CS 224W Final Project: Comparing Performance Across Paradigms of Community Detection in Bipartite Networks

Max Bodoia (mbodoia), Laura Griffiths (laurajg), Arjun Puranik (apuranik)

I. INTRODUCTION

The problem of detecting community structure in networks has recently received a great deal of attention in the scientific community. Many different kinds of algorithms have been proposed to solve this problem, ranging from hierarchical clustering to modularity-based methods. The majority of these methods make few assumptions about the underlying structure of the network in order to be as general as possible. However, in some instances the network under consideration is known to exhibit a certain structure, and in these instances it seems natural to look for a community-detection algorithm that can take advantage of this structure.

One particular kind of network that often appears in real-world situations is a bipartite network. Examples of such networks include scientific publication networks (where nodes represent authors/papers and edges represent authorship), document networks (where nodes represent documents/words and edges represent the presence of a word in a document), and purchasing networks (where nodes represent users/products and edges represent purchases). Furthermore, when dealing with such networks we are often only interested in the communities present on one side of the network (e.g. paper topics in scientific publication networks or product categories in purchasing networks). Finding the best method for detecting communities on one side of a bipartite network is therefore a topic of both practical and theoretical interest.

Much prior work has been done on community detection in bipartite networks. Broadly speaking, existing methods can be said to fall under one of three "paradigms:"

1. Apply a generic community-detection algorithm to the full bipartite network ([1], [2], [3]).
2. Apply a community-detection algorithm designed specifically for bipartite networks to the full bipartite network ([4], [5]).
3. Construct an induced unipartite network on the side of the bipartite network that is of interest, then apply a generic community-detection algorithm to this induced network ([3], [5]).

However, despite the variety of methods available for community detection in bipartite networks, relatively little research has been done to compare the effectiveness of these methods. Some authors ([5] and [4]) allude to existence of different paradigms, and others ([3]) provide side-by-side comparisons of their algorithm with other alternatives within the same paradigm. But none of the papers go so far as to compare their results with the results of methods from other paradigms. Furthermore, the few preliminary results and comparisons that do exist in the literature often use "toy" networks of only a few hundred nodes ([1], [2] and [5] test their algorithms on networks with only a few hundred nodes at most). So we are left with three broad approaches to community detection in bipartite networks, and no definite answer as to which approach should be followed for any particular application.

Our project will attempt to shed light on such an answer. Specifically, we will address the following question: given a large, real-world bipartite network with ground-truth communities, which of the three paradigms listed above is most successful at recovering these communities? We plan to answer this question by applying each paradigm to a novel dataset provided by Bitly.

In section 2 we will give more detailed characterizations of the various methods mentioned above. In section 3, we describe the novel Bitly dataset to which we will apply these methods. In section 4, we explain the metrics we use to evaluate the performance of the methods. In section 5, we present the results of our evaluation along with some high-level analysis, then conclude in section 6 by exploring the implications of these results for future work.

II. PARADIGMS

A. Paradigm 1

The first paradigm of community detection in bipartite networks represents the "naive" approach in which no attempt is made to leverage the bipartiteness of the graph. That is, methods in this paradigm apply a generic community detection algorithm to the bipartite network in question. We consider four such methods.

1. Hierarchical Clustering (1-Hierarchical)

The first method of paradigm 1 is Hierarchical Clustering (1-Hierarchical). Newman and Girvan ([1]) introduce this method in their 2001 paper as "the traditional method for detecting community structure in networks." To implement hierarchical clustering, one must first define a weight W_{ij} between every pair of nodes i and j in the network. If the dataset naturally lends itself to being modeled as a network with weighted edges then these weights can be used. Otherwise, the weights are defined in terms of various statistics about the network. Newman and Girvan offer a few examples of how weights are commonly set. After experimentation, in our implementation, we chose to define weight between a pair of nodes in our unweighted bipartite network as the length of the shortest path between them which can be calculated for any two nodes using BFS. In our weighted unipartite network we take into account the weight of the edge and so define the weight of a pair of nodes i and j as the maximum flow which can pass from node i to node j. We chose these definitions because they have no parameters which need to be adjusted (unlike other common weight functions).

Once the weights have been set, the hierarchical clustering algorithm can assign nodes to clusters. The algorithm

is initialized with a distinct cluster for each node. At each step, the algorithm chooses the pair of nodes (i, j) with highest weight (excluding pairs already chosen) and merges the cluster containing node i with the cluster containing node j. This iterative merging process will eventually produce one giant cluster which contains all nodes in the network. However, since we wish to detect multiple communities within the network, we halt the merging process in our implementation once the total number of clusters remaining is equal to the number of ground truth communities present in the network. Putting it all together, we end up with the following procedure (1-Hierarchical):

1. Initialize weight W_{ij} for each pair of nodes i and j as either the length of the shortest path or the max-flow between i and j using BFS or existing flow algorithms (e.g. Ford-Fulkerson).
2. Initialize each node as belonging to its own cluster.
3. Iterate over the set of all node pairs i and j in descending order of weight W_{ij}. At each step, merge the cluster containing i with the cluster containing j (if distinct).
4. Terminate when the number of clusters remaining equals the number of ground-truth communities.

The time complexity of hierarchical clustering depends both on our distance measure and our clustering/merging methodology. The generic form has a time complexity of $O(n^3)$ however we can improve that using a priority-queue algorithm. Average-link and complete-link clustering, which we experimented with in our implementation, have a time-complexity of $O(n^2 \log n)$. Our final results utilize average-link clustering to offset the tendency of complete-link clustering to be highly receptive to outliers.

2. Edge-Betweenness Clustering (1-EdgeBetweenness)

The second method of paradigm 1 is edge-betweenness clustering (1-EdgeBetweenness). This method, which involves focusing on edges between clusters rather than central edges within clusters, is the primary contribution of the 2001 Newman and Girvan paper ([1]). They formalize the notion of the 'betweenness' of an edge by defining it as the number of shortest paths between pairs of vertices that pass through that edge. For pairs of vertices with multiple shortest paths between them, each path is given equal fractional weight, which it contributes to all the edges along it.

