Efficient Community Detection in Large Networks using Content and Links

Yiye Ruan, David Fuhry, Srinivasan Parthasarathy
Department of Computer Science and Engineering
The Ohio State University
{ruan,fuhry,srini}@cse.ohio-state.edu

ABSTRACT

In this paper we discuss a very simple approach of combining content and link information in graph structures for the purpose of community discovery, a fundamental task in network analysis. Our approach hinges on the basic intuition that many networks contain noise in the link structure and that content information can help strengthen the community signal. This enables ones to eliminate the impact of noise (false positives and false negatives), which is particularly prevalent in online social networks and Web-scale information networks.

Specifically we introduce a measure of signal strength between two nodes in the network by fusing their link strength with content similarity. Link strength is estimated based on whether the link is likely (with high probability) to reside within a community. Content similarity is estimated through cosine similarity or Jaccard coefficient. We discuss a simple mechanism for fusing content and link similarity. We then present a biased edge sampling procedure which retains edges that are locally relevant for each graph node. The resulting backbone graph can be clustered using standard community discovery algorithms such as Metis and Markov clustering.

Through extensive experiments on multiple real-world datasets (Flickr, Wikipedia and CiteSeer) with varying sizes and characteristics, we demonstrate the effectiveness and efficiency of our methods over state-of-the-art learning and mining approaches several of which also attempt to combine link and content analysis for the purposes of community discovery. Specifically we always find a qualitative benefit when combining content with link analysis. Additionally our biased graph sampling approach realizes a quantitative benefit in that it is typically several orders of magnitude faster than competing approaches.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*

Keywords

Web mining, graph clustering, content analysis

1. INTRODUCTION

An increasing number of applications on the World Wide Web rely on combining link and content analysis (in different ways) for subsequent analysis and inference. For example, search engines, like Google, Bing and Yahoo! typically use content and link information to index, retrieve and rank web pages. Social networking sites like Twitter, Flickr and Facebook, as well as the aforementioned search engines, are increasingly relying on fusing content (pictures, tags, text) and link information (friends, followers, and users) for deriving actionable knowledge (e.g. marketing and advertising).

In this article we limit our discussion to a fundamental inference problem — that of combining link and content information for the purposes of inferring clusters or communities of interest. The challenges are manifold. The topological characteristics of such problems (graphs induced from the natural link structure) makes identifying community structure difficult. Further complicating the issue is the presence of noise (incorrect links (false positives) and missing links (false negatives). Determining how to fuse this link structure with content information efficiently and effectively is unclear. Finally, underpinning these challenges, is the issue of scalability as many of these graphs are extremely large running into millions of nodes and billions of edges, if not larger.

Given the fundamental nature of this problem, a number of solutions have emerged in the literature. Broadly these can be classified as: i) those that ignore content information (a large majority) and focus on addressing the topological and scalability challenges, and ii) those that account for both content and topological information. From a qualitative standpoint the latter presumes to improve on the former (since the null hypothesis is that content should help improve the quality of the inferred communities) but often at a prohibitive cost to scalability.

In this article we present CODICIL[1], a family of highly efficient graph simplification algorithms leveraging both content and graph topology to identify and retain important edges in a network. Our approach relies on fusing content and topological (link) information in a natural manner. The output of CODICIL is a transformed variant of the original graph (with content information), which can then be clustered by any fast content-insensitive graph clustering algorithm such as METIS or Markov clustering. Through extensive experiments on real-world datasets drawn from Flickr, Wikipedia, and CiteSeer, and across several graph clustering algorithms, we demonstrate the effectiveness and efficiency of our methods. We find that CODICIL runs several orders of magnitude faster than those state-of-the-art approaches and often identifies communities of comparable or superior quality on these datasets.

This paper is arranged as follows. In Section 2 we discuss existent research efforts pertaining to our work. The algorithm of CODICIL, along with implementation details, is presented in Section 3. We report quantitative experiment results in Section 4, and

---

[1]COmmunity Discovery Inferred from Content Information and Link-structure

demonstrate the qualitative benefits brought by CODICIL via case studies in Section 5. We finally conclude the paper in Section 6.

2. RELATED WORK

Community Discovery using Topology (and Content): Graph clustering/partitioning for community discovery has been studied for more than five decades, and a vast number of algorithms (exemplars include Metis [15], Graclus [6] and Markov clustering [27]) have been proposed and widely used in fields including social network analytics, document clustering, bioinformatics and others. Most of those methods, however, discard content information associated with graph elements. Due to space limitations, we suppress detailed discussions and refer interested readers to recent surveys (e.g. [9]) for a more comprehensive picture. Leskovec et al. compared a multitude of community discovery algorithms based on conductance score, and discovered the trade-off between clustering objective and community compactness [16].

Various approaches have been taken to utilize content information for community discovery. One of them is generative probabilistic modeling which considers both contents and links as being dependent on one or more latent variables, and then estimates the conditional distributions to find community assignments. PLSA-PHITS [5], Community-User-Topic model [29] and Link-PLSA-LDA [20] are three representatives in this category. They mainly focus on studies of citation and email communication networks. Link-PLSA-LDA, for instance, was motivated for finding latent topics in text and citations and assumes different generative processes on citing documents, cited documents as well as citations themselves. Text generation is following the LDA approach, and link creation from a citing document to a cited document is controlled by another topic-specific multinomial distribution.

Yang et al. [28] introduced an alternative discriminative probabilistic model, PCL-DC, to incorporate content information in the conditional link model and estimate the community membership directly. In this model, link probability between two nodes is decided by nodes' *popularity* as well as community membership, which is in turn decided by content terms. A two-stage EM algorithm is proposed to optimize community membership probabilities and content weights alternately. Upon convergence, each graph node is assigned to the community with maximum membership probability.

Researchers have also explored ways to augment the underlying network to take into account the content information. The SA-Cluster-Inc algorithm proposed by Zhou et al. [30], for example, inserts virtual *attribute nodes* and *attribute edges* into the graph and computes all-pair random walk distances on the new *attribute-augmented graph*. K-means clustering is then used on original graph nodes to assign them to different groups. Weights associated with attributes are updated after each k-means iteration according to their clustering tendencies. The algorithm iterates until convergence.

Ester et al. [8] proposed an heuristic algorithm to solve the *Connected k-Center* problem where both connectedness and radius constraints need to be satisfied. The complexity of this method is dependent on the longest distance between any pair of nodes in the feature space, making it susceptible to outliers. Biologists have studied methods [13, 26] to find functional modules using network topology and gene expression data. Those methods, however, bear domain-specific assumptions on data and are therefore not directly applicable in general.

Recently Günnemann et al. [12] introduced a subspace clustering algorithm on graphs with feature vectors, which shares some similarity with our topic. Although their method could run on the full feature space, the search space of their algorithm is confined by the intersection, instead of union, of the epsilon-neighborhood and the density-based combined cluster. Furthermore, the construction of both neighborhoods are sensitive to their multiple parameters.

While decent performance can be achieved on small and medium graphs using those methods, it often comes at the cost of model complexity and lack of scalability. Some of them take time proportional to the number of values in each attribute. Others take time and space proportional to the number of clusters to find, which is often unacceptable. Our method, in contrast, is more lightweight and scalable.

Clustering/Learning Multiple Graphs: Content-aware clustering is also related to multiple-view clustering, as content information and link structure can be treated as two views of the data. Strehl and Ghose [23] discussed three consensus functions (cluster-wise similarity partitioning, hyper-graph partitioning and meta-clustering) to implement cluster ensembles, in which the availability of each individual view's clustering is assumed. Tang et al. [24] proposed a linked matrix factorization method, where each graph's adjacency matrix is decomposed into a "characteristic" matrix and a common factor matrix shared among all graphs. The purpose of factorization is to represent each vertex by a lower-dimensional vector and then cluster the vertices using corresponding feature vectors. Their method, while applicable to small-scale problems, is not designed for web-scale networks.

