Scalable Graph Clustering Using Stochastic Flows: Applications to Community Discovery

Venu Satuluri
Dept. of Computer Science and Engineering
The Ohio State University
satuluri@cse.ohio-state.edu

Srinivasan Parthasarathy
Dept. of Computer Science and Engineering
The Ohio State University
srini@cse.ohio-state.edu

ABSTRACT

Algorithms based on simulating stochastic flows are a simple and natural solution for the problem of clustering graphs, but their widespread use has been hampered by their lack of scalability and fragmentation of output. In this article we present a multi-level algorithm for graph clustering using flows that delivers significant improvements in both quality and speed. The graph is first successively coarsened to a manageable size, and a small number of iterations of flow simulation is performed on the coarse graph. The graph is then successively refined, with flows from the previous graph used as initializations for brief flow simulations on each of the intermediate graphs. When we reach the final refined graph, the algorithm is run to convergence and the high-flow regions are clustered together, with regions without any flow forming the natural boundaries of the clusters. Extensive experimental results on several real and synthetic datasets demonstrate the effectiveness of our approach when compared to state-of-the-art algorithms.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data Mining; G.2.2 [**Graph Theory**]: Graph Algorithms

General Terms

Algorithms,Performance

Keywords

Graphs, Clustering, Communities, Networks

1. INTRODUCTION

Clustering graphs or discovering communities within networks is an important problem with many applications in a number of disciplines. Examples abound and range from social network analysis[14] to image segmentation[17] and from analyzing protein interaction networks[2] to the circuit layout problem[6].

Given the importance of the problem numerous solutions have been proposed in the literature. Spectral methods that target weighted cuts [17] form an important class of such algorithms and are shown to be very effective for problems such as image segmentation. Multi-level graph partitioning algorithms such as Metis[9] are well known to scale well,and have been used in studies of some of the biggest graph datasets[11]. Graclus [4] is another multi-level partitioning algorithm that optimizes weighted cuts (including normalized cuts) by optimizing an equivalent weighted kernel K-means loss function. The avoidance of expensive eigenvector computation gives Graclus a big boost in speed while retaining or improving upon the quality of spectral approaches. Divisive/agglomerative approaches have been popular in network analysis[14], but they are expensive and do not scale well[3]. Markov Clustering (MCL)[5], a graph clustering algorithm based on (stochastic) flow simulation, has proved to be highly effective at clustering biological networks [2, 13].

Here, we focus on the class of flow-based graph clustering algorithms represented by MCL. MCL offers several advantages in that it is an elegant approach based on the natural phenomenon of flow, or transition probability, in graphs. It has been shown to be robust to topological noise effects (a desirable property for a number of domains)[2], and while not completely non-parametric, varying a simple parameter can result in clusterings of different granularities. However, inspite of its popularity in the bioinformatics community for the above reasons, MCL has drawn limited attention from the data mining community primarily because it does not scale very well even to moderate sized graphs[3]. Additionally, the algorithm tends to fragment communities, a less than desirable feature in many situations.

In this article we seek to develop an algorithm that retains the strengths of MCL while redressing its weaknesses. We first analyze the basic MCL algorithm carefully to understand the cause for these two limitations. We then identify a simple regularization step that can help alleviate the fragmentation problem. We call this algorithm Regularized-MCL (R-MCL). Subsequently we realize a scalable multi-level variant of R-MCL. The basic idea of the multi-level procedure is to coarsen the graph (in a manner reminiscent of Metis), run R-MCL on the coarsened graph, and then refine the graph in incremental steps. The central intuition is that using flow values derived from simulation on coarser graphs can lead to good initializations of flow in the refined graphs. Key to the refinement operation is a novel way to project flows such that the sanctity of the clustering algo-

KDD'09, June 28–July 1, 2009, Paris, France.
Copyright 2009 ACM 978-1-60558-495-9/09/06 ...$5.00.

rithm is maintained. The multi-level approach also allows us to obtain large gains in speed. We refer to this algorithm as Multi-Level Regularized MCL (MLR-MCL).

In our empirical study we compare and contrast R-MCL and MLR-MCL with the original MCL algorithm[5], Graclus[4] and Metis[9] along the twin axes of scalability and quality on several real and synthetic datasets. Key highlights of our study include:

- **MLR-MCL vs R-MCL and MCL**: MLR-MCL typically outperforms both R-MCL and MCL in terms of quality and delivers performance that is roughly 2-3 orders of magnitude faster than MCL.
- **MLR-MCL vs. Graclus and Metis** : MLR-MCL delivers significant (10-15%) improvements in N-Cut values over Graclus in 4 of 6 datasets. We also typically outperform Graclus in terms of speed and are competitive with Metis.
- **MLR-MCL vs. Graclus and MCL on PPI networks**: The top 8 clusters of proteins found by MLR-MCL are rated better than the best cluster returned by either Graclus and MCL.

2. PRELIMINARIES

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be our input graph with $\mathcal{V}$ and $\mathcal{E}$ denoting the node set and edge set respectively. Let A be the $|\mathcal{V}| \times |\mathcal{V}|$ adjacency matrix corresponding to the graph, with $A(i, j)$ denoting the weight of the edge between the vertex v_i and the vertex v_j. This weight can represent the strength of the interaction in the original network - e.g. in an author collaboration network, the edge weight between two authors could be the frequency of their collaboration. If the graph is unweighted, then the weight on each edge is fixed to 1. As many interaction networks are undirected, we also assume that $\mathcal{G}$ is undirected, although our method is easy to extend to directed graphs. Therefore A will be a symmetric matrix.

2.1 Stochastic matrices and flows

A column-stochastic matrix is simply a matrix where each column sums to 1. A column stochastic (square) matrix M with as many columns as vertices in a graph $\mathcal{G}$ can be interpreted as the matrix of the *transition probabilities* of a random walk (or a Markov chain) defined on the graph. The i^{th} column of M represents the transition probabilities *out* of the v_i; therefore $M(j, i)$ represents the probability of a transition from vertex v_i to v_j. We use the terms stochastic matrix and column-stochastic matrix interchangeably.

