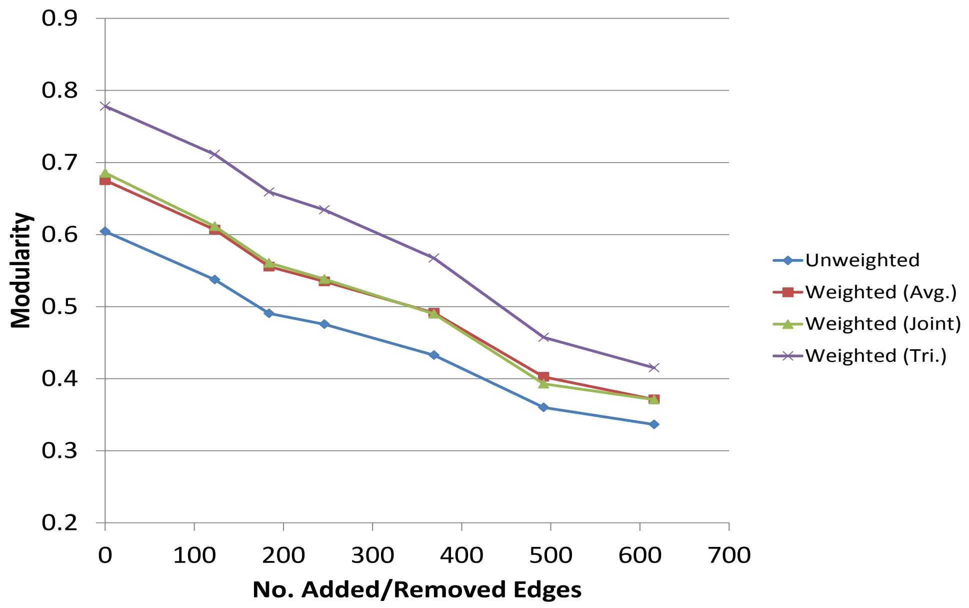

Figure 3: The modularity of communities computed by the Louvain algorithm for unweighted and weighted football graphs, where a certain number of edges are added and removed randomly. Here, the same number of edges are added and removed.

metrics, the affinity quantifies the proximity between any two vertices in a graph in terms of their clustering strength. This makes it an ideal vertex proximity measure for community detection, because it helps a community detection algorithm to select the vertices that are most likely to be in the same community.

We propose a simple edge weighting scheme that derives edge weights from the vertex affinity measures. We applied the edge weighting scheme to the college football graph, an extensively studied real-world graph with clearly defined ground truth. We ran the state-of-art Louvain algorithm on the test graph preprocessed by the proposed edge weighting scheme and measured the performance. The results show that the affinity-based edge weighting scheme can improve the modularity of final communities computed by the Louvain algorithm by as much as 30%. Further, we also show that the edge weighting scheme enhances the community detection quality even for graphs with fuzzy community structures.

References

[1] B. Adamcsek, G. Palla, I. J. Farkas, I. Derényi, and T. Vicsek. Cfinder: locating cliques and overlapping modules in biological networks. *Bioinformatics*, 22(8):1021–1023, 2006.

[2] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of community hierarchies in large networks. *CoRR*, abs/0803.0476, 2008.

[3] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner. On finding graph clusterings with maximum modularity. In *Proceedings of the 33rd International Conference on Graph-theoretic Concepts in Computer Science*, WG'07, pages 121–132, Berlin, Heidelberg, 2007. Springer-Verlag.

[4] D. Chakrabarti. Autopart: parameter-free graph partitioning and outlier detection. In *PKDD '04: Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 112–124, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[5] A. Clauset. Finding local community structure in networks. *Physical Review E*, 72:026132, 2005.

[6] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70(6):066111, Dec. 2004.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, Cambridge, MA, second edition, 2001.

[8] M. Coscia, G. Rossetti, F. Giannotti, and D. Pedreschi. Demon: A local-first discovery method for overlapping communities. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 615–623, New York, NY, USA, 2012. ACM.

[9] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas. Comparing community structure identification. *Journal of Statistical Mechanics*, Sept. 2005.