Graph Sampling for Fast Clustering: Graph sampling (also known as "sparsification" or "filtering") has attracted more and more focus in recent years due to the explosive growth of network data. If a graph's structure can be preserved using fewer nodes and/or edges, community discovery algorithms can obtain similar results using less time and memory storage. Maiya and Berger-Wolf [17] introduced an algorithm which greedily identifies the node that leads to the greatest *expansion* in each iteration until the user-specified node count is reached. By doing so, an expander-like node-induced subgraph is constructed. After clustering the subgraph, the unsampled nodes can be labeled by using collective inference or other transductive learning methods. This extra post-processing step, however, operates on the original graph as a whole and easily becomes the scalability bottleneck on larger networks.

Satuluri et al. [22] proposed an edge sampling method to preferentially retain edges that connect two similar nodes. The localized strategy ensures that edges in the relatively sparse areas will not be over-pruned. Their method, however, does not consider content information either.

Edge sampling has also been applied to other graph tasks. Karger [14] studied the impact of random edge sampling on original graph's cuts, and proposed randomized algorithms to find graph's minimum cut and maximum flow. Aggarwal et al. [1] proposed using edging sampling to maintain structural properties and detect outliers in graph streams. The goals of those work are not to preserve community structure in graphs, though.

3. METHODOLOGY

We begin by defining the notations used in the rest of our paper. Let $\mathcal{G}_t = (\mathcal{V}, \mathcal{E}_t, \mathcal{T})$ be an undirected graph with n vertices $\mathcal{V} = v_1, \ldots, v_n$, edges $\mathcal{E}_t$, and a collection of n corresponding term vectors $\mathcal{T} = \boldsymbol{t}_1, \ldots, \boldsymbol{t}_n$. We use the terms "graph" and "network" interchangeably as well as the terms "vertex" and "node". Elements in each term vector $\boldsymbol{t}_i$ are basic content units which can be single words, tags or n-grams, etc., depending on the context of underlying network. For each graph node $v_i \in \mathcal{V}$, let its term vector be $\boldsymbol{t}_i$.

Our goal is to generate a simplified, edge-sampled graph $\mathcal{G}_{sample} =$

$(\mathcal{V}, \mathcal{E}_{sample})$ and then use $\mathcal{G}_{sample}$ to find communities with coherent content and link structure. $\mathcal{G}_{sample}$ should possess the following properties:

- $\mathcal{G}_{sample}$ has the same vertex set as $\mathcal{G}_t$. That is, no node in the network is added or removed during the simplification process.
- $|\mathcal{E}_{sample}| \ll |\mathcal{E}_t|$, as this enables both better runtime performance and lower memory usage in the subsequent clustering stage.
- Informally put, the resultant edge set $\mathcal{E}_{sample}$ would connect node pairs which are both structure-wise and content-wise similar. As a result, it is possible for our method to add edges which were absent from $\mathcal{E}_t$ since the content similarity was overlooked.

3.1 Key Intuitions

The main steps of the CODICIL algorithm are:

1. Create content edges.
2. Sample the union of content edges and topological edges with bias, retaining only edges that are relevant in local neighborhoods.
3. Partition the simplified graph into clusters.

The constructed content graph and simplified graph have the same vertices as the input graph (vertices are never added or removed), so the essential operations of the algorithm are constructing, combining edges and then sampling with bias. Figure 1 illustrates the work flow of CODICIL.

From the term vectors $\mathcal{T}$, content edges $\mathcal{E}_c$ are constructed. Those content edges and the input topological edges $\mathcal{E}_t$ are combined as $\mathcal{E}_u$ which is then sampled with bias to form a smaller edge set $\mathcal{E}_{sample}$ where the most relevant edges are preserved. The graph composed of these sampled edges is passed to the graph clustering algorithm which partitions the vertices into a given number of clusters.

3.2 Basic Framework

The pseudo-code of CODICIL is given in Algorithm 1. CODICIL takes as input 1) $\mathcal{G}_t$, the original graph consisting of vertices V, edges $\mathcal{E}_t$ and term vectors $\mathcal{T}$ where $\boldsymbol{t}_i$ is the content term vector for vertex v_i, $1 \leq i \leq |\mathcal{V}| = |\mathcal{T}|$, 2) k, the number of nearest content neighbors to find for each vertex, 3) $normalize(\boldsymbol{x})$, a function that normalizes a vector $\boldsymbol{x}$, 4) α, an optional parameter that specifies the weights of topology and content similarities, 5) l, the number of output clusters desired, 6) $clusteralgo(\mathcal{G}, l)$, an algorithm that partitions a graph $\mathcal{G}$ into l clusters and 7) $similarity(\boldsymbol{x}, \boldsymbol{y})$ to compute similarity between $\boldsymbol{x}$ and $\boldsymbol{y}$. Note that any content-insensitive graph clustering algorithm can be plugged in the CODICIL framework, providing great flexibility for applications.

3.2.1 Creating Content Edges

Lines 2 through 7 detail how content edges are created. For each vertex v_i, its k most content-similar neighbors are computed[2]. For each of v_i's top-k neighbors v_j, an edge (v_i, v_j) is added to content edges $\mathcal{E}_c$. In our experiments we implemented the $TopK$ sub-routine by calculating the cosine similarity of $\boldsymbol{t}_i$'s TF-IDF vector and each other term vector's TF-IDF vector. For a content unit

---

Algorithm 1 CODICIL

Input: $\mathcal{G}_t = (\mathcal{V}, \mathcal{E}_t, \mathcal{T})$, k, $normalize(\cdot)$, $\alpha \in [0, 1]$, l, $clusteralgo(\cdot, \cdot)$, $similarity(\cdot, \cdot)$
Returns: $\mathcal{C}$ (a disjoint clustering of $\mathcal{V}$)
1: \\Create content edges $\mathcal{E}_c$
2: $\mathcal{E}_c \leftarrow \emptyset$
3: **for** $i = 1$ to $|\mathcal{V}|$ **do**
4:    **foreach** $v_j \in TopK(v_i, k, \mathcal{T})$ **do**
5:       $\mathcal{E}_c \leftarrow \mathcal{E}_c \cup (v_i, v_j)$
6:    **end for**
7: **end for**
8: \\Combine $\mathcal{E}_t$ and $\mathcal{E}_c$. Retain edges with a bias towards locally relevant ones
9: $\mathcal{E}_u \leftarrow \mathcal{E}_t \cup \mathcal{E}_c$
10: $\mathcal{E}_{sample} \leftarrow \emptyset$
11: **for** $i = 1$ to $|\mathcal{V}|$ **do**
12:    \\Γ_i contains v_i's neighbors in the edge union
13:    $\Gamma_i \leftarrow ngbr(v_i, \mathcal{E}_u)$
14:    **for** $j = 1$ to $|\Gamma_i|$ **do** $\boldsymbol{sim^t}_{ij} \leftarrow similarity(ngbr(v_i, \mathcal{E}_t), ngbr(\gamma_j, \mathcal{E}_t))$
15:    $\boldsymbol{simnorm^t}_i \leftarrow normalize(\boldsymbol{sim^t}_i)$
16:    **for** $j = 1$ to $|\Gamma_i|$ **do** $\boldsymbol{sim^c}_{ij} \leftarrow similarity(\boldsymbol{t}_i, \boldsymbol{t}_{\gamma_j})$
17:    $\boldsymbol{simnorm^c}_i \leftarrow normalize(\boldsymbol{sim^c}_i)$
18:    **for** $j = 1$ to $|\Gamma_i|$ **do** $\boldsymbol{sim}_{ij} \leftarrow \alpha \cdot \boldsymbol{simnorm^t}_{ij} + (1 - \alpha) \cdot \boldsymbol{simnorm^c}_{ij}$
19:    \\Sort similarity values in descending order. Store the corresponding node IDs in idx_i
20:    $[val_i, idx_i] \leftarrow descsort(\boldsymbol{sim}_i)$
21:    **for** $j = 1$ to $\left\lceil \sqrt{|\Gamma_i|} \right\rceil$ **do**
22:       $\mathcal{E}_{sample} \leftarrow \mathcal{E}_{sample} \cup (v_i, v_{idx_{ij}})$
23:    **end for**
24: **end for**
25: $\mathcal{G}_{sample} \leftarrow (\mathcal{V}, \mathcal{E}_{sample})$
26: $\mathcal{C} \leftarrow clusteralgo(\mathcal{G}_{sample}, l)$ \\Partition into l clusters
27: **return** $\mathcal{C}$