We also refer to the transition probability from v_i to v_j as the *stochastic flow* or simply the *flow* from v_i to v_j. Correspondingly, a column-stochastic transition matrix of the graph $\mathcal{G}$ is also referred to as a flow matrix of $\mathcal{G}$ or simply a *flow of* $\mathcal{G}$. Given a flow matrix M, the i^{th} column contains the flows *out* of node v_i, or its *out-flows*; correspondingly the i^{th} row contains the *in-flows.* Note that while all the columns (or out-flows) sum to 1, the rows (or in-flows) are not required to do so.

The most common way of deriving a column-stochastic transition matrix M for a graph is to simply normalize the columns of the adjacency matrix to sum to 1

$$M(i, j) = \frac{A(i, j)}{\sum_{k=1}^{n} A(k, j)}$$

In matrix notation, $M := AD^{-1}$, where D is the diagonal degree matrix of $\mathcal{G}$ with $D(i, i) = \sum_{j=1}^{n} A(j, i)$. We will refer to this particular transition matrix for the graph as the *canonical transition matrix* M_G. However, it is worth keeping in mind that one can associate other stochastic matrices with the graph $\mathcal{G}$.

Both MCL and our methods introduced in Section 3 can be thought of as simulating stochastic flows (or simulating random walks) on graphs according to certain rules. For this reason, we refer to these processes as *flow simulations.*

2.2 Markov Clustering (MCL) Algorithm

We next describe the Markov Clustering (MCL) algorithm for clustering graphs, proposed by Stijn van Dongen [5], in some detail as it is relevant to understanding our own method.

The MCL algorithm is an iterative process of applying two operators - *expansion* and *inflation* - on an initial stochastic matrix M, in alternation, until convergence. Both expansion and inflation are operators that map the space of column-stochastic matrices onto itself. Additionally, a *prune* step is performed at the end of each inflation step in order to save memory. Each of these steps is defined below:

Expand: Input M , output M_{exp}.

$$M_{exp} = Expand(M) \overset{def}{=} M * M$$

The i^{th} column of M_{exp} can be interpreted as the final distribution of a random walk of length 2 starting from vertex v_i, with the transition probabilities of the random walk given by M. One can take higher powers of M instead of a square (corresponding to longer random walks), but this gets computationally prohibitive very quickly.

Inflate: Input M and inflation parameter r, output M_{inf}.

$$M_{inf}(i, j) \overset{def}{=} \frac{M(i, j)^r}{\sum_{k=1}^{n} M(k, j)^r}$$

M_{inf} corresponds to raising each entry in the matrix M to the power r and then normalizing the columns to sum to 1. By default $r = 2$. Because the entries in the matrix are all guaranteed to be less than or equal to 1, this operator has the effect of exaggerating the inhomogeneity in each column (as long as $r > 1$). In other words, flow is strengthened where it is already strong and weakened where it is weak.

Prune: In each column, we remove those entries which have very small values (where "small" is defined in relation to the rest of the entries in the column), and the retained entries are rescaled to have the column sum to 1. This step is primarily meant to reduce the number of non-zero entries in the matrix and hence save memory.We use the *threshold pruning* heuristic, where we compute a threshold based on the average and maximum values within a column, and prune entries below the threshold. [5]

Pseudo-code for MCL is presented in Algorithm 1. The addition of self-loops to the input graph avoids dependence of the flow distribution on the length of the random walk simulated so far, besides ensuring at least one non-zero entry per column.

Intuitively, the MCL process may be understood as expanding and contracting the flow in the graph alternately. The expansion step spreads the flow out of a vertex to potentially new vertices and also enhances the flow to those

Algorithm 1 MCL

```
$A := A + I$ // Add self-loops to the graph
$M := AD^{-1}$ // Initialize M as the canonical transition matrix

repeat
    $M := M_{exp} :=$ Expand($M$)
    $M := M_{inf} :=$ Inflate($M, r$)
    $M :=$ Prune($M$)
until $M$ converges

Interpret $M$ as a clustering
```

vertices which are reachable by multiple paths. This has the effect of enhancing within-cluster flows as there are more paths between two nodes that are in the same cluster than between those in different clusters. However, just applying the expansion step repeatedly will result in all the columns of M becoming equal to the principal eigenvector of the canonical transition matrix M_G. The inflation step is meant to prevent this from happening by introducing a non-linearity into the process, while also having the effect of strengthening intra-cluster flow and weakening inter-cluster flow. At the start of the process, the distribution of flows out of a node is relatively smooth and uniform; as more iterations are executed, the distribution becomes more and more peaked. *Crucially, all the nodes within a tightly-linked group of nodes will start to flow to one node within the group towards the end of the process.* This allows us to identify all the vertices that flow to the same "attractor" node as belonging to one cluster.

Interpretation of M as a clustering: As just mentioned, after some number of iterations, most of the nodes will find one "attractor" node to which all of their flow is directed i.e. there will be only one non-zero entry per column in the flow matrix M. We declare convergence at this stage, and assign nodes which flow into the same node as belonging to one cluster.

2.3 Limitations of MCL

The MCL algorithm is a simple and intuitive algorithm for clustering graphs that takes an approach that is different from that of the majority of other approaches to graph clustering such as spectral clustering [17, 4], divisive/agglomerative clustering [14], heuristic methods [10] and so on. Further more, it does not require a specification of the number of clusters to be returned; the coarseness of the clustering can instead be indirectly affected by varying the inflation parameter r, with lower values of r (up to 1) leading to coarser clusterings of the graph. MCL has received a lot of attention in the bioinformatics field, with multiple researchers finding it to be very effective at clustering biological interaction networks ([2, 13]).

However, there are two major limitations to MCL:

Lack of scalability: That MCL is slow has been noted by data mining researchers before ([7, 3]). The Expand step, which involves matrix multiplication, is very time consuming in the first few iterations when many entries in the flow matrix have not been pruned out and is the main component of the overall running time. The Expand step in the very first iteration of the algorithm in particular takes $O(\sum_{i=1}^{|\mathcal{V}|} d_i^2)$ operations, where d_i is the degree of vertex v_i, which is unacceptably slow for large graphs. Expansion steps in subsequent iterations take $O(|\mathcal{V}|k^2)$ time, where k is the average number of non-zero entries per column. In the first few iterations before the flow matrix becomes sparse, k is typically in the range of hundreds or thousands for large graphs, still leading to an unacceptable time complexity.