Newman and Girvan's method rests on the assumption that edges across clusters will tend to have higher betweenness than edges within clusters. They justify this assumption by noting that "if a network contains communities or groups that are only loosely connected by a few intergroup edges, then all shortest paths between different communities must go along one of these few edges" ([1]). So their algorithm identifies communities by beginning with the full network and iteratively removing edges with high betweenness (recalculating betweenness after each removal), until no edges remain. As in the case of hierarchical clustering, our implementation of the algorithm terminates once the number of connected components equals the number of ground-truth communities. This gives us the following procedure (1-EdgeBetweenness):

1. Compute the betweenness B_{ij} for every edge ij by computing the shortest paths between every pair of nodes and incrementing B_{ij} by 1 for each edge ij on each such path (or by $\frac{1}{k}$ for pairs of nodes with k equidistant shortest paths).
2. Remove the edge ij with highest betweenness B_{ij}.
3. Repeat steps (1) and (2) until the number of connected components in the network equals the number of ground-truth communities.

Each iteration of the algorithm requires time $O(|V|*|E|)$. The original algorithm we iterate until no edges remain, giving us a final time-complexity of $O(|V| * |E|^2)$. We introduce an early stopping criteria however the degree to which that improves our efficiency depends on the structure of the graph.

3. Basic Modularity-based Clustering (1-BasicModularity)

The third method of paradigm 1, basic modularity-based clustering (1-BasicModularity), is also due to Newman ([3]). This method relies on a particular measure called "modularity" which can be associated with different clusterings or partitions of the network. Roughly speaking, the modularity of a clustering corresponds to "the number of edges falling within groups minus the expected number in an equivalent network with edges placed at random" ([3]). More precisely, we define the modularity matrix B as

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$$

where A is the adjacency matrix of the network, k_i and k_j are the degrees of nodes i and j, and m is the number of edges. The modularity Q of a network is then the sum of B_{ij} over all pairs i and j contained in the same cluster.

Since we can think of modularity as representing the extent to which a partition of the network is statistically surprising, higher modularity values indicate the existence of community structure within a particular division of the network. Newman observes that we can find a partition of the network with high modularity by computing the most positive eigenvalue of B and its associated eigenvector, then assigning nodes to one of two clusters based on their corresponding sign in this eigenvector. Partitions of the network into more than two groups can be accomplished by further subdividing the initial two groups using the same method. However, in order to divide a subset g of all nodes, we must use the generalized modularity matrix $B^{(g)}$ given by

$$B^{(g)}_{ij} = B_{ij} - \delta_{ij} \sum_{k \in g} B_{ik}$$

where δ_{ij} is 1 if $i = j$ and 0 otherwise. Note that when g is the set of all nodes V, this is just the original modularity matrix B. Thus we have the following procedure (1-BasicModularity):

1. Begin with a single cluster containing all the nodes.
2. For each existing cluster g, compute the generalized modularity matrix $B^{(g)}$. Then, compute the optimal partition of this cluster by splitting the nodes into two groups according to the sign of their corresponding value in the leading eigenvector.

3. Compute the new modularity Q of the network for each subdivision of the network computed in step 2, and choose the subdivision which gives the highest new modularity.

4. Repeat steps (2) and (3) until no further subdivision can increase the modularity Q of the network.

4. Basic Spectral Clustering (1-BasicSpectral)

In the spectral clustering paradigm as described in [4], we try to find a good partitioning of a graph by minimizing the inter-cluster edges and maximizing the intra-cluster edges. In the case of a 2-partitioning (V_1, V_2), the objective is to maximize

$$\frac{\text{cut}(V_1, V_2)}{|V_1|} + \frac{\text{cut}(V_1, V_2)}{|V_2|}$$

where $\text{cut}(V_1, V_2)$ is the number of edges between V_1 and V_2. It turns out that the optimal partition is given by the eigenvector corresponding to the second largest eigenvalue of the Laplacian matrix associated with the graph. Where n is the number of nodes, the Laplacian matrix of an unweighted graph is an $n \times n$ symmetric matrix where $L_{ii} = \deg(i)$, and $L_{ij} = -1$. So the algorithm is just

1. Form the Laplacian matrix L as described above.

2. Compute or approximate the second eigenvector of L to get our partition.

The k-partitioning algorithm is similarly defined (described in full detail in [4]; we describe the full k-partitioning algorithm for the bipartite case below); except, for example, we look at the eigenvectors associated with the top $\lceil \log_2 k \rceil$ eigenvalues (starting at 2).

B. Paradigm 2

The second paradigm of community detection in bipartite networks includes methods that are specially tailored to be used on such networks.

1. Bipartite Recursively Induced Modules (2-BRIM)

The first method of paradigm 2 is the bipartite recursively induced module method (2-BRIM) introduced by Michael Barber [5]. This method is a direct extension of Newman's modularity-based algorithm [3] to bipartite graphs. Both Newman and Barber make use of a modularity matrix $B = A - P$, where A is the adjacency matrix and P is a matrix of edge formation probabilities. Newman's model defines $P_{ij} = \frac{k_i k_j}{2m}$, where k_i and k_j are the degrees of nodes i and j, in order to ensure that the expected degree of each node under P is equal to its actual degree. Barber specializes this null model for bipartite graphs by further requiring that $P_{ij} = 0$ for nodes i and j on the same side of the bipartite network, which ensures that the edge formation probability for edges which violate the bipartite property of the network is 0. Using this modified edge formation probability matrix, Barber derives a bipartite-specific modularity matrix defined by

$$\bar{B}_{ij} = \frac{k_i d_j}{m}$$

where k_i denotes the degrees of the p nodes on one side of the bipartite graph and d_j denotes the degrees of the q nodes on the other side.