---

c, its TF-IDF value in a term vector $\boldsymbol{t}_i$ is computed as

$$tf\text{-}idf(c, \boldsymbol{t}_i) = \sqrt{tf(c, \boldsymbol{t}_i)} \cdot \log\left(1 + \frac{|\mathcal{T}|}{\sum_{j=1}^{|\mathcal{T}|} tf(c, \boldsymbol{t}_j)}\right) . \quad (1)$$

The cosine similarity of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is

$$cosine(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\|_2 \cdot \|\boldsymbol{y}\|_2} . \quad (2)$$

The k vertices corresponding to the k highest TF-IDF vector cosine similarity values with v_i are selected as the top-k neighbors of v_i.

3.2.2 Local Ranking of Edges and Graph Simplification

Line 9 takes the union of the newly-created content edge set $\mathcal{E}_c$ and the original topological edge set $\mathcal{E}_t$. In lines 10 through 24, a sampled edge set $\mathcal{E}_{sample}$ is constructed by retaining the most relevant edges from the edge union $\mathcal{E}_u$. For each vertex v_i, the edges to retain are selected from its local neighborhood in $\mathcal{E}_u$ (line 13). We compute the topological similarity (line 14) between node v_i and its neighbor γ_j as the relative overlap of their respective topological neighbor sets, $I = ngbr(v_i, \mathcal{E}_t)$ and $J = ngbr(\gamma_j, \mathcal{E}_t)$, using $similarity$ (either cosine similarity as in Equation 2 or Jaccard
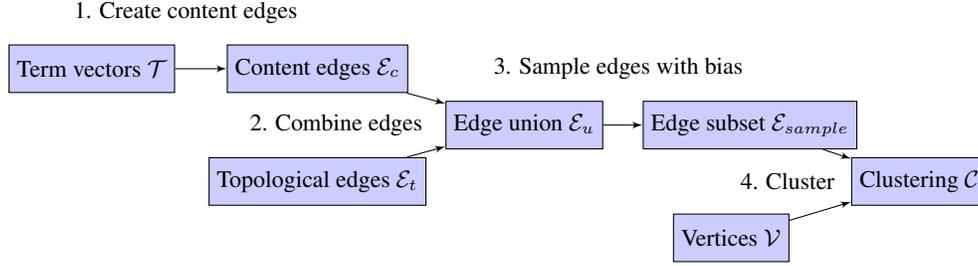
---

[2]Besides top-k criteria, we also investigated using all-pairs similarity above a given global threshold, but this tended to produce highly imbalanced degree distributions.

1. Create content edges

Term vectors $\mathcal{T}$ → Content edges $\mathcal{E}_c$

3. Sample edges with bias

2. Combine edges → Edge union $\mathcal{E}_u$ → Edge subset $\mathcal{E}_{sample}$

Topological edges $\mathcal{E}_t$

4. Cluster → Clustering $\mathcal{C}$

Vertices $\mathcal{V}$

Figure 1: Work flow of CODICIL

coefficient as defined below):

$$jaccard(I, J) = \frac{|I \cap J|}{|I \cup J|} \quad . \tag{3}$$

After the computation of the topological similarity vector $\boldsymbol{sim^t}_i$ finishes, it is normalized by $normalize$ (line 15). In our experiments we implemented $normalize$ with either $zero\text{-}one$, which simply rescales the vector to $[0, 1]$:

$$zero\text{-}one(\vec{x}) = (x_i - min(\vec{x}))/(max(\vec{x}) - min(\vec{x})) \tag{4}$$

or $z\text{-}norm^3$, which centers and normalizes values to zero mean and unit variance:

$$z\text{-}norm(\vec{x}) = \frac{x_i - \hat{\mu}}{\hat{\sigma}}, \hat{\mu} = \frac{\sum_{i=1}^{|\vec{x}|} x_i}{|\vec{x}|}, \hat{\sigma}^2 = \frac{1}{|\vec{x}| - 1} \sum_{i=1}^{|\vec{x}|} (x_i - \hat{\mu})^2 \quad . \tag{5}$$

Likewise, we compute v_i's content similarity to its neighbor γ_j by applying $similarity$ on term vectors $\boldsymbol{t}_i$ and $\boldsymbol{t}_{\gamma_j}$ and normalize those similarities (lines 16 and 17). The topological and content similarities of each edge are then aggregated with the weight specified by α (line 18).

In lines 20 through 23, the edges with highest similarity values are retained. As stated in our desiderata, we want $|\mathcal{E}_{sample}| \ll |\mathcal{E}_t|$ and therefore need to retain fewer than $|\Gamma_i|$ edges. Inspired by [22], we choose to keep $\lceil \sqrt{|\Gamma_i|} \rceil$ edges. This form has the following properties: 1) every vertex v_i will be incident to at least one edge, therefore the sparsification process does not generate new singleton, 2) concavity and monotonicity ensure that larger-degree vertices will retain no fewer edges than smaller-degree vertices, and 3) sublinearity ensures that smaller-degree vertices will have a larger fraction of their edges retained than larger-degree vertices.

3.2.3 Partitioning the Sampled Graph

Finally in lines 25 through 27 the sampled graph $\mathcal{G}_{sample}$ is formed with the retained edges, and the graph clustering algorithm $clusteralgo$ partitions $\mathcal{G}_{sample}$ into l clusters.

3.2.4 Extension to Support Complex Graphs

The proposed CODICIL framework can also be easily extended to support community detection from other types of graph. If an input graph has weighted edges, we can modify the formula in line 18 so that sim_{ij} becomes the product of combined similarity and original edge weight. Support of attribute graph is also straightforward, as attribute assignment of a node can be represented by an indicator vector, which is in the same form of a text vector.

---

[3]Montague and Aslam [19] pointed out that $z\text{-}norm$ has the advantage of being both shift and scale invariant as well as outlier insensitive. They experimentally found it best among six simple combination schemes discussed in [10].

3.3 Key Speedup Optimizations

3.3.1 $\boldsymbol{TopK}$ Implementation

When computing cosine similarities across term vectors $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_{|\mathcal{T}|}$, one can truncate the TF-IDF vectors by only keeping m elements with the highest TF-IDF values and set other elements to 0. When m is set to a small value, TF-IDF vectors are sparser and therefore the similarity calculation becomes more efficient with little loss in accuracy.

We may also be interested in constraining content edges to be within a topological neighborhood of each node v_i, such that the search space of $TopK$ algorithm can be greatly reduced. Two straightforward choices are 1) "1-hop" graph in which the content edges from v_i are restricted to be in v_i's direct topological neighborhood, and 2) "2-hop" graph in which content edges can connect v_i and its neighbors' neighbors.