Fragmentation of output: MCL tends to produce too many clusters. For example, on the yeast protein-protein interaction network of 4741 nodes, MCL outputs 1416 clusters. (We obtained similar results on other datasets, as well as with varying values of the inflation parameter r.) While it still manages to find some significant clusters (as evidenced by its success in bioinformatics [2]), clearly such high fragmentation is undesirable.

3. OUR ALGORITHMS

We seek to develop an algorithm for graph clustering that retains the strengths of MCL while alleviating the weaknesses. We do this by first making a modification to the basic MCL process, resulting in an algorithm we call *Regularized MCL*, and then we embed this latter algorithm in a multi-level framework that further improves both the quality of the output and the speed of the algorithm.

A weight transformation step: Before discussing the algorithm, we first describe a weight transformation step that we apply to the input graphs. This step was suggested by Dhillon et al. [4], who use it as part of the coarsening process in their multi-level framework. Given the input adjacency matrix A and the degree diagonal matrix D, define the transformed adjacency matrix A^* as:

$$A^*(i,j) = \frac{A(i,j)}{D(i,i)} + \frac{A(i,j)}{D(j,j)}$$

The purpose of the above step is to downweight the edges involving high-degree (or hub) nodes, as they can have an undue influence on the clustering process.

3.1 Regularized MCL (R-MCL)

Why does MCL output so many clusters? One way of looking at the issue is to understand it as MCL allowing the columns of many pairs of neighboring nodes in the flow matrix M to diverge significantly. This happens because the MCL algorithm uses the adjacency structure of the input graph only at the start, to initialize the flow matrix to the canonical transition matrix M_G. After that, the algorithm works only with the current flow matrix M, and there is nothing in the algorithm that prevents the columns of neighboring nodes to differ widely without any penalty. This is what allows MCL to "overfit" the graph by outputting too many clusters.

We seek to address this issue by *regularizing* or *smoothing* the flow distributions out of a node w.r.t. its neighbors. Let $\mathbf{q}_i$, i=1:k, be the flow distributions of the k neighbors of a given node in the graph. (Each $\mathbf{q}_i$ is basically a column from the current flow matrix M.) Let w_i, i=1:k, be the respective normalized edge weights, i.e. $\sum_{i=1}^{k} w_i = 1$. Note that since we add self-loops, each node is also one of its own neighbors. We ask the following question: how do we update the flow distribution out of a given node (call it $\mathbf{q}^*$) so that it is, in some sense, the least divergent from the flow distributions of its neighbors? Following [19], we can formalize this requirement for $\mathbf{q}^*$ as:

$$\mathbf{q}^* = \arg\min_{\mathbf{q}} \sum_{i=1}^{k} w_i KL(\mathbf{q_i}||\mathbf{q})$$

where $KL(\mathbf{p}||\mathbf{q})$ is the KL divergence between two probability distributions $\mathbf{p}$ and $\mathbf{q}$ - a commonly used divergence measure for probability distributions - defined as:

$$KL(\mathbf{p}||\mathbf{q}) = \sum_x \mathbf{p}(x) \log \frac{\mathbf{p}(x)}{\mathbf{q}(x)}$$

There exists a closed form solution for $\mathbf{q}^*$ [19, 15], and can be shown to be:

$$\mathbf{q}^*(j) = \sum_{i=1}^{k} w_i \mathbf{q}_i(j) \tag{1}$$

Hence, we replace the *Expand* operator in the MCL process with a new operator which updates the flow distribution of each node according as Equation 1. We call this the *Regularize* operator, and it can be conveniently expressed in matrix notation as right multiplication with the canonical transition matrix M_G of the graph.

$$Regularize(M) = M_{reg} \overset{def}{=} M * M_G$$

Pseudo-code for Regularized MCL is given in Algorithm 2. The Inflate and Prune steps are the same as for MCL, and the interpretation of M as a clustering is also the same as has been described in Section 2.2.

Algorithm 2 Regularized MCL

```
$A := A + I$ // Add self loops and transform weights
$M := M_G := AD^{-1}$ //Initialize M as the canonical transition matrix

repeat
    $M := M_{reg} := M * M_G$
    $M := M_{inf} := Inflation(M, r)$
    $M := Prune(M)$
until $M$ converges

Interpret $M$ as a clustering as described in Section 2.2
```

While, as we shall see in Section 4, Regularized MCL does produce fewer clusters of better quality, it still suffers from the scalability issues of the original MCL. We address this issue next and also discuss how the qualitative performance of Regularized MCL can be further improved.

3.2 Multi-level Regularized MCL (MLR-MCL)

We next explain a multi-level version of Regularized MCL, which we call Multi-level Regularized MCL, or MLR-MCL. The main intuition behind using a multi-level framework in our context is that the flow values resulting from simulation on the coarser graphs can be effectively used to initialize the flows for simulation on the bigger graphs. The algorithm also runs much faster because the initial few iterations, which are also the most time taking, are run on the smallest graphs first. [1]

A schematic providing a high-level overview of our multi-level algorithm is given in Figure 1. The pseudo-code is given in Algorithm 3. MLR-MCL operates in three phases:

1. **Coarsening**: The input graph $\mathcal{G}$ is coarsened successively into a series of smaller graphs $\mathcal{G}_1, \mathcal{G}_2, \ldots$ until we are left with a graph $\mathcal{G}_l$ of manageable size (a few hundred nodes typically). Each coarsening step consists of first constructing a *matching* on the graph, where a matching is defined as a set of edges no two of which are incident on the same vertex. The two vertices that are incident on each edge in the matching are collapsed to form a super-node, and the edges of the super-node are the union of the edges of its constituent nodes. We use two arrays in each coarse graph, NodeMap1 and NodeMap2, to keep track of the coarsening. NodeMap1 maps a node in the coarse graph to its first constituent node; similarly NodeMap2 maps a node to its second constituent node (at most two nodes can be collapsed to one super node). We use a particular kind of matching known as *heavy edge matching*; both heavy edge matching and efficient randomized algorithms for constructing it are described elsewhere [9].