The partition which maximizes the modularity according to the refined modularity matrix can be found using a method similar to the one originally used by Newman. This method, however, assumes a fixed number of clusters c, and writes the modularity as

$$Q = \frac{1}{m} R^T \bar{B} T$$

where R is a $p \times c$ matrix representing the clusters on one side of the graph (so $R_{ik} = 1$ if and only if node i belongs to cluster k) and T is a $q \times c$ matrix representing the clusters on the other side of the graph.

In this case, the method holds the partition on one side of the bipartite network fixed and maximizes the modularity with respect to the other side. We can set R to maximize the modularity when T is held fixed by computing $\bar{T} = \bar{B}T$ (a $p \times c$ matrix) and setting $R_{ik} = 1$, where k is the column index that maximizes $\bar{T}_{ik}$. An analogous procedure can be used to find the value of T that maximizes the modularity when R is held fixed. This gives us the following procedure (2-BRIM):

1. Compute the bipartite modularity matrix $\bar{B}$.

2. Randomly initialize the $p \times c$ and $q \times c$ matrices R and T to have all zeros except for a single 1 in each row. This corresponds to a random assignment of nodes to the c clusters.

3. Compute the partition of one side of the graph which maximizes the modularity Q as written above.

4. Repeat step (3) alternatively for the two sides of the network until no further subdivision leads to increased modularity.

2. Spectral Co-clustering (2-SCC)

Dhillon's spectral co-clustering algorithm is a computationally more efficient adaptation of the basic spectral clustering paradigm (where we try to find partition with more inter-cluster edges than intra-cluster edges) to the bipartite case [4]. The running time in the bipartite-specific algorithm depends on parameter $n_1 n_2$ as opposed to $(n_1 + n_2)^2$ for a general algorithm. The k-multipartitioning algorithm is as follows (directly from [4]).

Let A be the adjacency matrix of our graph, and denote by D_1 and D_2 the diagonal degree matrices associated with the two parts of our bipartite graph.

1. Compute $A_n = D_1^{-1/2} A D_2^{-1/2}$.

2. Compute the $\ell = \lceil \log_2 k \rceil$ singular vectors $u_2, \ldots, u_{\ell+1}$ and $v_2, \ldots, v_{\ell+1}$ (ordered).

3. Compute $Z = [D_1^{-1/2} U, D_2^{-1/2} V]^T$, where $U = [u_2, \ldots, u_{\ell+1}]$ and $V = [v_2, \ldots, v_{\ell+1}]$. Z is an ℓ-dimensional representation of our n nodes.

4. Run k-means algorithm in this ℓ-dimensional space to find our clustering.

C. Paradigm 3

The third paradigm of community detection in bipartite networks involves the construction of an induced unipartite network from the original bipartite network, to which a generic community detection algorithms is then applied. This construction takes advantage of the fact that we only care about communities among the nodes on one side of the bipartite network. For example, suppose that we are interested in detecting communities on the right side of the network. We can then remove all the nodes on the left side of the network, and add an edge between every pair of nodes i and j on the right side which share a neighbor in the original graph (i.e. such there there is some node k on the left side such that ik and jk are edges in the original graph). Furthermore, we assign weights to the edges of the induced graph corresponding to the number of mutual neighbors (i.e. the number of such nodes k). We then apply all the same algorithms that fall under paradigm 1 to this induced unipartite graph.

1. Hierarchical Clustering (3-Hierarchical)

In the hierarchical clustering method of paradigm 3 (3-Hier), the clustering algorithm itself is the same as the one used in 1-Hier. However, because the induced unipartite network has weighted edges, we can apply this algorithm using these weights instead of the weight function used in 1-Hier.

2. Edge-Betweenness Clustering (3-EdgeBetweenness)

The edge-betweenness method of paradigm 3 (3-Betw) remains mostly the same as the one in paradigm 1 (1-Betw). However, when computing the shortest-path lengths between pairs of vertices, we treat an edge of weight w as a sequence of w edges. Equivalently, each edge contributes its weight w to the length of a path that contains it. This captures the idea that nodes with many mutual neighbors are likely to belong to the same group, because it means that edges with high weight are less likely to be included on shortest paths, and thus less likely to be removed from the graph.

3. Basic Modularity-Based methods (3-BasicModularity)

The basic modularity-based method of paradigm 3 (3-BM) is almost identical to the corresponding method in paradigm 1 (1-BM). The only difference is that the adjacency matrix A used to compute the modularity matrix B is no longer a 0-1 matrix, but rather has values corresponding to edge weights in the induced unipartite graph.

4. Basic Spectral Methods (3-BasicSpectral)

In the induced unipartite case, the edges have weights. Spectral graph clustering in fact generally assumes edge weights: it clusters based on the strength of association between nodes, where the weights represent this strength. So in fact, spectral methods may work particularly well here. The Laplacian matrix is defined similarly, except we have edge weights E_{ij} that are not all 1 or 0 (no edge is 0 weight). So diagonal entries $L_{ii} = \sum_k E_{ik}$ and otherwise $L_{ij} = -E_{ij}$.

III. DATA

Each day, Bitly processes and stores data on approximately 300 million click events on shortened links. In particular, they store information about the site visited and a user identifier. This enables us to construct a bipartite graph where the nodes represent users/sites and the edges represent clicks.

We have access to all of Bitly's raw data, made up of all clicks and link shortens over the last five years. For this project we decided to focus on a subset of clicks on links to New York Times (NYT) web pages. We had two main motivations in selecting this target website: (1) NYT URL's include article category information, which enables us to extract this classification and establish a ground truth for our communities. (2) There is a vast amount of data on NYT link clicks. The NYT alone has been shortening links through Bitly since November 2009. Furthermore, many individual users also shorten links to NYT content, and each shorten often receives a large volume of clicks.

To select our data subset we first ran a mapreduce job to get a sense of scale of our potential dataset(s). We found that in one hour (2pm EST) of one day, the New York times receives clicks from 21K distinct users.