Many contemporary text search systems make use of inverted indices to speed up the operation of finding the k term vectors (documents) with the largest values of Equation 2 given a query vector $\boldsymbol{t}_i$. We used the implementation from Apache Lucene for the largest dataset.

3.3.2 Fast Jaccard Similarity Estimation

To avoid expensive computation of the exact Jaccard similarity, we estimate it by using minwise hashing [3]. An unbiased estimator of sets A and B's Jaccard similarity can be obtained by

$$\hat{jaccard}(A, B) = \frac{1}{h} \sum_{i=1}^{h} I(\min(\pi_i(A)) = \min(\pi_i(B))) \quad , \tag{6}$$

where $\pi_1, \pi_2, \cdots, \pi_h$ are h permutations drawn randomly from a family of minwise independent permutations defined on the universe A and B belong to, and I is the identity function. After hashing each element once using each permutation, the cost for similarity estimation is only $O(h)$ where h is usually chosen to be less than $|A|$ and $|B|$.

3.3.3 Fast Cosine Similarity Estimation

Similar to Jaccard coefficient, we can apply random projection method for fast estimate of cosine similarity [4]. In this method, each hash signature for a d-dimensional vector $\boldsymbol{x}$ is $h(\boldsymbol{x}) = \text{sgn}(\boldsymbol{x}, \boldsymbol{r})$, where $\boldsymbol{r} \in \{0, 1\}^d$ is drawn randomly. For two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, the following holds:

$$Pr[h(\boldsymbol{x}) = h(\boldsymbol{y})] = 1 - \frac{\arccos(cosine(\boldsymbol{x}, \boldsymbol{y}))}{\pi} \quad . \tag{7}$$

3.4 Performance Analysis

Lines 3–7 of CODICIL are a preprocessing step which compute for each vertex its top-k most similar vertices. Results of this one-time computation can be reused for any $k' \leq k$. Its complexity

depends on the implementation of the $TopK$ operation. On our largest dataset Wikipedia this step completed within a few hours.

We now consider the loop in lines 11–24 where CODICIL loops through each vertex. For lines 14 and 16 we use the Jaccard estimator from Section 3.3.2 for which runs in $O(h)$ with a constant number of hashes h. The normalizations in lines 15 and 17 are $O(|\Gamma_i|)$ and the inner loop in lines 21–23 is $O(\sqrt{|\Gamma_i|})$. Sorting edges by weight in line 20 is $O(|\Gamma_i| \log |\Gamma_i|)$. The size of Γ_i, the union of topology and content neighbors, is at most n but on average much smaller in real world graphs. Thus the loop in lines 11–24 runs in $O(n^2 \log n)$.

The overall runtime of CODICIL is the edge preprocessing time, plus $O(n^2 \log n)$ for the loop, plus the algorithm-dependent time taken by $clusteralgo$.

4. EXPERIMENTS

We are interested in empirically answering the following questions:

- **Do the proposed content-aware clustering methods lead to better clustering than using graph topology only?**
- **How do our methods compare to existing content-aware clustering methods?**
- **How scalable are our methods when the data size grows?**

4.1 Datasets

Three publicly-available datasets with varying scale and characteristic are used. Their domains cover document network as well as social network. Each dataset is described below, and Table 1 follows, listing basic statistics of them.

4.1.1 CiteSeer

A citation network of computer science publications[4], each of which labeled as one of six sub-fields. In our graph, nodes stand for publications and undirected edges indicate citation relationships. The content information is stemmed words from research papers, represented as one binary vector for each document. Observe that the density of this network (average degree 2.74) is significantly lower than normally expected for a citation network.

4.1.2 Wikipedia

The static dump of English Wikipedia pages (October 2011). Only regular pages belonging to at least one category are included, each of which becomes one node. Page links are extracted. Cleaned bi-grams from title and text are used to represent each document's content. We use categories that a page belongs to as the page's class labels. Note that a page can be contained in more than one category, thus ground truth categories are overlapping.

4.1.3 Flickr

From a dataset of tagged photos[5] we removed infrequent tags and users associated with only few tags. Each graph node stands for a user, and an edge exists if one user is in another's contact list. Tags that users added to uploaded photos are used as content information. Flickr user groups are collected as ground truth. Similar to Wikipedia categories, Flickr user groups are also overlapping.

4.2 Baseline Methods

In terms of strawman methods, we compare the CODICIL methods with three existing content-aware graph clustering algorithms, SA-Cluster-Inc [30], PCL-DC [28] and Link-PLSA-LDA (L-P-LDA) [20]. Their methodologies have been briefly introduced in Section 2. When applying SA-Cluster-Inc, we treat each term in $\mathcal{T}$ as a binary-valued attribute, i.e. for each graph node i every attribute value indicates whether the corresponding term is present in t_i or not. For L-P-LDA, since it does not assume a distinct distribution over topics for each cited document individually, only citing documents' topic distributions are estimated. As a result, there are 2313 citing documents in CiteSeer dataset and we report the F-score on those documents using their corresponding ground-truth assignments.

Previously SA-Cluster-Inc has been shown to outperform k-SNAP [25] and PCL-DC to outperform methods including PLSA-PHITS [5], LDA-Link-Word [7] and Link-Content-Factorization [31]. Therefore we do not compare with those algorithms.

Two content-insensitive clustering algorithms are included in the experiments as well. The first method, "Original Topo", clusters the original network directly. The second method samples edges solely based on structural similarity and then clusters the sampled graph [22], and we refer to it as "Sampled Topo" hereafter.

Finally, we also adapt LDA and K-means[6] algorithm to cluster graph nodes using content information only. When applying LDA, we treat each term vector t_i as a document, and one product of LDA's estimation procedure is the distribution over latent topics, θ_{t_i}, for each t_i (more details can be found at the original paper by Blei et al. [2]). Therefore, we treat each latent topic as a cluster and assign each graph node to the cluster that corresponds to the topic of largest probability. We use GibbsLDA++[7], a C++ implementation of LDA using Gibbs sampling [11] which is faster than the variational method proposed originally. Results of this method are denoted as "LDA".

4.3 Experiment Setup

4.3.1 Parameter Selection

There are several tunable parameters in the CODICIL framework, first of which is k, the number of content neighbors in the $TopK$ sub-routine. We propose the following heuristic to decide a proper value for k: the value of k should let $|\mathcal{E}_c| \approx |\mathcal{E}_t|$. As a result, k is set to 50 for both Wikipedia ($|\mathcal{E}_c| = 150,955,014$) and Flickr ($|\mathcal{E}_c| = 722,928$). For CiteSeer, we experiment with two relatively higher k values (50, $|\mathcal{E}_c| = 103,080$ and 70, $|\mathcal{E}_c| = 143,575$) in order to compensate the extreme sparsity in the original network. Though simplistic, this heuristic leads to decent clustering quality, as shown in Section 4.5, and avoids extra effort for tuning.

Another parameter of interest is α, which determines the weights for structural and content similarities. We set α to 0.5 unless otherwise specified, as in Section 4.7. The number of hashes (h) used for minwise hashing (Jaccard coefficient) is 30, and 512 for random projection (cosine similarity). Experiments with both choices of $similarity$ function are performed. As for m, the number of non-zero elements in term vectors, we let $m = 10$ for Wikipedia and Flickr. This optional step is omitted for CiteSeer since the speedup is insignificant.