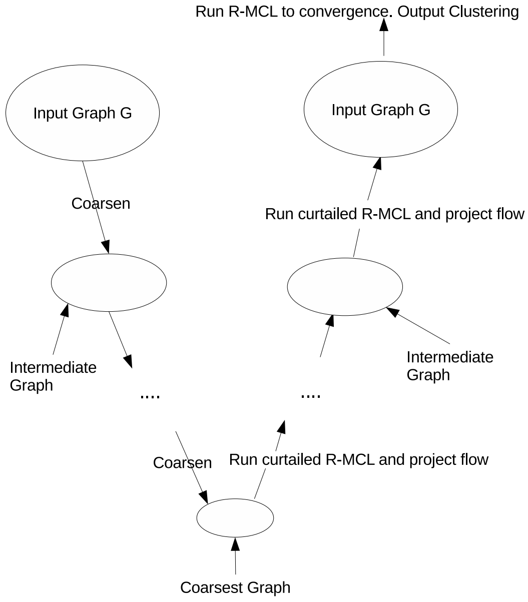
Run R-MCL to convergence. Output Clustering
Input Graph G
Input Graph G
Coarsen
Run curtailed R-MCL and project flow
Intermediate Graph
Intermediate Graph
....
....
Coarsen
Run curtailed R-MCL and project flow
Coarsest Graph

Figure 1: A high-level overview of Multi-level Regularized MCL.

2. **Curtailed R-MCL along with refinement**: Beginning with the coarsest graph, R-MCL is run for a few iterations (typically 4 to 5). We refer to this abbreviated version of R-MCL as Curtailed R-MCL. The flow values at the end of this Curtailed R-MCL run are then projected on to the refined graph of the current graph as per Algorithm 4, and R-MCL is run again for a few iterations on the refined graph, and this is repeated until we reach the original graph.

3. **R-MCL on original graph**: With flow values initialized from the previous phase, R-MCL is run on the final graph until convergence. The flow matrix at the end is converted into a clustering in the usual way, with all the nodes that flow into the same “attractor” node being assigned into one cluster.

What is the intuition behind running R-MCL for only a

[1] We discuss why it is hard to similarly embed MCL as well in a multi-level framework in the extended version of this paper [15].

few iterations on the coarse graphs from $\mathcal{G}_l$ down to $\mathcal{G}_1$? We do this as we do not want the flows in a coarse graph to converge; if we run R-MCL until convergence on one of the intermediate graphs, then the same cluster assignments will likely carry over till the original graph, thus not utilizing the additional adjacency information present in the bigger graphs. At the same time we want the flow values to capture some of the high-level cluster structure of the coarser graphs, and also want the flow matrix to be sparse enough to make running R-MCL on the bigger graph computationally tractable. So we strike a balance and run R-MCL for a small number of iterations. In practice, we have observed that running R-MCL for 4 to 5 iterations on the intermediate graphs gives good results.

Algorithm 3 Multi-level Regularized MCL

```
Input: Original graph $\mathcal{G}$, Inflation parameter $r$, Size of coarsest graph $c$

// Phase 1: Coarsening
// Coarsen graph successively down to at most $c$ nodes.
$\{\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_l\}$ = CoarsenGraph($\mathcal{G}, c$)
// $\mathcal{G}_0$ is the original graph and $\mathcal{G}_l$ is the coarsest graph

// Phase 2: Curtailed R-MCL along with refinement
// Initialize $M$ to canonical flow matrix of the coarsest graph $\mathcal{G}_l$
$M := M_G := A_l D_l^{-1}$

// Starting with the coarsest graph, iterate through successively refined graphs.
for $i = l$ down to 1 do

    // Run R-MCL for a small number of iterations.
    for small number of iterations do
        $M$ := Regularize($M$) = $M * M_G$
        $M$ := Inflate($M$, $r$)
        $M$ := Prune($M$)
    end for

    // Project flow from the coarse graph $\mathcal{G}_i$ onto the refined graph $\mathcal{G}_{i-1}$
    $M$ := ProjectFlow($\mathcal{G}_i$, $M$)
    // Canonical transition matrix of the refined graph $\mathcal{G}_{i-1}$, for the next round of Curtailed R-MCL
    $M_G := A_{i-1} D_{i-1}^{-1}$
end for

// Phase 3: Run R-MCL on original graph until convergence
repeat
    $M$ := Regularize($M$) = $M * M_G$
    $M$ := Inflate($M$)
    $M$ := Prune($M$)
until $M$ converges
Interpret $M$ as a clustering as described in Section 2.2
```

Algorithm 4 ProjectFlow

```
Inputs: Coarse graph $\mathcal{G}_c$, Flow on the coarse graph $M_c$.
Output: Projected flow matrix on the refined graph $M_r$

NodeMap1 := $\mathcal{G}_c$.NodeMap1
NodeMap2 := $\mathcal{G}_c$.NodeMap2

for each non-zero entry (i,j) in $M_c$ do
    $M_r$(NodeMap1($i$),NodeMap1($j$)) :=$M_c$(i,j)
    $M_r$(NodeMap1($i$),NodeMap2($j$)) :=$M_c$(i,j)
    $M_r$(NodeMap2($i$),NodeMap1($j$)) :=0
    $M_r$(NodeMap2($i$),NodeMap2($j$)) :=0
end for
return $M_r$
```

The problem of flow projection: The remaining part of MLR-MCL is the algorithm for projecting flow from a coarse (smaller) graph to a refined (bigger) graph. Projection of flow is concerned with using the flow values of a smaller graph to provide a good initialization of the flow values in the bigger graph. It is not obvious at first sight how this should be done - since there are two nodes in the bigger graph corresponding to each node in the smaller graph, a flow value between two nodes in the smaller graph must be used to derive the flow values between four pairs of nodes in the bigger graph. To look at it another way, if n_c is the size of the coarse graph, then the n_c^2 entries of the flow matrix of the coarse graph must be used to derive the $4 * n_c^2$ entries of the refined graph. How to do this?
Our solution: The naive strategy here is to assign the flow between two nodes in the refined graph as the flow between their respective parents in the coarse graph. However, this doubles the number of nodes that any node in the refined graph flows out to. This, combined with the fact that the out-flows of each node sum to 1, results in excessive smoothing of the out-flows of each node. (Recall that as the MCL process converges, the out-flow distribution of the nodes gets more and more peaked.) Hence, we instead choose only one child node for each parent node and project all the flow into the parent node to the chosen child node.

However, this raises the question: which child node do we pick in order to assign all the flow into? It turns out that it doesn't matter which child node we pick, as long as for each parent, the choice is consistent. We state this in Theorem 1; the proof is provided in the extended version of this paper [15].