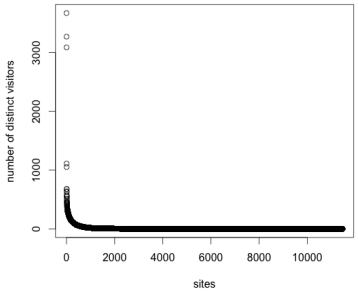
In order to ensure that our graph was not too sparse, we then took the users who had clicked on NYT links in that hour and found all the edges between them and other NYT content over a two month period. This served to limit the number of user nodes to that set of users as well as to control the expansion of site nodes to those connected to this set of users. It also allowed us to increase the likelihood that each user was connected to more than one site node, which increased our incidence of co-clicked sites between users.

We wrote another mapreduce job which took in the list of user identifiers found in our first job and ran over larger timeframes. After iterating we decided to finally run over two months (approximately 18 billion JSON objects) of data and returned [userIdentifier, siteIdentifier] pairs and [siteIdentifier, siteURL] pairs. Then we took the URL's and processed them to extract the topic ground truth (encoded as a directory name e.g. "/politics/"). The most common topics were opinion (1680 nodes), US (1494 nodes), world (1434 nodes), business (1011 nodes), sports (846 nodes), nyregion (738 nodes) and arts (570 nodes). Finally, we took the siteIdentifiers, stored as a typical Bitly hash (i.e. "4k9g6u"), and the user identifiers, and converted them into integer IDs. Using these we constructed our graph.

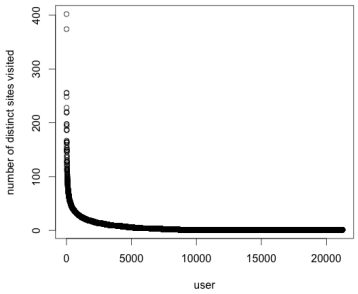
The resulting graph contains **32K** nodes, of which **21K** are users and **11K** are sites, as well as **128K edges**. On average, each user has clicked on links to **6** distinct NYT sites, and each NYT site has received on average **11** average

clicks per site.

We can look at the distribution of number of distinct visitors per site:



And the number of sites visited per user:



and we see that each appears to follow a power-law distribution with very few articles receiving a very large number of distinct visitors, and very few users visiting a very large number of articles as we would expect.

However, after initial work on our algorithms, we made two discoveries. The first was that the communities generated around these topics were highly noisy because of overlap in topic themes. For example, an 'opinion' article could discuss the same content as an article on US news. To handle this we took our initial list of 24 topics

['opinion', 'us', 'world', 'business', 'sports', 'nyregion', 'upshot', 'arts', 'fashion', 'science', 'magazine', 'technology', 'health', 'books', 'movies', 'dining', 'travel', 'theater', 'your-money', 'real-estate', 'education', 'garden', 'automobiles', 'jobs']

and performed three actions. First, we removed any topics which were ambiguous (i.e. whose content could include content from many other topics). For example 'magazine' and 'opinion.' Second, we removed topics which appeared only a very small number of times, such as 'jobs', 'automobiles' and 'garden.' Finally, we consolidated related topics. For example, 'science,' 'health' and 'technology' became 'science.' This process left us with the following five final topic groups:
['arts', 'business', 'currentAffairs', 'science', 'sports']

The second change we made was to use only the largest connected component for our graph. Prior to this switch, many of our algorithms were suffering from the presence of non-connected elements. After the two changes above we had a final graph containing **19.5K** nodes, of which **12.75** are users and **6.75K** are sites, and **78.5K edges**.

IV. EVALUATION

We use the ground-truth communities present in the Bitly dataset to evaluate the performance of the various community detection methods under consideration. Specifically, we compare the ground-truth partition with the partitions induced by these methods according to one of four metrics: best cluster-matching-based accuracy (BCMA), Rand index (RI), Jaccard index (JI), and normalized mutual information (NMI). All four of these metrics are described in [6].

We chose to use four distinct metrics because doing so produces a more nuanced picture about the performance of the methods. If one method outperforms another according to all three metrics, then we can be confident in concluding that the former is more effective than the latter. On the other hand, if the metrics disagree about the relative effectiveness of two methods, then the differences between the metrics can offer insight into the specific kind of community structure detected by the two methods.

A. Best Cluster-Matching Accuracy

In cluster matching, the accuracy is calculated with respect to a specific matching between the algorithm's clusters and the ground-truth clusters. A matching is a one-to-one correspondence between all clusters in the clustering with fewer clusters (coarser) and a subset of the clusters in the finer clustering. For a given matching, the accuracy is just the portion of the vertices whose ground-truth cluster and algorithm-output cluster are matched. The best matching is the one that maximizes this accuracy, and the accuracy of this best matching is our metric. While perhaps in some ways the most intuitive (under the assumption that discovered clusters match with ground-truth clusters, this metric is a natural choice), this metric is flawed. If the algorithm produces a slightly finer or coarser clustering that otherwise corresponds well to the ground-truth, then some clusters will be unmatched and the accuracy will likely be an underestimate.

B. Rand Index

The Rand Index is based on the relationships between pairs of vertices. It treats the ground-truth and algorithm clustering symmetrically (which seems like a nice property). We consider a pair of vertices correct if they either share the same cluster in both clusterings or if they are in different clusters in both clusterings. The fraction of pairs of vertices that are correct is the Rand Index. Let a_{11} be the number of pairs of vertices that are in the same cluster in both clusterings. Let a_{00} be the number of pairs of vertices that are in different clusters in both clusterings. And let a_{01} and a_{10} be the vertices that are in the same cluster in one clustering and in different clusters in the other. So then the Rand Index is

$$\frac{a_{11} + a_{00}}{a_{11} + a_{10} + a_{01} + a_{00}}$$

This metric seems a bit more robust to clustering coarseness. A potential weakness is over-weighting a_{00}: there are automatically many more pairs of vertices in different

clusters than in the same cluster (when we have a good number of clusterings), but we weight these pairs the same (and intuitively, different-ness in itself may not really indicate correctness).