4.3.2 Clustering Algorithm

We combine the CODICIL framework with two different clus-

	$|\mathcal{V}|$	$|\mathcal{E}_t|$	# CC	$|CC_{max}|$	# Uniq. Content Unit	Avg $|t_i|$	# Class
Wikipedia	3,580,013	162,085,383	10	3,579,995	1,459,335	202	595,355
Flickr	16,710	716,063	4	16,704	1,156	44	184,334
CiteSeer	3,312	4,536	438	2,110	3,703	32	6

Table 1: Basic statistics of datasets. # CC: number of connected components. $|CC_{max}|$: size of the largest connected component. Avg $|t_i|$: average number of non-zero elements in term vectors. # Class: number of (overlapping) ground truth classes.

tering algorithms, Metis[8] [15] and Multi-level Regularized Markov Clustering (MLR-MCL)[9] [21]. Both clustering algorithms are also applied on strawman methods.

4.4 Effect of Simplification on Graph Structure

In this section we investigate the impact of topological simplification (or sampling) on the spectrum of the graph. For both CiteSeer and Flickr (results for Wikipedia are similar to that of Flickr) we compute the Laplacian of the graph and then examine the top part of its eigenspectrum (first 2000 eigenvectors). Specifically, in Figure 2 we order the eigenvectors from the smallest one to the largest one (on the X axis) and plot corresponding eigenvalues (on the Y axis).

(a) Citeseer

(b) Flickr
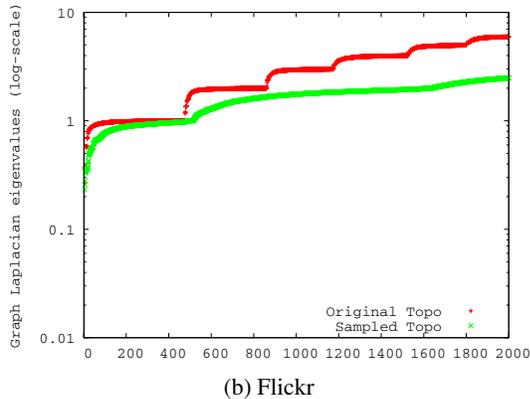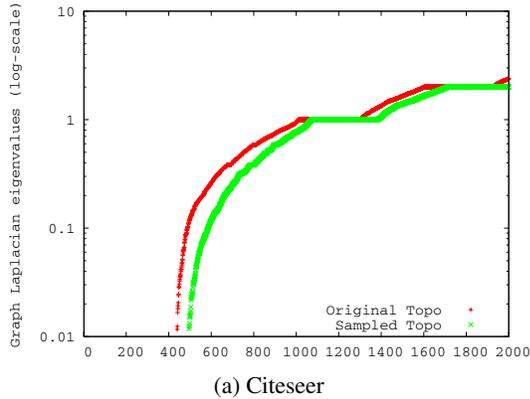
Figure 2: Eigenvalues of graph Laplacian before and after simplification

The multiplicity of 0 as an eigenvalue in such a plot corresponds

---

[8] http://glaros.dtc.umn.edu/gkhome/metis/metis/download
[9] http://www.cse.ohio-state.edu/~satuluri/research.html

to the number of independent components within the graph [18]. For CiteSeer we see an increase in the number of components as a result of topological simplification whereas for Flickr (similarly for Wikipedia) the number of components is unchanged. Our hypothesis is that for datasets like CiteSeer this will have a negative impact on the quality of the resulting clustering. We further hypothesize that our content-based enhancements will help in overcoming this shortfall.

Note that the sum of eigenvalues for the complete spectrum is proportional to the number of edges in the graph [18] so this explains why the plots for the original graphs are slightly above those for the simplified graph even though the overall trends (e.g. spectral gap, relative changes in eigenvalues), except for the number of components, are quite similar for both datasets.

4.5 Clustering Quality

We are interested in comparison between the predicted clustering and the real community structure since group/category information is available for all three datasets. Later in Section 5 we will evaluate CODICIL's performance qualitatively. While it is tempting to use conductance or other cut-based objectives to evaluate the quality of clustering, they only value the structural cohesiveness but not the content cohesiveness of resultant clustering, which is exactly the motivation of content-aware clustering algorithm. Instead, we use average F-score with regard to the ground truth as the clustering quality measure, as it takes content grouping into consideration and ensures a fair comparison among different clusterings. Given a predicted cluster p and with reference to a ground truth cluster g (both in the form of node set), we define the precision rate as $\frac{|p \cap g|}{|p|}$ and the recall rate as $\frac{|p \cap g|}{|g|}$. The F-score of p on g, denoted as $F(p, g)$, is the harmonic mean of precision and recall rates.

For a predicted cluster p, we compute its F-score on each g in the ground truth clustering G and define the maximal obtained as p's F-score on G. That is:

$$F(p, G) = \max_{g \in G} F(p, g) \ . \tag{8}$$

The final F-score of the predicted clustering P on the ground truth clustering G is then calculated as the weighted (by cluster size) average of each predicted cluster's F-score:

$$F(P, G) = \sum_{p \in P} \frac{|p|}{|\mathcal{V}|} F(p, G) \ . \tag{9}$$

This effectively penalizes the predicted clustering that is not well-aligned with the ground truth, and we use it as the quality measure of all methods on all datasets.

4.5.1 CiteSeer

In Figure 3 we show the experiment results on CiteSeer. Since it is known that the network has six communities (i.e. sub-fields in computer science), there is no need to vary l, the number of desired clusters. We report results using Metis (similar numbers were observed with Markov clustering). For PCL-DC, we set the parameter λ to 5 as suggested in the original paper, yielding an F-score of

0.570. The F-scores of SA-Cluster-Inc and L-P-LDA are 0.348 and 0.458, respectively. As we can see clearly in the bar chart, clustering based on topology alone results in a performance well below the state-of-the-art content-aware clustering methods. This is not surprising as the input graph has 438 connected components and therefore most small components were randomly assigned a prediction label. Although such approach has no impact on topology-based measures (e.g. normalized cut or conductance), it greatly spoils the F-score measure against the ground truth. Moreover, topology-based simplification further deteriorates the clustering performance as it creates even more connected components, as we projected in Section 4.4. Neither is LDA able to provide a competitive result, as it is oblivious to link structure embedded in the dataset. Surprisingly though, K-means only manages to produce a very unbalanced clustering (the largest cluster always contains more than 90% of all papers) even after 50 iterations, and its F-score (averaged over five runs) is only 0.336.
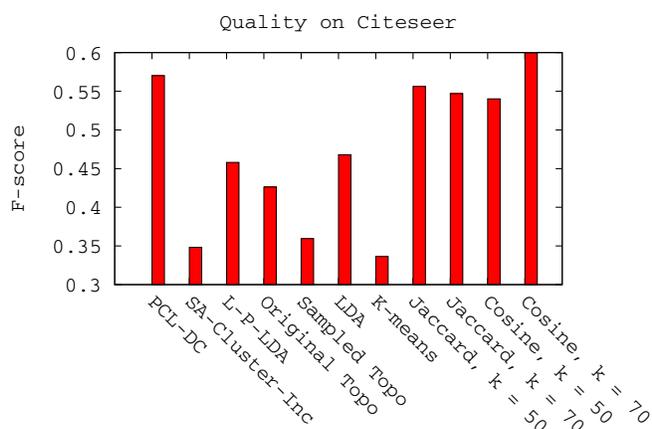
Figure 3: F-score of Metis on CiteSeer

On the other hand, our content-aware approaches (using Metis as the clustering method) were able to handle the issue of disconnection as they also include content-similar edges. For both similarity measures, the F-scores are within 90% range of PCL-DC, and it outperforms PCL-DC when k increases to 70.

While achieving the quality that is comparable with existing methods, the CODICIL series are significantly faster. PCL-DC takes 234 seconds on this dataset and SA-Cluster-Inc requires 306 seconds. LDA finishes in 40 seconds. In contrast, the sum of CODICIL's edge sampling and clustering time never exceeds 1 second. Therefore, the CODICIL methods are at least one order of magnitude faster than state-of-the-art algorithms.