THEOREM 1. *The MLR-MCL algorithm produces the same final clustering regardless of which child node is picked at each parent node to be assigned all its in-flows.*

For this reason, for each node v_i in the coarse graph, we arbitrarily pick the first child node NodeMap1(i) and assign all the flow that was going into v_i to NodeMap1(i).

While we treat the two child nodes asymmetrically when we are assigning the flows into them, the flows *out* of the two child nodes are assigned the same values. This being the case, can the algorithm treat these two nodes differently? Recall that the *Regularize* step utilizes the adjacency information in the graph by assigning a linear combination of the flows of a node's neighbors as the flows of a node. *Hence, even if NodeMap1(i) and NodeMap2(i) start out with the same flows out of them , they will have different flows out of them after the Regularize step if they have different neighbors.* This ensures that the additional adjacency information that is present in the refined graph is used to re-adjust the flows of the nodes.

3.3 Discussion of MLR-MCL

We now discuss the time-complexity of MLR-MCL and the quality of its output.
Scalability and time-complexity: The main component of the running time of R-MCL is the Regularize step which involves matrix multiplication. The time complexity of the Regularize step in the first iteration is $O(\sum_{i=1}^{|\mathcal{V}|} d_i^2)$, similar to that for MCL. The subsequent iterations require $O(k|\mathcal{E}|)$, where k is the average number of non-zero entries per column in the flow matrix. This rules out the direct use of R-MCL on large graphs.

The analysis is similar with MLR-MCL, but with a crucial difference. The Regularize step of the first iteration is carried out on the coarsest graph, so the time complexity of $O(\sum_{i=1}^{|\mathcal{V}|} d_i^2)$ for the first Regularize step now applies to the coarsest graph. As the coarsest graph is small, this is an affordable step. As the algorithm proceeds,we simulate flow on bigger graphs, but at the same time the flow matrix also becomes sparser,enabling the algorithm to scale easily. Empirically we observe that after the first Curtailed R-MCL run on the coarsest graph, there are rarely more than a few tens of non-zero entries per column. The overall time complexity of the algorithm is well approximated as $O(k|\mathcal{E}| + \sum_{i=1}^{|\mathcal{V}_c|} d_i^2)$, where the d_is are the degrees of the nodes in the coarsest graph, and k is a small constant, typically in the tens.

Quality: Embedding R-MCL in a multi-level framework leads to improvements in quality as well, as we show in Section 4.2. Coarsening the graph allows the algorithm to utilize the global topology of the graph to provide an effective initialization of the flow values for the simulations on the bigger graphs. At the same time because iterations are run on the final graph as well, the algorithm is able to adjust suitably to the local topology. All of this translates into clusters that are of higher quality than those that are produced by either MCL or R-MCL.

4. EXPERIMENTS

We describe extensive evaluation for the methods proposed in the paper. We performed experiments on 7 real world datasets; four of these were author collaboration networks - three from the Physics community (Astro, HepPh and HepTh), and one from the Computer Science community (DBLP) -, one is a who-trusts-whom network from Epinions.com (Epinions)[2], one is a paper citation network (Cora) [3] and the last is the Protein-Protein Interaction network of yeast (Yeast-PPI)[4]. Details are given in Table 1. We also synthetically generated 5 datasets of increasing size for evaluating scalability, details given in Table 2.

The experiments were performed on a dual core machine (Dual 250 Opteron) with 2.4GHz of processor speed and 8GB of main memory. The software for each of our baselines was downloaded from the respective author's webpages. Our implementation was in C/C++, as were the implementations of all of our baselines. The matrices were stored using a sparse matrix representation in our implementation.

4.1 Evaluation criteria

Except for the Yeast PPI network where we use a domain-specific evaluation, we will use *normalized cut* or *conductance* as our measure of cluster quality.

The *normalized cut* of a cluster $\mathcal{C}$ in the graph $\mathcal{G}$ is defined as (where A is the adjacency matrix of the graph)

$$Ncut(\mathcal{C}) = \frac{\sum_{v_i \in \mathcal{C}, v_j \notin \mathcal{C}} A(i,j)}{\sum_{v_i \in \mathcal{C}} degree(v_i)}$$

Name	$\vert\mathcal{V}\vert$	$\vert\mathcal{E}\vert$	Avg. degree
Cora	17604	74180	8.42
Dblp	16196	45031	5.56
Astro-Ph	17903	196972	22.00
Hep-Ph	11204	117619	21.00
Hep-Th	8638	24806	5.78
Epinions	75877	405739	10.69
Yeast-PPI	4741	15148	6.39

Table 1: Details of real datasets

Name	$\vert\mathcal{V}\vert$	$\vert\mathcal{E}\vert$	No. of clusters
synth_1M	1000, 000	58, 256, 766	10000
synth_500K	500, 000	22, 682, 795	5000
synth_100K	100, 000	3, 447, 123	1000
synth_50K	50, 000	2, 272, 534	500
synth_10K	10, 000	319, 383	100

Table 2: Details of synthetic datasets

The normalized cut of a cluster then, is simply the number of edges that are "cut" when dividing this cluster from the rest of the graph, normalized by the total degree of the cluster.

The normalized cut of a clustering $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$ is the sum of the normalized cuts of the individual clusters.

$$Ncut(\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}) = \sum_{i=1}^{k} Ncut(\mathcal{C}_i)$$

The *average normalized cut* of a clustering is the average of the normalized cuts of each of the constituent clusters, and lies between 0 and 1. Averaging the N-Cut score allows us to compare clustering arrangements with different numbers of clusters.

The Normalized Cut criterion has been commonly noted as capturing very well the intuitive notion of the "goodness" of a particular clustering [12, 8, 17]. An additional reason for choosing this criterion is that one of our baselines, Graclus [4] is one of the state-of-the-art algorithms for optimizing precisely this criterion.

4.2 Comparison with MCL

In our first set of experiments we compare the performance of R-MCL and MLR-MCL with the baseline MCL algorithm. Table 3 documents results obtained on 6 real datasets. The key trends one can glean from this study are as follows. [5]

First, MLR-MCL clearly dominates both R-MCL and MCL in terms of scalability. It is about 2 orders of magnitude faster than MCL and about one order of magnitude faster than R-MCL for most of the datasets. Second, in all cases both MLR-MCL and R-MCL report far fewer clusters than MCL. This trend again serves to highlight the fragmentation problem of MCL. Third, in terms of average normalized cut scores MLR-MCL dominates MCL and also usually outperforms R-MCL. On two datasets, namely Astro and Hep-Th, we find that R-MCL achieves a marginally better average N-Cut score.