C. Jaccard Index

The Jaccard index is a direct fix to the over-weighting of a_{00} in the Rand Index: we give weight zero to the a_{00} term. We no longer award a clustering if it keeps separate pairs separate (which we expect to happen often regardless of correctness). This results in a more discriminative metric in many cases. The Jaccard index is

$$\frac{a_{11}}{a_{11} + a_{10} + a_{01}}$$

D. Normalized Mutual Information

Another approach with perhaps a stronger theoretical basis is to view the clusterings as multinomial random variables. The probability space is the nodes, the values of the variables are the various clusters. Our metric is then the normalized mutual information (NMI) of the two resulting random variables, a measure of the statistical dependence between the two variables, with 0 the NMI of independent variables and 1 the NMI of identical variables. The formula is

$$\frac{2I(X,Y)}{H(X) + H(Y)}$$

where X and Y are our two random variables, the mutual information

$$I(X,Y) = \sum_x \sum_y P(x,y) \log \frac{P(x,y)}{P(x)P(y)}$$

, and for a random variable Z, the entropy $H(Z) = -\sum_z P(z) \log z$.

V. PREDICTIONS

Before implementing the algorithms, we made some predictions about which ones would perform the best. These predictions can be divided into two categories: those about speed, and those about clustering quality.

Regarding speed, we predicted that algorithms in Paradigm 1 would be slower than those in Paradigm 3. This is not an obvious (or as it turns out, always correct) observation - although the induced unipartite graph used for Paradigm 3 has fewer nodes than the original bipartite graph (6740 vs 19471), it has many more edges (777284 vs 78476). We expected that the much smaller matrix sizes (due to fewer nodes) would lead to a faster algorithm (we weren't exactly sure how the sparsity would affect things, but we thought the matrix size would be a more dominant factor in most algorithms). In paradigm 2, we expected SCC to be only slightly faster than the basic spectral: we figured that eigenvalue computation leverages sparsity enough that we automatically get speedups without the tricks of SCC.

In terms of accuracy, on the other hand, we predicted that most algorithms in Paradigm 3 would be less accurate than those in the other paradigms. This seems intuitively plausible because the induced unipartite graph is a deterministic function of the full bipartite graph: it has strictly less information than the full bipartite graph. We figured the algorithms would automatically leverage the greater information present in the full graph structure. We expected that most algorithms in paradigm 1 would do quite well, and that BRIM in paradigm 2 would do the best, as it leverages the bipartite structure of the graph, which the algorithms in paradigm 1 do not. The induced unipartite graphs have the advantage of more explicitly capturing relevant information in the edge weights; we thought that this benefit would not be significant (but it turns out that paradigm 3 was best, so perhaps it was).

VI. RESULTS/DISCUSSION

Table 1 lists our results in full. In this section, we briefly discuss the results of each algorithm and compare it to its competitors. First, though, we make some general comments about the different metrics and how they should be interpreted. One thing to note is that the metrics do not always agree on the accuracy of an algorithm's clustering. This is most evident in the fact that a ranking of the algorithms according to RI scores would give almost exactly the opposite picture as a ranking according to BCMA or JI scores. In fact, according to RI scores, the random clustering algorithm appears to perform the best. Furthermore, the algorithms with the best BCMA and JI scores (1-Hierarchical, 1-BasicModularity, 3-Hierarchical) have some of the lowest NMI scores. So each of these metrics gives us a different picture of an algorithm's clustering quality.

In order to appreciate the meaning of these differences, we make use of an additional piece of information: the sizes of the clusters returned by our algorithms. These sizes are not uniform across the various algorithms. Some algorithms, like 2-BRIM, 3-BasicModularity, and 3-BasicSpectral, have cluster sizes that are relatively close to the sizes of the ground-truth communities. Other algorithms have more stratified cluster sizes. The most extreme examples are 1-BasicModularity and 1-Hierarchical, which each contain multiple degenerate clusters (with 0 or 1 nodes) and place the vast majority of the nodes into a single cluster. These algorithms also score the best on BCMA and JI.

But should we take these apparently optimistic scores seriously? Intuitively, it would seem that we should not - such clusterings cannot possibly contain much information about the underlying communities because they are so different from our ground truth set. This suggests that we must be cautious in ranking algorithms based on scores alone without considering other factors. Specifically, it seems that when the sizes of the ground-truth clusters differ significantly from those returned by the algorithm, the metrics BCMA, RI, and JI metrics give a less clear picture of clustering accuracy. The NMI metric, on the other hand, seems robust to such differences: the two algorithms with the most "extreme" clusterings also score the worst according to this metric.

These differences support the idea that a wide variety of metrics is needed to get an accurate picture of how our algorithms stack up. They are also surprising, and the rea-

Paradigm 1	**BCMA**	**RI**	**JI**	**NMI**	**Runtime (s)**	**Cluster sizes**
1-Hierarchical	0.4412	0.4015	0.2758	0.0020	3738	(965, 1, 39, 5734, 1)
1-EdgeBetweenness	-Prohibitive-	-Prohibitive-	-Prohibitive-	-Prohibitive-	-Prohibitive-	-Prohibitive-
1-BasicModularity	0.4877	0.3261	0.3044	0.0009	420.29	(0, 0, 0, 158, 6582)
1-BasicSpectral	0.3599	0.5121	0.2234	0.0075	4.16	(4, 5, 822, 993, 4916)
Paradigm 2	**BCMA**	**RI**	**JI**	**NMI**	**Runtime (s)**	**Cluster sizes**
2-BRIM	0.3227	0.5877	0.1855	0.0088	8584.85	(563, 812, 876, 1411, 3078)
2-SCC	0.3542	0.5191	0.2205	0.0079	0.96	(4, 5, 990, 1706, 4035)
Paradigm 3	**BCMA**	**RI**	**JI**	**NMI**	**Runtime (s)**	**Cluster sizes**
3-Hierarchical	0.4949	0.3118	0.3089	0.0040	9185	(29, 1699, 3433, 28, 551)
3-EdgeBetweenness	-Prohibitive-	-Prohibitive-	-Prohibitive-	-Prohibitive-	-Prohibitive-	-Prohibitive-
3-BasicModularity	0.2718	0.5957	0.1546	0.0117	137.94	(493, 1126, 1176, 1552, 2393)
3-BasicSpectral	0.3488	0.4989	0.2126	0.0219	10.48	(64, 506, 675, 1138, 4357)
Random	**BCMA**	**RI**	**JI**	**NMI**	**Runtime (s)**	**Cluster sizes**
Random	0.2122	0.6143	0.1383	0.0008	None	(1312, 1322, 1338, 1345, 1423)