4.5.2 Wikipedia

For the Wikipedia dataset, we were unable to run the experiment on SA-Cluster-Inc, PCL-DC, L-P-LDA, LDA and K-means as their memory and/or running time requirement became prohibitive on this million-node network. For example, storing 10,000 centroids alone in K-means requires 54 GBs).

Figures 4a and 4c plot the performances using MLR-MCL and Metis, respectively. Since category assignments as the ground truth are overlapping, there is no gold standard for the number of clusters. We therefore varied l in both clustering algorithms. Our content-aware clustering algorithms constantly outperforms Sampled Topo by a large margin, indicating that CODICIL methods are able to simplify the network and recover community structure at the

same time. CODICIL methods' F-scores are also on par or better than those of Original Topo.

4.5.3 Flickr

Figure 5a shows the performances of various methods with MLR-MCL on Flickr, where SA-Cluster-Inc, PCL-DC, LDA and K-means can also finish in a reasonable time (L-P-LDA still takes more than 30 hours). Again, l was varied for the clustering algorithm. Similar to results on CiteSeer, CODICIL methods again lead the baselines by a considerable margin. The F-scores of SA-Cluster-Inc, LDA, and K-means never exceed 0.2, whereas CODICIL methods' F-scores are often higher, together with Original & Sampled Topo.

Readers may have noticed that for PCL-DC only three data points ($l = 50, 75, 100$) are obtained. That is because its excessive memory consumption crashed our workstation after using up 16 GBs of RAM for larger l values. We also observe that while PCL-DC generates a group membership distribution over l groups for each vertex, fewer than l communities are discovered. That is, there exist groups of which no vertex is a prominent member. Furthermore, the number of communities discovered is decreasing as l increases (45, 43 and 39 communities for $l = 50, 75, 100$), which is opposite to other methods' trends. All three clusterings' F-scores are less than 0.25. Similarly, multiple runs of K-means (K is set to 400, 800, 1200, and 1600) can only identity roughly 200 communities.

4.6 Scalability

The running time on CiteSeer has already been discussed, and here we focus on Flickr and Wikipedia. For CODICIL methods, the running time includes both edge sampling and clustering stage. The plots' Y-axes (running time) are in log scale.

4.6.1 Flickr

We first report scalability results on Flickr (see Figure 5b). For SA-Cluster-Inc, the value of l (the desired output cluster count), ranging from 100 to 5000, does not affect its running time as it always stays between 1 and 1.25 hours with memory usage around 12GB. The running time of LDA appears, to a large extent, linear in the number of latent topics (i.e. l) specified, climbing up from 2.56 hours ($l = 200$) to 15.88 hours ($l = 1600$). For PCL-DC, the running time with three l values (50, 75, 100) is 0.5, 2.0 and 2.8 hours, respectively.

As for our content-aware clustering algorithms, running them on Flickr requires less than 8 seconds, which is three to four orders of magnitude faster than SA-Cluster-Inc, PCL-DC and LDA. Original Topo takes more than 10 seconds, and Sampled Topo runs slightly faster than CODICIL methods.
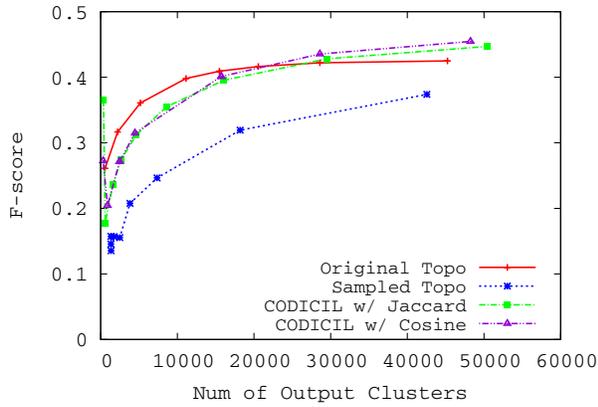
4.6.2 Wikipedia

Original Topo, Sampled Topo and all CODICIL methods finished successfully. The running time is plotted in Figures 4b and 4d. When clustering using MLR-MCL, our methods are at least one order of magnitude faster than clustering based on network topology alone. For Metis, CODICIL is also more than four times faster. The trend lines suggest our methods have promising scalability for analysis on even larger networks.

4.7 Effect of Varying α on F-score

So far all experiments performed fix α at 0.5, meaning equal weights of structural and content similarities. In this sub-section we track how the clustering quality changes when the value of α is varied from 0.1 to 0.9 with a step length of 0.1.
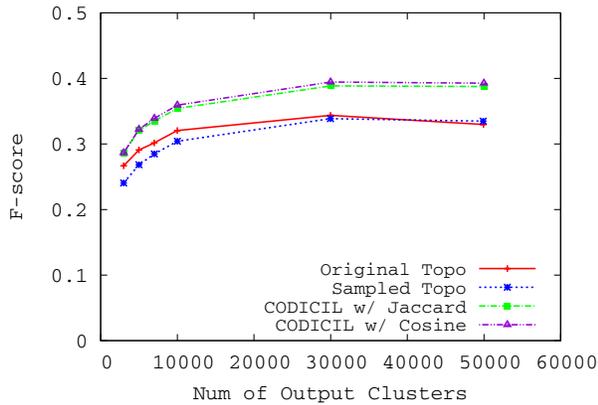
On Wikipedia (Figure 6a) and Citeseer (Figure 6b), F-scores are greatest around $\alpha = 0.5$, supporting the decision of assigning
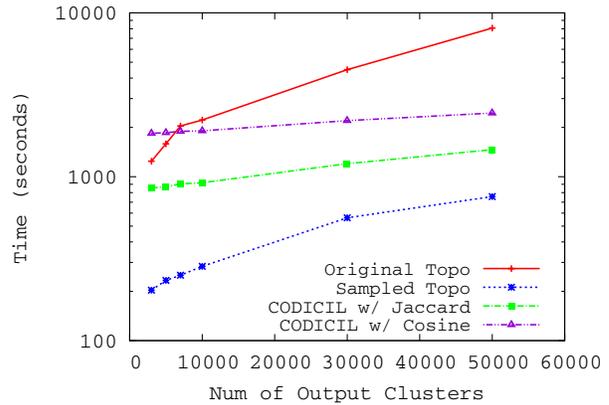
(a) F-score of MLR-MCL on Wikipedia
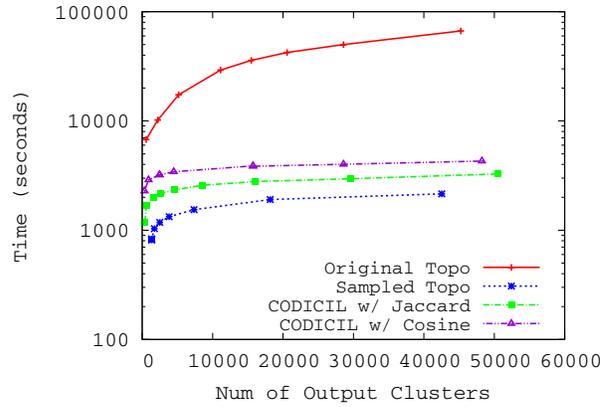
(b) Running time of MLR-MCL on Wikipedia

(c) F-score of Metis on Wikipedia

(d) Running time of Metis on Wikipedia

Figure 4: Experiment Results on Wikipedia

equal weights to structural and content similarities. Results differ on Flickr where F-score is constantly improving when α increases (i.e. more weight assigned to topological similarity).

4.8 Effect of $\mathcal{E}_c$ Constraint on F-score

In Section 3.3.1 we discuss the possibility of constraining content edges within a topological neighborhood for each node v_i. Here we provide a brief review on how the qualities of resultant clusterings are impacted by such constraint. For the sake of space, we focus on the F-scores on Wikipedia and Flickr.