[10] I. Derényi, G. Palla, and T. Vicsek. Clique Percolation in Random Networks. *Physical Review Letters*, 94(16):160202–+, Apr. 2005.

[11] J. Duch and A. Arenas. Community detection in complex networks using extremal optimization. *Physical Review E*, 72:027104, Jan. 2005.

[12] Facebook. `https://developers.facebook.com/docs/reference/opengraph`.

[13] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75 – 174, 2010.

[14] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proc. Natl. Acad. Sci.*, 99:7821, 2002.

[15] S. Gregory. Finding overlapping communities in networks by label propagation. *New Journal of Physics*, 12(10), October 2010.

[16] R. Guimera, M. Sales-Pardo, and L. A. N. Amaral. Modularity from fluctuations in random graphs and complex networks. *Physical Review E*, 70:025101, 2004.

[17] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining (KDD '05)*, pages 177–187, 2005.

[18] M. E. J. Newman. Modularity and community structure in networks. *Proc. Natl. Acad. Sci.*, 103:8577, 2006.

[19] C. Pizzuti, S. E. Rombo, and E. Marchiori. Complex detection in protein-protein interaction networks: A compact overview for researchers and practitioners. In *Proceedings of the 10th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, EvoBIO'12, pages 211–223, Berlin, Heidelberg, 2012. Springer-Verlag.

[20] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. *Proc. Natl. Acad. Sci.*, 101:2658, 2004.

[21] J. Reichardt and S. Bornholdt. Statistical mechanics of community detection. *Physical Review E*, 74:016110, 2006.

[22] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, Aug. 2000.

[23] C. J. van Rijsbergen. *Information Retrieval.* Butterworth, 1979.

[24] S. Vigna, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks, 2010.

[25] U. von Luxburg, A. Radl, and M. Hein. Hitting times, commute distances and the spectral gap for large random geometric graphs. *CoRR*, abs/1003.1266, 2010.

[26] D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393:440–442, 4 June 1998.

[27] J. Xie and B. K. Szymanski. Community detection using a neighborhood strength driven label propagation algorithm. In *Proceedings of the 2011 IEEE Network Science Workshop*, NSW '11, pages 188–195, Washington, DC, USA, 2011. IEEE Computer Society.

[28] A. Yoo, G. Sanders, V. Henson, and P. Vassilevski. A novel vertex affinity for community detection. Technical report, Center for Applied Scientific Computation, Lawrence Livermore National Laboratory, 2014.

Table 1: Description of the graph for the NCAA Football conferences and conference members in year 2000. There are 115 vertices (teams) in 12 conferences in the graph, including a group of independent teams. Each team is represented by a unique ID. ID for conferences are given in paranthesis.

Southeastern (9)		**Big 12 (1)**		**Mid American (6)**	
27	Florida	40	Colorado	99	Marshall
95	Georgia	72	Iowa State	18	Akron
70	South Carolina	52	Kansas	71	Ohio
76	Tennessee	3	Kansas State	61	Miami (Ohio)
62	Vanderbilt	102	Missouri	31	Bowling Green State
56	Kentucky	74	Nebraska	34	Buffalo
17	Auburn	107	Oklahoma State	54	Kent State
96	LSU	98	Texas	14	Western Michigan
87	Ole Miss	81	Texas A&M	85	Toledo
65	Mississippi State	10	Baylor	12	Northern Illinois
113	Arkansas	84	Oklahoma	26	Ball State
20	Alabama	5	Texas Tech	43	Eastern Michigan
				38	Central Michigan
Big 10 (3)		**Pac 10 (8)**		**Conference USA (5)**	
32	Michigan	51	Washington	57	Louisville
39	Purdue	108	Oregon State	44	East Carolina
13	Northwestern	68	Oregon	92	Cincinnati
47	Ohio State	77	Stanford	75	Southern Mississippi
15	Wisconsin	21	UCLA	112	Alabama Birmingham
60	Minnesota	8	Arizona State	86	Tulane
6	Penn State	22	Arizona	66	Memphis
2	Iowa	7	Southern California	48	Houston
64	Illinois	78	Washington State	91	Army
100	Michigan State	111	California		
106	Indiana				
ACC (0)		**Big East (2)**		**Western Athletic (10)**	
1	Florida State	101	Miami (Florida)	110	TCU
103	Clemson	19	Virginia Tech	83	UTEP
37	Georgia Tech	55	Pittsburgh	46	Fresno State
33	Virginia	35	Syracuse	73	SanJose State
25	North Carolina State	29	Boston College	88	Tulsa
89	North Carolina	30	West Virginia	49	Rice
109	Maryland	79	Temple	114	Hawaii
105	Wake Forest	94	Rutgers	53	SMU
45	Duke			67	Nevada
Mountain West (7)		**Independent (11)**		**Big West (4)**	
41	Colorado State	82	Notre Dame	28	Boise State
93	Air Force	36	Central Florida	90	Utah State
104	UNLV	63	Middle Tennessee State	50	Idaho
0	Brigham Young	42	Connecticut	69	New Mexico State
4	New Mexico	58	Louisiana Tech	11	North Texas
23	Utah	97	Louisiana Lafayette	24	Arkansas State
9	San Diego State	59	Louisiana Monroe		
16	Wyoming	80	Navy		