TABLE I. Results for all algorithms

sons for them are not immediately obvious. However, we can explain them by examining their mathematical foundations more closely. Because of the difficulty of our particular clustering problem, our data serve as stress tests for the metrics: the more ad-hoc metrics (BCMA, RI, JI) are usually designed with an intuition of what happens when a clustering is close to perfect, and mostly to distinguish between something like "close to perfect" and "closer to perfect." We would like to distinguish between not-so-great clusterings. Mathematically, two (limiting) clusterings we would like a metric to consider bad are the one-bucket clustering which buckets (nearly) all nodes into a bucket, and a "random" clustering: say the (approximate) expected value of the metric on a random clustering (where cluster assignment is chosen at random).

Let $c_1, \ldots, c_k$ be the sizes of the k ground-truth clusters. For the one-bucket clustering, the Rand Index and Jaccard Index are $\sum \binom{c_i}{2}/\binom{n}{2} \approx \sum c_i^2/n^2$. For k equal clusters this value is about $1/k$, though its larger for more unbalanced clusterings. The BCMA is $\max\{c_i\}/n$, and the NMI is 0. The result for NMI seems to match our intuitions about cluster quality better than that for BCMA: a one-bucket clustering is bad. While at first it seems that rating the one-bucket clustering high for an unbalanced clustering is good, on closer scrutiny BCMA is not a great metric here: consider a 80-20 ground truth clustering. BCMA rates a 100-0 detected clustering the same as a (72,2)-(8,18) clustering. It seems clear that the latter clustering is quite a bit better though (there's some error, but it has both clusters figured out decently well as opposed to just putting all in one bucket: analogous to accuracy vs F-score).

For the random clustering, let $s = \sum \binom{c_i}{2}$ be the number of ground-truth pairs in the same clusters, and $d = \binom{n}{2} - s$ pairs in different clusters. The probability that any pair is in the same cluster is $1/k$, and that any pair is in a different cluster is $(k-1)/k$ (which kind of high). With this, we have that the expected Rand Index, Jaccard Index are

$$\frac{(1/k)s + (1-1/k)d}{s+d}, \frac{(1/k)s}{s+(1/k)d}$$

Generally s is a quite a bit smaller than d, and the Rand Index of a random clustering consequently is pretty high: for an even ground-truth clustering, we have $s/s + d \approx 1/k$, so expected RI $= (1/k)^2 + (1-1/k)^2$, which is pretty high (0.6ish for $k = 5$, approaches 1 for large k); this seems intuitively quite bad. For example the RI of a random clustering is the same/better than a 2-clustering which consists of unions of the ground-truth clusters: for even k, the 2-clustering has RI $(s+(n/2)^2)/(s+d) \approx 1/2 + 1/k$; formula is similar for odd k, and for an even clustering, it turns out these RIs are about equal for $k = 5$, and the random is better for $k \geq 6$. Expected JI about these indices on the other hand is $1/(2k-1)$, which seems okay. One counterintuitive thing to note here is that the one-bucket clustering has much lower Rand Index than random expected, but JI of one-bucket is slightly higher than expected random. Both the one-bucket and random clustering should intuitively be bad: in RI they are considered very different, while in JI they are not the same but quite close. This is a bit of a knock against RI, in the sense that it thinks a random clustering is significantly better than one-bucket.

For $n >> k$, expected BCMA is only slightly larger than $1/k$ (regardless of the ground-truth clustering): with high probability, the (nontrivially sized) ground-truth clusters will be close-to-evenly spread among the random clusters, so we can't do much better than $1/k$. And NMI is zero if the ground-truth clusters are evenly split by detected clusters. The expected random NMI w.r.t a particular ground-truth clustering is a bit complicated (close to even, so close to zero). Experimentally it's about 0.00079 for our ground-truth (10000 trials, with sample std. dev 0.00028). Most of our clusterings have significantly higher NMI (and if we randomly pick clusters in proportion to

ground-truth frequencies mean/std. dev are 0.00086 and 0.00029). A theoretical approximation could be derived by noting that roughly for a ground-truth cluster of size c_i, the random cluster frequencies will be roughly σ from $1/k$-uniform where σ is the Binomial$(1/k, c_i)$ standard deviation.

NMI appears to best pass the sanity checks, and so we think it's the best metric theoretically. Also, NMI seems to capture perhaps the deepest intuition on what makes a good clustering. It's a mathematical answer to the question: how much does a detected clustering tell us about the true clustering, i.e. given the detected clustering, relatively, how much information (bits) do we need in order to specify the ground-truth clustering?

These results give us some perspective as to how we should interpret the different scores assigned to our algorithms by the metrics. We now proceed to summarize the results of each individual algorithm.

A. Paradigm 1

1. 1-Hierarchical

There are two main issues with 1-Hierarchical. The first is that it does not scale well. We see this impact on performance in our runtime. Our algorithm demands that we calculate a distance measure between each pair of nodes. Since our bipartite graph is unweighted, we can construct this distance matrix in $O(V^2)$ time complexity. We then need to sort these measures and proceed according to the methodology of the linking methodology we select. For this project, we chose to use average-link clustering, which gives us a final time complexity of $O(V^2 logV)$.

The second obvious issue when we examine the result of 1-Hierarchical is the uneven distribution of the data across clusters. In particular, we see two large clusters containing over 99% of the nodes. Interestingly, we if increase our number of clusters to 7 from 5 we see these groups being to break-up relatively well. Our cluster of 5734 nodes splits into two clusters, one containing 3964 nodes, the other 1770. This illustrates an inherent issue in hierarchical clustering, which is that it performs local optimization, and we cannot split a cluster once it has been agglomerated. This means we may frequently struggle to reach a global optimal solution.