Figures 7a and 7b show F-scores achieved on Wikipedia, using different $\mathcal{E}_c$ constraints. *Full* means no constraint and $TopK$ subroutine searches the whole vertex set $\mathcal{V}$, whereas *1-hop* constrains the search to within a one-hop neighborhood, and likewise for *2-hop*. Plots of *full* and *2-hop* almost overlap with each other, suggesting that searching within the 2-hop neighborhood can provide sufficiently strong content signals on this dataset. For Flickr (Figures 7c and 7d), interestingly *2-hop* and *1-hop* have a slight lead over *full*. This may be an indication that in online social networks, compared with information networks, content similarity between two closely connected users emits stronger community signals.

4.9 Discussions

An interesting observation on the biased edge sampling is that it always results in an improvement in running time. However, sampling just the topology graph results in a clear loss in accuracy whereas content-conscious sampling is much more effective

with accuracies that are on par with the best performing methods at a fraction of the cost to compute. We observe this for all three datasets.

We also find that for probabilistic-model-based methods (PCL-DC, L-P-LDA and LDA) as well as K-means, their running time is at least linear in l, the desired number of output clusters, which becomes a critical drawback in face of large-scale workloads. As the network grows, the number of clusters also increases naturally. Plots on CODICIL methods' running time, on the other hand, suggest a logarithmic increase with regard to the number of clusters, which is more affordable.

5. CASE STUDIES

In this section, we demonstrate the benefits of leveraging content information on two Wikipedia pages: "Machine Learning" and "Graph (Mathematics)".

In the original network, "machine learning" has a total degree of 637, and many of its neighbors (including "1-2-AX working memory task", "Wayne State University Computer Science Department", "Chou-Fasman method", etc.) are at best peripheral to the context. When we sample the graph according to its link structure only, 119 neighbors are retained for "machine learning". Although this eliminates some noise, many others, including the three entries above, are still preserved. Moreover, it also removes during the process many neighbors which should have been kept, e.g. "naive Bayes classifier", "support vector machine", and so on.
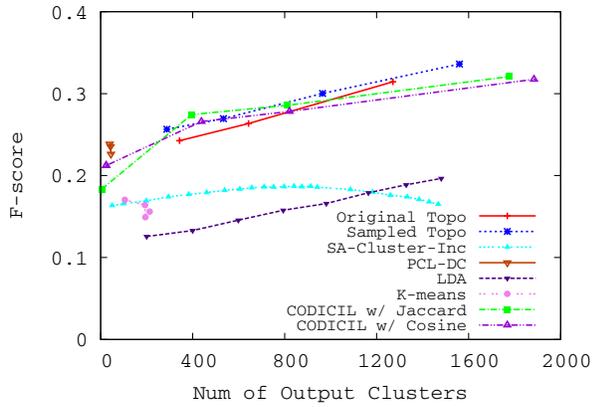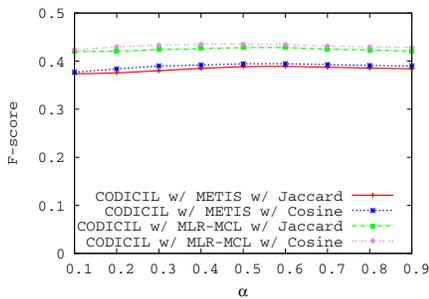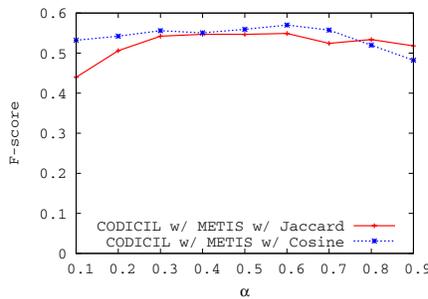
(a) F-score of MLR-MCL on Flickr
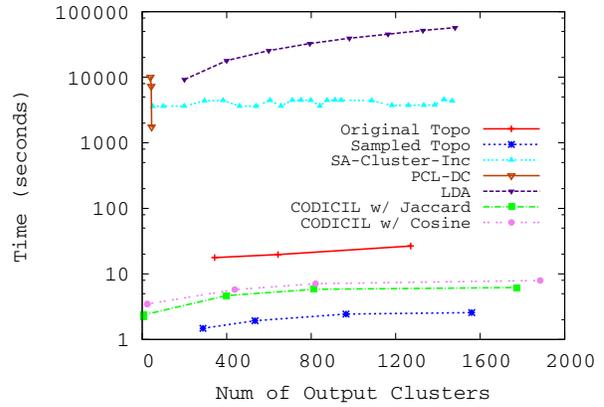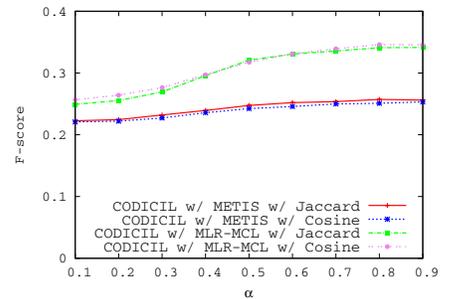
(b) Running time of MLR-MCL on Flickr

Figure 5: Experiment Results on Flickr

(a) Varying α on Wikipedia

(b) Varying α on Citeseer

(c) Varying α on Flickr

Figure 6: Effect of Varying α on F-score (Avg. # Clusters for Wikipedia: 29,414, Avg. # Clusters for Flickr: 1,911)

The CODICIL framework, in contrast, alleviates both problems. Apart from removing noisy edges, it also keeps the most relevant ones. For example, "AdaBoost", "ensemble learning", "pattern recognition" all appear in "machine learning"'s neighborhood in the sampled edge set $\mathcal{E}_{sample}$. Perhaps more interestingly, we find that CODICIL adds "neural network", an edge absent from the original network, into $\mathcal{E}_{sample}$ (recall that it is possible for CODICIL to include an edge even it is not in the original graph, given its content similarity is sufficiently high). This again illustrates the core philosophy of CODICIL: to complement the original network with content information so as to better recover the community structure.

Similar observations can be made on the "Graph (Mathematics)" page. For example, CODICIL removes entries including "Eric W. Weisstein", "gadget (computer science)" and "interval chromatic number of an ordered graph". It also keeps "clique (graph theory)", "Hamiltonian path", "connectivity (graph theory)" and others, which would otherwise be removed if we sample the graph using link structure alone.

6. CONCLUSION

We have presented an efficient and extremely simple algorithm for community identification in large-scale graphs by fusing content and link similarity. Our algorithm, CODICIL, selectively retains edges of high relevancy within local neighborhoods from the fused graph, and subsequently clusters this backbone graph with any content-agnostic graph clustering algorithm.

Our experiments demonstrate that CODICIL outperforms state-of-the-art methods in clustering quality while running orders of magnitude faster for moderately-sized datasets, and can efficiently handle large graphs with millions of nodes and hundreds of millions of edges. While simplification can be applied to the original topology alone with a small loss of clustering quality, it is particularly potent when combined with content edges, delivering superior clustering quality with excellent runtime performance.

7. ACKNOWLEDGEMENTS

8. REFERENCES

[1] C. Aggarwal, Y. Zhao, and P. Yu. Outlier detection in graph streams. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 399–409. IEEE, 2011.

[2] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *JMLR*, 3:993–1022, 2003.

[3] A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *STOC'98*, pages 327–336. ACM, 1998.

[4] M. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC'02*, pages 380–388. ACM, 2002.