4.3 Weak Scaling Behavior

In this experimental study we examine the weak scaling behavior of MLR-MCL and compare it with the state of

[2] Astro, HepPh, HepTh and Epinions were obtained from http://cs-www.cs.yale.edu/homes/mmahoney/NetworkData/

[3] Obtained from Andrew McCallum's web page: http://www.cs.umass.edu/ mccallum/code-data.html

[4] Obtained from the Database of Interacting Proteins: http://dip.doe-mbi.ucla.edu/dip/Main.cgi

[5] In [15], we describe a further experiment where we vary the inflation parameter for MCL and note that the trends do not significantly change.

Dataset	MLR-MCL				R-MCL				MCL			
	Clusters	N-Cut	Avg. N-Cut	Time	Clusters	N-Cut	Avg. N-Cut	Time	Clusters	N-Cut	Avg. N-Cut	Time
Hep-Ph	264	76.77	**0.29**	**0.91**	458	190.03	0.41	5.41	1464	827.31	0.56	85
Cora	670	238.19	**0.35**	**1.26**	880	367.62	0.41	3.58	2991	1888.6	0.63	82
Astro	411	153.43	0.37	**2.62**	343	124.40	**0.36**	8.84	1940	1301.5	0.67	515
Dblp	723	152.24	**0.21**	**0.51**	1750	575.19	0.32	1.57	1943	648.48	0.33	12.0
Epinions	1632	735.87	**0.45**	**26.86**	4025	1863	0.46	32.3	15663	10041	0.64	4383
Hep-Th	795	293	0.37	**0.37**	735	266.4	**0.36**	1.05	1655	855	0.51	12

Table 3: Comparison of MLR-MCL, R-MCL and MCL on 6 real datasets. The same inflation parameter value of $r = 2$ was used for all 3 methods. The times are in seconds. Best results are in bold.

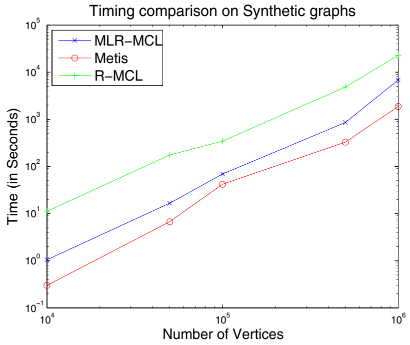

Figure 2: Weak scaling

the art Metis algorithm. This experiment evaluates the performance of these algorithms as the size and complexity of the dataset is increased. For this experiment we used the synthetic datasets described earlier, ranging from a 10,000 node graph to a 1 million node graph. From Figure 2, we observe that both MLR-MCL and Metis show good scalability trends. While MLR-MCL is competitive, Metis clearly demonstrates superior performance (by a factor of two typically) on this synthetic dataset. As a counterpoint we also present results on R-MCL which is about 2 orders of magnitude slower across the board than MLR-MCL. We do not include results for MCL here since it was found to be extremely slow even on the smaller datasets. Graclus also is not included here since the algorithm was found to suffer from severe memory thrashing effects when attempting to run on the the 500,000 vertex and 1 million vertex datasets.

4.4 Comparison with Graclus and Metis

In the next set of experiments we compare the qualitative performance of MLR-MCL with Graclus and Metis. We detail the results here on 6 real datasets. In all experiments MLR-MCL is run with the inflation parameter $r = 2.0$. For MLR-MCL we vary the coarseness of the clustering by varying the size of the coarsest graph; we subsequently run Graclus and Metis to output the same number of clusters as has been found by MLR-MCL. We plot the average normalized cut of each algorithm as a function of the number of clusters. (It must be kept in mind that with more number of clusters, seemingly small differences in average N-Cut can translate into significant differences in the total N-Cut.) From Figure 4a-f one can easily observe that MLR-MCL is either competitive with Graclus(DBLP and Hep-Ph) or better (Astro-Ph, Cora, Epinions and Hep-Th) in terms of the normalized cut objective. The improvement in Avg Ncut over Graclus for these latter four datasets is in the range of 10-15%, if we consider the median number of clusters, which is quite significant. Another obvious trend is that both these algorithms outperform Metis, often significantly. Drilling deeper into the data we find that this can be explained by the fact that both Graclus and MLR-MCL admit a more skewed (actually the skew for both is quite similar) clustering arrangement whereas Metis tends to force a more balanced partitioning. Another interesting observation is that when the number of clusters discovered is more, MLR-MCL typically performs better. A third point of note is that the two datasets where Graclus is competitive with MLR-MCL - Dblp and Hep-Th - are the two datasets with the lowest average degree of the 6 datasets (5.56 and 5.78 respectively).

Scalability Evaluation: In the next set of experiments we compare and contrast the scalability of MLR-MCL with Metis and Graclus on three of the real datasets, while varying the number of clusters. From Figure 5a for the Epinions dataset we find that MLR-MCL is competitive with Metis, and that both algorithms are faster than Graclus.The trends in Figures 5b and c are consistent in that Metis outperforms MLR-MCL, which in turn outperforms Graclus, especially with increasing number of clusters.

4.5 Clustering PPI networks: A Case Study

The goal of analyzing protein-protein interaction (PPI) networks is to extract groups of proteins that either take part in the same biological process (*induction of cell death* is an example) or perform similar molecular functions (e.g. *RNA binding*). This is a challenging problem; it is estimated that the protein function of about one-fourth of the proteins is unknown even for the most well-studied organisms such as yeast.[16].