A third issue with the 1-Hierarchical algorithm is that there is no principled way setting the number of clusters beforehand. Here we may have been better served by allowing the algorithm to generate more clusters and selecting the 5 largest, instead of artificially limiting the result to 5 to force alignment with our ground-truth communities.

2. 1-EdgeBetweenness

The original Girvan and Newman basic implementation of 1-EdgeBetweenness is prohibitively slow for a graph of this size. As discussed in the paradigms section, at each iteration we must calculate the edge betweenness of all edges in order to identify the 'bottlenecks' between communities. Using the Girvan and Newman algorithm this requires, for each edge, to find the contribution to its 'betweenness' from each vertex, then sum over our V vertices giving us a runtime of $O(VE)$ per iteration so a final worst-case running time of $O(E^2V)$. Since our communities are not trivially separable, the number of edges we must remove is non-trivial resulting in a prohibitively high running time. A suggested faster implementations would introduce a stochastic element to run over only a subset of our edges.

A secondary issue, as in the case of our 1-Hierarchical algorithm, is that this algorithm does not readily suggest how many communities our data naturally falls into. While this freedom from restricting the number of communities might be viewed as a benefit of a community finding algorithm, without any stopping criteria the traditional algorithm continues its divisive approach until all edges have been removed. In order to derive meaningful communities for a practical application you must introduce some way of breaking before you are left with purely unconnected nodes.

3. 1-BasicModularity

The 1-BasicModularity algorithm presents an interesting case. It is designed in such a way that it does not search for a predetermined number of clusters, but rather continues an iterative clustering process until no further subdivisions of the network can increase the modularity. When applied to the overall bipartite network, it stopped after only two clusters had been found, and grouped more than 97% of the site nodes in a single cluster. We then tried forcing the algorithm to continue clustering until it produced the number of ground-truth clusters, but this was hardly better since two of the five clusters ended up containing only user nodes.

The clustering produced by the original, self-stopping version of the algorithm (which is listed in the results table) has the best BCMA and JI scores of any algorithm we tested. As we noted earlier, however, this does not necessarily mean that 1-BasicModularity is a good algorithm. Indeed, it scored essentially the same as Random clustering according to the NMI metric (both had a score of 0.0009 when rounded to four decimal places).

Perhaps the best indication of this algorithm's performance is cluster sizes: the picture it gives us about the ground-truth communities is barely any clearer than the picture we could have come up with by grouping all the nodes into one giant cluster. Furthermore, the lackluster runtime of the algorithm means that it cannot make up for this inefficacy through speed. So we conclude that there is little reason to consider 1-BasicModularity for community detection in bipartite graphs.

4. 1-BasicSpectral

The performance of the 1-BasicSpectral algorithm seems to be somewhere in the middle of the road. Its BCMA and JI scores are among the highest of the algorithms that do not produce extreme clustering sizes. We might worry that the cluster sizes of 1-BasicSpectral are still too extremal (there are two degenerate clusters containing 4 and 5 nodes). This is a legitimate concern - however, the NMI score for this algorithm is significantly better than the scores of the other extremal algorithms and the random baseline. So it seems that the algorithm is at least on to something.

The main selling point of this algorithm is speed. It is the second-fastest of all the algorithms tested, beaten only by 2-SCC. This makes sense, as it makes use of eigenvalue/vector calculations on a rather sparse matrix. So we can conclude that 1-BasicSpectral is a reasonably accurate community detection algorithm that runs very quickly.

B. Paradigm 2

1. 2-BRIM

In terms of performance, the 2-BRIM algorithm also seems to be somewhere in the middle. It has the third-best NMI score, but its BCMA and JI scores are on the low end of the spectrum. However, when these scores are compared only to algorithms with comparable NMI scores, it appears to be more effective. Only one other algorithm (3-BasicSpectral) does better across the board. Another upshot of this algorithm is that it doesn't produce any degenerate clusters - all of its clusters contain at least a few hundred nodes.

The main downside of 2-BRIM is obviously its dismal runtime. It is the slowest of all the algorithms that terminated at all, and is nearly four orders of magnitude slower than the fastest algorithm tested (2-SCC). However, it is worth noting that 2-BRIM's speed is due to the fact that it implements a series of local optimization procedures. In this sense, its slowness isn't an inherent property of the algorithm, but rather a property of the implementation. Our implementation conducted 100 such optimizations and returned the clustering with the highest modularity out of all the local maxima it encountered. This number could be reduced in theory, but one would have to be willing to accept a less accurate clustering - in our trial a new best clustering was found even on the 74th iteration. So it seems that 2-BRIM is a flexible algorithm that allows for a trade-off between speed and effectiveness: it has the potential to be either slow and moderately accurate, or fast but less accurate.

2. 2-SCC

The performance of the 2-SCC algorithm is on the higher end of the spectrum, with the second-best BCMA and JI scores of all the non-extremal algorithms, and a moderately good NMI score. These scores closely match those of 1-BasicSpectral. This is no surprise mathematically, as 2-SCC optimizes the same objective function as 1-BasicSpectral, except with a slightly different method using singular values/vectors of a smaller matrix as opposed to eigenvalues/vectors. The numerical variations in these computations likely explain the differences we observe.

The obvious advantage of this algorithm is its speed. It is by far the fastest of any algorithm testing, taking less than a second to finish. In particular, its about 5 times faster than the basic spectral algorithm. This is also to be expected, as one of the main theoretical selling points of the 2-SCC algorithm is its use of the bipartite structure to decompose the matrices and speed up computations. So 2-SCC seems to be a reasonably effective and extremely fast algorithm, which is probably preferable in all cases to 1-BasicSpectral due to their similarity in clustering and difference in speed.