[5] D. Cohn and T. Hofmann. The missing link-a probabilistic model of document
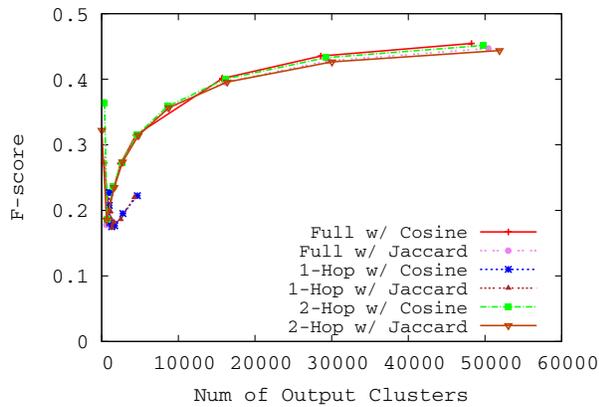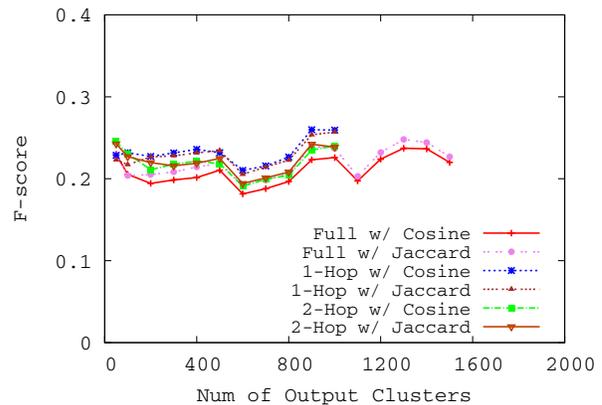
(a) Varying $\mathcal{E}_c$ Constraint, MLR-MCL on Wiki.
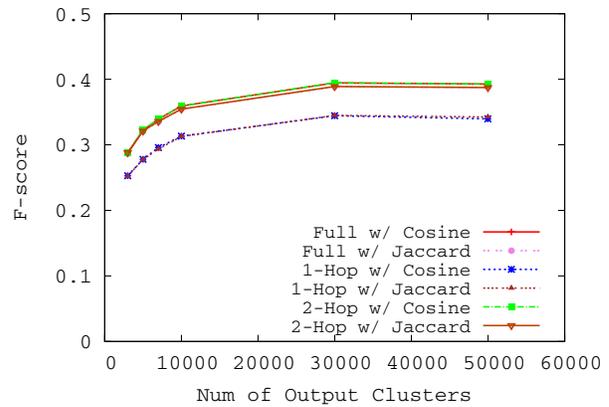
(b) Varying $\mathcal{E}_c$ Constraint, METIS on Wiki.
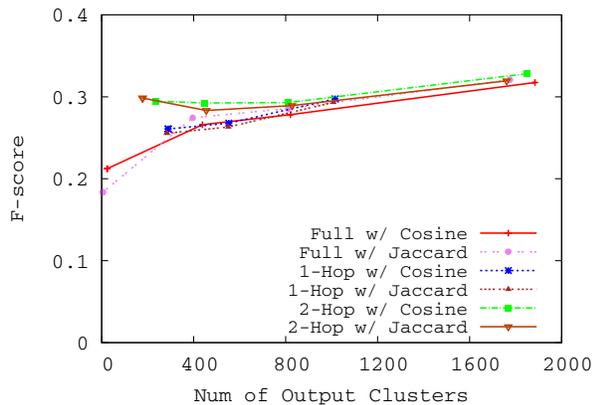
(c) Varying $\mathcal{E}_c$ Constraint, MLR-MCL on Flickr

(d) Varying $\mathcal{E}_c$ Constraint, METIS on Flickr

Figure 7: Effect of $\mathcal{E}_c$ Constraint on F-score

content and hypertext connectivity. In *NIPS'01*, volume 13, page 430. The MIT Press, 2001.

[6] I. Dhillon, Y. Guan, and B. Kulis. Kernel k-means: spectral clustering and normalized cuts. In *SIGKDD'04*, pages 551–556. ACM, 2004.

[7] E. Erosheva, S. Fienberg, and J. Lafferty. Mixed-membership models of scientific publications. *PNAS*, 101(Suppl 1):5220, 2004.

[8] M. Ester, R. Ge, B. Gao, Z. Hu, and B. Ben-Moshe. Joint cluster analysis of attribute data and relationship data: the connected k-center problem. In *SDM'06*, pages 25–46, 2006.

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

[10] E. Fox and J. Shaw. Combination of multiple searches. *NIST SPECIAL PUBLICATION SP*, pages 243–243, 1994.

[11] T. Griffiths and M. Steyvers. Finding scientific topics. *PNAS*, 101(Suppl 1):5228, 2004.

[12] S. Günnemann, B. Boden, and T. Seidl. Db-csc: a density-based approach for subspace clustering in graphs with feature vectors. In *PKDD 2011*, pages 565–580. Springer-Verlag, 2011.

[13] D. Hanisch, A. Zien, R. Zimmer, and T. Lengauer. Co-clustering of biological networks and gene expression data. *Bioinformatics*, 18(suppl 1):S145–S154, 2002.

[14] D. Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, 24(2):383–413, 1999.

[15] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[16] J. Leskovec, K. Lang, and M. Mahoney. Empirical comparison of algorithms for network community detection. In *Proceedings of the 19th international conference on World wide web*, pages 631–640. ACM, 2010.

[17] A. Maiya and T. Berger-Wolf. Sampling community structure. In *WWW 2010*, pages 701–710. ACM, 2010.

[18] B. Mohar. The laplacian spectrum of graphs. *Graph theory, combinatorics, and applications*, 2:871–898, 1991.

[19] M. Montague and J. Aslam. Relevance score normalization for metasearch. In *CIKM'01*, pages 427–433. ACM, 2001.

[20] R. Nallapati, A. Ahmed, E. Xing, and W. Cohen. Joint latent topic models for text and citations. In *SIGKDD'08*, pages 542–550. ACM, 2008.

[21] V. Satuluri and S. Parthasarathy. Scalable graph clustering using stochastic flows: applications to community discovery. In *SIGKDD'09*, pages 737–746. ACM, 2009.

[22] V. Satuluri, S. Parthasarathy, and Y. Ruan. Local graph sparsification for scalable clustering. In *SIGMOD 2011*, pages 721–732. ACM, 2011.

[23] A. Strehl and J. Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *The Journal of Machine Learning Research*, 3:583–617, 2003.

[24] W. Tang, Z. Lu, and I. Dhillon. Clustering with multiple graphs. In *ICDM'09*, pages 1016–1021. IEEE, 2009.

[25] Y. Tian, R. A. Hankins, and J. M. Patel. Efficient aggregation for graph summarization. In *SIGMOD'08*, pages 567–580, 2008.

[26] I. Ulitsky and R. Shamir. Identification of functional modules using network topology and high-throughput data. *BMC Systems Biology*, 1(1):8, 2007.

[27] S. van Dongen. Graph clustering by flow simulation. *PhD Thesis*, 2000.

[28] T. Yang, R. Jin, Y. Chi, and S. Zhu. Combining link and content for community detection: a discriminative approach. In *SIGKDD'09*, pages 927–936. ACM, 2009.

[29] D. Zhou, E. Manavoglu, J. Li, C. Giles, and H. Zha. Probabilistic models for discovering e-communities. In *WWW'06*, pages 173–182. ACM, 2006.

[30] Y. Zhou, H. Cheng, and J. Yu. Clustering large attributed graphs: An efficient incremental approach. In *ICDM 2010*, pages 689–698. IEEE, 2010.

[31] S. Zhu, K. Yu, Y. Chi, and Y. Gong. Combining content and link for classification using matrix factorization. In *SIGIR'07*, pages 487–494. ACM, 2007.