We use as our dataset the PPI network of *S. cervisiae* or yeast, which contains 4741 proteins with 15148 known interactions. We perform a domain-specific evaluation using The Gene Ontology database [18], which provides three vocabularies (or annotations) of known associations – Molecular Function, Biological Process and Cellular Component. The first two have functional significance while the last one refers to the localization of proteins within a cell. Researchers have used this ontology in the past to validate the biological significance of clusters. Merely counting the number of proteins that share an annotation within each extracted cluster is misleading since the underlying frequency of the annotations is not uniform - more proteins are characterized by an annotation at the top of the hierarchy than at the bottom. For this reason, p-values are often used to calculate the statistical significance of such clusters [1]. Intuitively these values capture the probability of seeing a particular grouping, or better, by random chance using a background distribution (typically hyper-geometric). Let the total number of proteins be N with a total of M proteins sharing a particular annotation. The p-value of observing m or more proteins that share the same annotation in a cluster of n proteins, using the Hyper-geometric Distribution is:

$$p - value = \sum_{i=m}^{n} \frac{\binom{M}{i}\binom{N-M}{n-i}}{\binom{N}{n}}$$

Smaller p-values imply that the grouping is less likely to be random. [6]

It is worth remarking before we discuss the results that earlier research has shown that MCL typically outperforms MCODE and several other domain specific community discovery algorithms for such biological networks[2]. In our study we compared the performance of MCL, MLR-MCL and Graclus on this domain specific qualitative metric. Metis was found to perform poorly on this metric, primarily because of its tendency to favor balanced clusters.

We set the inflation parameter r to 1.6 for both MCL and MLR-MCL. The size of the coarsest graph was set to 1000 nodes for MLR-MCL. MLR-MCL returned 427 clusters where as MCL returned 1615 clusters. We then ran Graclus to output 427 clusters to keep the comparison fair. Each cluster is associated with the annotation that minimizes the p-value for that cluster, and the corresponding p-value was retained as the p-value to represent that cluster.

Figures 3a and b compare the p-values of the top 100 clusters returned by each algorithm, under the Biological Process (P) and Molecular Function (F) vocabularies respectively. The Y-axis represents negative log p-values while the X-axis is simply an ordered list of the top-scoring clusters produced by the different graph clustering algorithms. Since better clusters have lower p-values, higher values on the graph represent a higher quality of clustering. As we see from the charts, MLR-MCL clearly outperforms Graclus and MCL among the top set of clusters and is clearly competitive or better than either across the board for both the Molecular Function and Biological Process ontologies.

Under the evaluation using Biological Process annotations, the top-scoring cluster returned by MLR-MCL obtained a p-value of $1.8e-80$, which is significantly better than $2.4e-30$ and $1.6e-28$, the top scoring p-values for Graclus and MCL respectively. In fact MLR-MCL returns 8 clusters which score better p-values than the best p-value scored by Graclus. The p-value of $1.8e-80$ was scored by a cluster of 88 proteins returned by MLR-MCL, out of which 55 are proteins currently known to be involved in the process of *nuclear mRNA splicing via spliceosome*. It is very interesting that the top scoring cluster for MCL was also in fact matched with the same annotation, but MCL managed to retrieve only 25 of the proteins known to be involved in this process. This clearly illustrates that MLR-MCL overcomes the main qualitative limitation of MCL - fragmentation of output.

5. CONCLUSION

In this work we have presented Regularized MCL and Multi-Level Regularized MCL, two flow based algorithms for graph clustering. Results on several real and synthetic datasets highlight the utility of the approach when compared with MCL, Metis and Graclus, three state-of-the-art graph clustering algorithms. Specifically, we find that the new algorithms are 2-3 orders of magnitude faster than MCL, and improve significantly on the quality of the output clusters. Similarly we find that our approaches outperform Metis and Graclus in terms of quality and are competitive in terms of scalability.

As part of ongoing and future research, we will investigate the utility of this approach for analyzing the temporal evolution of networks. Another line of inquiry is to extend it to directed graphs and bipartite graphs. We also believe that this algorithm is amenable to effective parallelization and worth studying.

Acknowledgments: We thank the reviewers for their thoughtful comments and suggestions. This work is supported in part by the following grants: NSF CAREER IIS-0347662, RI-CNS-0403342, CCF-0702586 and IIS-0742999

6. REFERENCES

[1] V. Arnau, S. Mars, and I. Marin. Iterative Cluster analysis of protein interaction data. *Bioinformatics*, 21(3):364–78, 2005.

[2] S. Brohee and J. van Helden. Evaluation of clustering algorithms for protein-protein interaction networks. *BMC Bioinformatics*, 7, 2006.

[3] D. Chakrabarti and C. Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Comput. Surv.*, 38(1):2, 2006.

[4] I. S. Dhillon, Y. Guan, and B. Kulis. Weighted Graph Cuts without Eigenvectors A Multilevel Approach. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(11):1944–1957, 2007.

[5] S. V. Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, 2000.

[6] S. Dutt and W. Deng. Cluster-aware iterative improvement techniques for partitioning large VLSI circuits. *ACM Trans. Des. Autom. Electron. Syst.*, 7(1):91–121, 2002.

[7] C. Faloutsos, K. S. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *KDD '04* , pages 118–127, New York, NY, USA, 2004. ACM.

[8] R. Kannan, S. Vempala, and A. Veta. On clusterings-good, bad and spectral. In *FOCS '00*, page 367, Washington, DC, USA, 2000. IEEE Computer Society.

[9] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20, 1999.

[10] B. Kernighan and S. Lin. An Efficient Heuristic Procedure for partitioning graphs. *The Bell System Technical J.*, 49, 1970.

[11] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. *CoRR*, abs/0810.1355, 2008.

[12] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Statistical properties of community structure in large social and information networks. In *WWW '08*, pages 695–704, New York, NY, USA, 2008. ACM.

[13] L. Li, C. J. Stoeckert, and D. S. Roos. OrthoMCL: identification of ortholog groups for eukaryotic genomes. *Genome Res*, 13(9):2178–2189, September 2003.

[14] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113, Feb 2004.

[15] V. Satuluri and S. Parthasarathy. Scalable Graph Clustering Using Stochastic Flows: Applications to Community Discovery. Technical Report OSU-CISRC-4/09-TR10, The Ohio State University.

[16] R. Sharan, I. Ulitsky, and R. Shamir. Network-based prediction of protein function. *Molecular Systems Biology*, 3, 2007.

[17] J. Shi and J. Malik. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000.

[18] The Gene Ontology Consortium. Gene Ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[19] K. Tsuda. Propagating distributions on a hypergraph by dual information regularization. In *ICML*, pages 920–927, 2005.