C. Paradigm 3

1. 3-Hierarchical

Our results for our 3-Hierarchical algorithm are similar to those we obtained with 1-Hierarchical. Again we see we have not effectively split our data into evenly distributed clusters, but we have improved on the performance of 1-Hierarchical in this respect. Also, as in the bipartite case we experience issues in the efficiency of the algorithm. Relative to our Paradigm 1 implementation we observe two changes. The first is, clearly, that the number of nodes has decreased, we are now including only the 6740 nodes which represent our websites. However, this is counterbalanced by: (1) We now have a weighted graph and so the distance calculation for each pair of nodes has become more complex. Using Ford-Fulkerson we have a time-complexity of $O(F(|V| + |E|))$. (2) Our number of edges has increased dramatically, Our unipartite graph has 777,284 edges compared to only 78,476 in our original bipartite network.

So despite the fact that we are running our clustering over graph with a smaller number of vertices the other changes to our network resulted in a significantly larger running time without meaningful improvements in results.

2. 3-EdgeBetweenness

As discussed for 1-EdgeBetweenness, the original Girvan and Newman edge-betweenness algorithm is prohibitively slow. They state that they believe the algorithm should be tractable for networks of up to 10,000 vertices. However, although in our unipartite graph we have less than 7,000 vertices, in the process of conversion we have introduced a much larger number of edges to the graph as discussed above. We experimented with modifications to the algorithm but were not able to find one which improved efficiency without deviating too far from what could legitimately be called an implementation of edge-betweenness.

3. 3-BasicModularity

The effectiveness of the 3-BasicModularity algorithm is somewhat difficult to determine. On the one hand, it has the lowest BCMA and JI scores of all the algorithms tested. On the other hand, it has the second-highest NMI score. What are we to make of this? The high NMI score is a good indicator that the clusters it produces are non-degenerate, and indeed we see that the sizes of these clusters are well-distributed. However, because it scores worse on the other metrics than algorithms which give similarly well-distributed clusters (most notably 2-BRIM and 3-BasicSpectral), it doesn't really stand out in terms of performance. Its runtime is also fairly middle-of-the-road. So we can conclude that 3-BasicModularity, while not ineffective, is not the best choice of the algorithms tested in terms of either performance or speed.

4. 3-BasicSpectral

The 3-BasicSpectral algorithm is high-performer in terms of clustering quality. For one thing, it has the best

NMI score of all. Furthermore, its BCMA and JI are the third-highest among non-extremal algorithms, and are only slightly below those of the two higher-performing algorithms. Its cluster sizes are generally well-distributed, though it does find a fairly small cluster with only 64 nodes. Finally, its speed is also top-notch, clocking in at slightly over 10 seconds. This makes it the third-fastest algorithm tested, and around one order of magnitude away from the fastest algorithm. Based on these results, we can conclude that the 3-BasicSpectral algorithm is a great all-around option - it isn't the best in any one category (apart from NMI), but it is near the top in all categories.

VII. CONCLUSION

In this project, we extracted a novel dataset from the Bitly archive of link-shortening data. This dataset takes the form of a large bipartite network, with nodes representing users/sites and edges representing click actions, as well as a set of topics for each site extracted from link urls that form ground-truth communities. We then implemented a variety of algorithms for community detection in bipartite networks, drawn from three broad paradigms: first, general community detection algorithms applied to the full network; second, bipartite-specific algorithms applied to the full network; and third, general algorithms applied to an induced unipartite network. We then applied these algorithms to our dataset in order to test their efficiency and effectiveness.

We found that there was no clear winner, either in terms of algorithms or in terms of paradigms. It did seem clear that the algorithms in Paradigm 1 were strictly worse than those in the other paradigms. The best algorithm from this paradigm, 1-BasicSpectral, seemed to be essentially a slower version of 2-SCC.

Among the other paradigms, the best choice of algorithm seems to depend on which factors are the most important, though three algorithms in particular stand out. 2-SCC provides an extremely fast algorithm which is moderately effective at detecting the ground-truth communities. 3-BasicSpectral runs slightly more slowly, but may give more accurate clusterings. Finally, 2-BRIM allows the flexibility to make tradeoffs between speed and accuracy and also avoids the pitfall of producing degenerate clusters, but doesn't score quite as highly as the other two in terms of overall clustering accuracy.

However, our work also leaves unanswered questions and room for improvement. For one thing, although extracting the raw ground-truth data directly from the site URLs certainly gives us the topic as defined by the New York Times, these explicit labelings may not be the most accurate representation of the content's underlying subject. We saw some examples of this in our initial results. We handled this with a rough grouping of topics, but our process for deciding which topics should be combined to create less noisy groups was not particularly rigorous.

A second way to improve upon our dataset would be to form edges between nodes based on link-shortens instead of link-clicks. We hypothesize that the content users shorten may be more representative of their interests and therefore form more cohesive communities than content they click on. Given more time, we would like to re-visit these question to obtain a more pure data set.

In the process of implementing our algorithms, we also observed that several of them (most notably the ones based on edge-betweenness) could be made faster by introducing a stochastic element rather than calculating measures for all data points. Future work could be done to investigate whether such improvements would make these algorithms competitive with the ones that our results found to be the best.

In spite of these shortcomings, we think that our project represents an important contribution to the field of community detection in bipartite networks, as it provides both a novel dataset and an extensive set of results on this dataset. We hope that these results shed some light on which community detection algorithms - and which paradigms - are most appropriate to use on large, bipartite networks.

[1] Girvan, Michelle, and Mark EJ Newman. "Community structure in social and biological networks." Proceedings of the National Academy of Sciences 99.12 (2002): 7821-7826.

[2] M. E. J. Newman and M. Girvan. "Finding and evaluating community structure in networks." Physical Review, E 69(026113), 2004.

[3] Newman, Mark EJ. "Modularity and community structure in networks." Proceedings of the National Academy of Sciences 103.23 (2006): 8577-8582.

[4] Dhillon, Inderjit S. "Co-clustering documents and words using bipartite spectral graph partitioning." Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2001.

[5] Barber, Michael J. "Modularity and community detection in bipartite networks." Physical Review E 76.6 (2007): 066102.

[6] Fortunato, Santo. "Community detection in graphs." Physics Reports 486.3 (2010): 75-174.