Table 2: Comparison of the communities computed by the Louvain algorithm with the ground truth communities in terms of the standard F-scores for the 2000 NCAA college football graph described in Table 1. The original graph is unweighted. Only positive F-score values are reported in the table for clarity.

Ground Truth	Computed Communities									
	0	1	2	3	4	5	6	7	8	9
0		1.00								
1					1.00					
2									0.89	
3	1.00									
4				0.60						
5										1.00
6							0.96			
7				0.73						
8								1.00		
9						0.83				
10			1.00							
11						0.40	0.09		0.22	

Table 3: Comparison of the communities computed by the Louvain algorithm with the ground truth communities in terms of standard F-scores for the 2000 NCAA college football graph described in Table 1. Only positive F-score values are reported in the table for clarity. The graph used in this experiment is a complete graph that is constructed from the affinity matrix computed for the original graph. The affinity measures are calculated by the `Average` method in this experiment.

Ground Truth	Computed Communities									
	0	1	2	3	4	5	6	7	8	9
0			1.00							
1									1.00	
2							0.89			
3	1.00									
4					0.60					
5						1.00				
6		0.93								
7					0.73					
8										1.00
9								0.86		
10				1.00						
11		0.17					0.22	0.33		

Table 4: Comparison of the communities computed by the Louvain algorithm with the ground truth communities in terms of standard F-scores for the 2000 NCAA college football graph described in Table 1. Only positive F-score values are reported in the table for clarity. The graph used in this experiment is a complete graph that is constructed from the affinity matrix computed for the original graph. The affinity measures are calculated by the `Joint` method in this experiment.

Ground	Computed Communities									
Truth	0	1	2	3	4	5	6	7	8	9
0			1.00							
1									1.00	
2							0.89			
3	1.00									
4					0.60					
5						0.82				
6		0.93								
7					0.73					
8										1.00
9								1.00		
10				1.00						
11		0.17				0.38	0.22			

Table 5: Comparison of the communities computed by the Louvain algorithm with the ground truth communities in terms of standard F-scores for the 2000 NCAA college football graph described in Table 1. Only positive F-score values are reported in the table for clarity. The graph used in this experiment is a complete graph that is constructed from the affinity matrix computed for the original graph. The affinity measures are calculated by the `Triangular` method in this experiment.

Ground	Computed Communities										
Truth	0	1	2	3	4	5	6	7	8	9	10
0						1.00					
1		1.00									
2									0.89		
3	1.00										
4					1.00						
5								1.00			
6				0.93							
7			1.00								
8										1.00	
9											0.86
10							1.00				
11				0.17					0.22		0.33

Table 6: The comparison of the modularity of the communities computed by the Louvain algorithm. The modularity values are computed for original unweighted graph as well as complete graphs, weighted by using different affinity calculation methods.

Weight Scheme		Modularity
Unweighted		0.604346
Edge Weighted	`Average`	0.67525
	`Joint`	0.685688
	`Triangular`	0.778116