[6] We used a publicly available package called GO-TermFinder for calculating the p-values. The url is http://search.cpan.org/dist/GO-TermFinder/

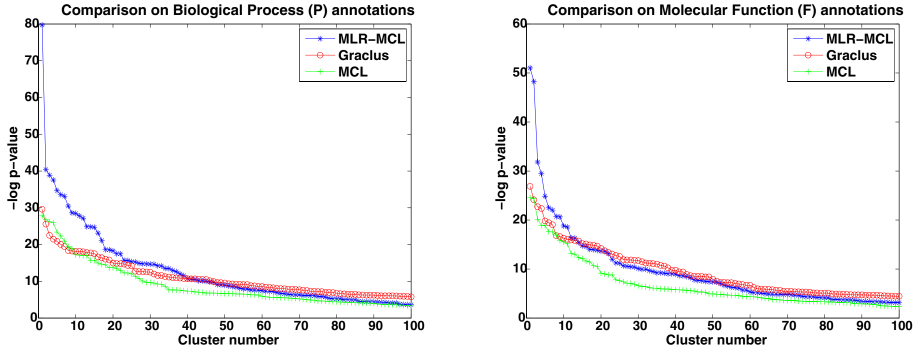

Figure 3: Comparison of (sorted) -log p-values on (a) Biological Process(P) annotations and (b) Molecular Function (F). Higher is better.

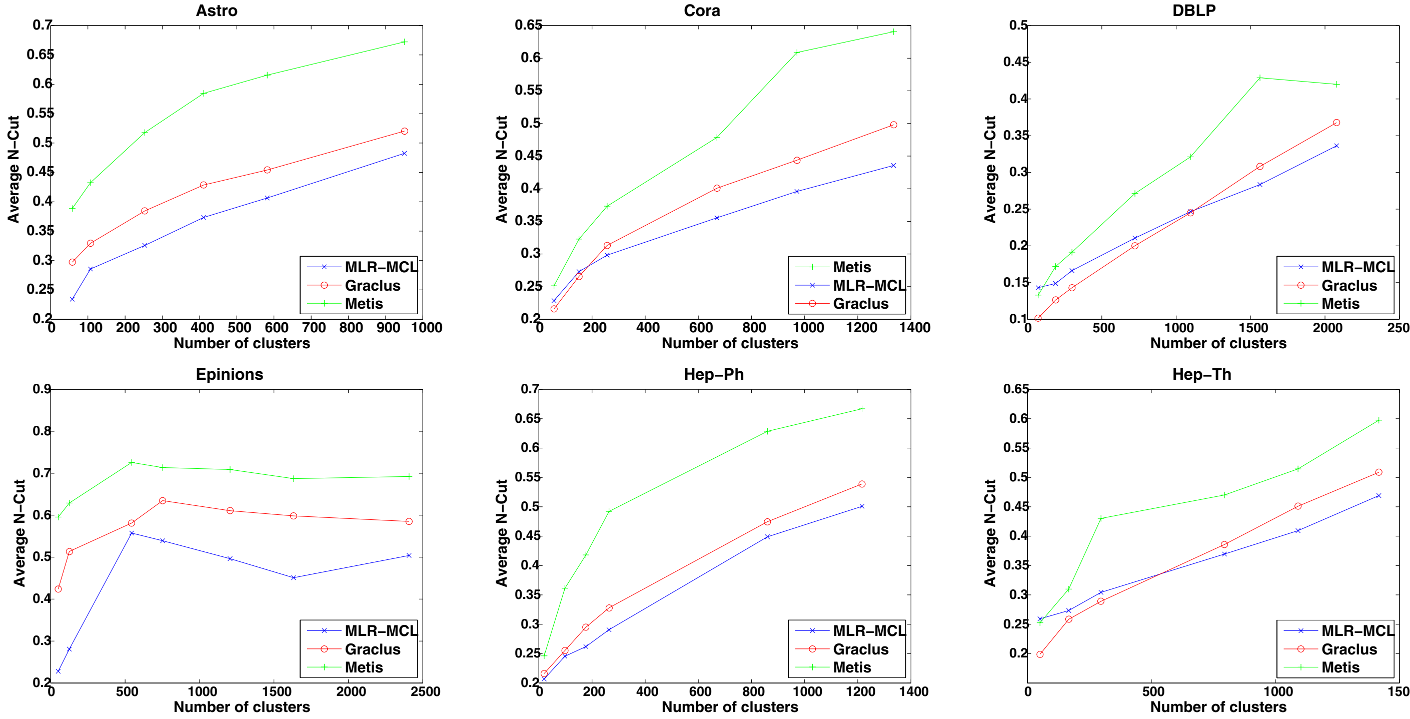

Figure 4: Comparison of Average Normalized Cut scores for varying number of clusters across 6 real world datasets. (a) Astro-Ph (b) Cora (c) Dblp (d) Epinions (e) Hep-Ph (f) Hep-Th. Lower is better.

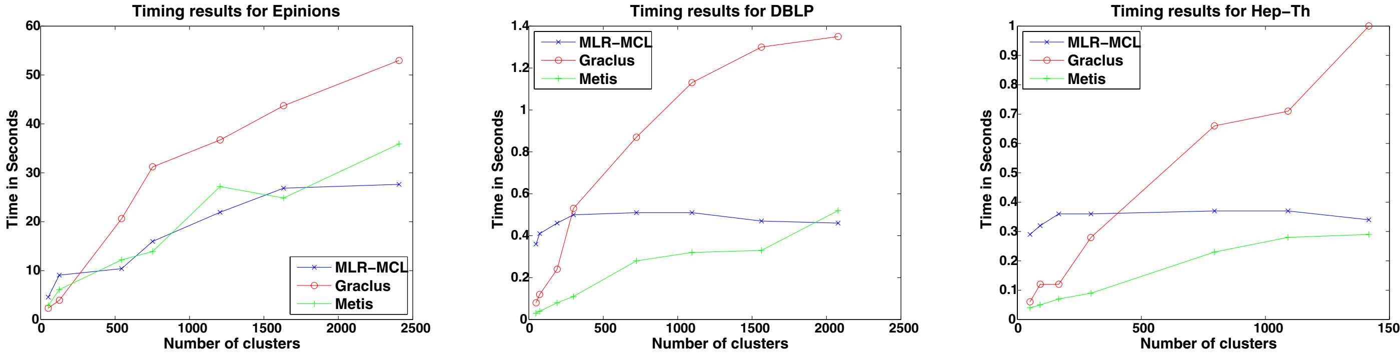

Figure 5: Comparison of timing for varying number of clusters across 3 real world datasets. (a) Epinions (b) Dblp (c) Hep-Th