# Finding Communities using Prior Knowledge

Karthik Subbian* Charu C. Aggarwal† Jaideep Srivastava‡ Philip S. Yu §

**Abstract**

The problem of community detection is a challenging one because of the presence of hubs and noisy links, which tend to create highly imbalanced graph clusters. Often, these resulting clusters are not very intuitive and difficult to interpret. With the growing availability of network information, there is a significant amount of prior knowledge available about the communities in social, communication and several other networks. These community labels may be noisy and very limited, though they do help in community detection. In this paper, we explore the use of such noisy labeled information for finding high quality communities. We will present an adaptive density-based clustering which allows flexible incorporation of prior knowledge in to the community detection process. We use a random walk framework to compute the node densities and the level of supervision regulates the node densities and the quality of resulting density based clusters. Our framework is general enough to produce both overlapping and non-overlapping clusters. We empirically show that even with a tiny amount of supervision, our approach can produce superior communities compared to popular baselines.

**Keywords:** Communities, Supervision, Clusters

## 1 Introduction

Network data shows unusually noisy behavior because of edges across unrelated nodes. The presence of hub nodes and noisy links can effectively thwart the clustering process, because these nodes tend to connect unrelated nodes in the network very strongly. As a result, one of the common observation in many community detection algorithms, is that nodes tend to form one large cluster through preferential attachment to hub nodes [2], and only very careful tuning of the algorithm is able to achieve balanced clusters. Even in such cases, the clusters which are found may not be very intuitive, and may still be dominated by noisy links.

In many scenarios involving very large networks, a substantial amount of prior knowledge may be available, which may reflect the application-specific knowledge about cluster membership. Some examples are as follows:

- In a scientific community network, it may be possible to label a small subset of nodes. While it is not practical to expect very large scale labeling, it is certainly possible to perform a small-scale labeling of the nodes.
- In a *heterogeneous* movie information network such as *IMDB* containing different node types such as movies and actors, some limited information about move genre may sometimes be available, though most actors or director nodes may not have such information.
- In a social network application, it may be desirable to cluster nodes based on their affinity to some products. While labels may not be known across all nodes, they may be available for (a very) small subset. For a target marketing application, goal-oriented clustering is superior to blind clustering.

The available labels may often be noisy, incomplete, and are often partially derived from unreliable data sources. Furthermore, in cases, where a user may wish to direct the clustering process with manual labeling, it is reasonable to expect only a very small subset of the network to have labels. Nevertheless, such sparse and incomplete information can still be useful in providing important hints that can direct the clustering process, so that the noisy unsupervised clusters created by preferential attachment to large hub nodes are avoided.

A related aspect of this issue has also been recently been observed in [2], which suggests that different regions of the network have a different level of local density. As a result, homogeneous clustering algorithms tend to create unusually large and incoherent clusters containing a significant percentage of the nodes from the network. When the local link density is very different across different network regions, the use of *global analysis* can either construct very small communities in sparse local regions, or report large and incoherent communities in dense regions. Therefore, it is important to use *local structural analysis* for determining the relevance of communities in a social network. While the work in [2] is able to achieve some of these goals, it is not designed for incorporating specific goals or prior knowledge in the clustering process. In this paper, we will design an effective approach to incorporate prior knowledge into the community detection process. We will present experimental results which test the effectiveness of our approach in different scenarios.

*University of Minnesota - Twin Cities, karthik@cs.umn.edu
†IBM T. J. Watson Research Center, charu@us.ibm.com
‡University of Minnesota - Twin Cities, srivasta@cs.umn.edu
§University of Illinois - Chicago, psyu@cs.uiuc.edu

**1.1 Related Work** The problem of unsupervised network clustering has been widely studied in the graph mining literature [2, 6, 7, 10]. Discussions of important statistical properties of web clusters are provided in [10]. Evolutionary characteristics of dynamic communities are studied in [5, 6]. The problem of clustering has also been studied in the context of combining node content in order to improve its effectiveness [12, 13]. While prior knowledge has been extensively used for clustering *multi-dimensional or text data* [3, 9, 14], the problem has remained largely unexplored for network data. Some methods exist for using pairwise constraints in clustering, though are largely inappropriate for the kind of supervision discussed in this paper [11]. In this paper, we will design a holistic and flexible framework in which the *a very tiny amount of prior knowledge can be used to greatly improve the clustering process*.

## 2 Density-Based Clustering Model

In this section, we will introduce the problem of supervised network clustering. We assume that we have an undirected network $G = (N, A)$, in which $N$ is the set of nodes, and $A$ is the set of edges. It is assumed that the number of nodes in $N$ is $n$. In many applications, the edges in the network may be associated with a weight, which indicates the strength of the relationship. We assume that the weight of edge $(i, j)$ is denoted by $w_{ij}$. For example, in an author-relationship network, the weights could represent the number of papers authored by a pair of individuals. In many network applications, the weight $w_{ij}$ is assumed to be 1, though we allow the use of a weight if needed for greater generality. A tiny subset $N_s \subseteq N$ of nodes are labeled. We assume that there are $l$ different labels denoted by $\{1 \ldots l\}$. All nodes in $N_s$ are labeled with one of the values drawn from $\{1 \ldots l\}$, whereas the nodes in $N - N_s$ are unlabeled. We would like to partition the nodes in $N$ into $k$ different sets of nodes $C_1 \ldots C_k$. In addition, we have a set of small subgraphs $\mathcal{O}$ which are referred to as the outlier set. For a given node $i$, we assume that the edges incident on it are denote by $I(i)$.

The overall idea of the supervised clustering approach is to design a density-based method in which clusters are defined in terms of *density-connected sets of nodes*. A density connected pair of nodes is one in which a path of nodes exists between the pair, such that each node has density above a pre-defined threshold. The concept of density-connectedness was defined in the context of multi-dimensional clustering algorithms by the DBSCAN method [8], and has been used quite successfully in conventional clustering methods on non-structural data. As we will see in this paper, the density-based method is also useful for structural clustering, and particularly so for the supervised case. On the other hand, the concept of density is much more challenging to define in the context of structural data.

How should density be defined for structural data? In the case of multi-dimensional data, pairwise distances between nodes can be clearly defined, which are useful in defining kernel-density estimators for the data points. While such pairwise distances can also be defined for structural data by using shortest-path distances, the complexity of such computations increase at least quadratically with the number of nodes in the network. This can be prohibitive for large-scale applications. A more intuitive way of understanding the clustering process is in the context of a page-rank style random-walk process in which a surfer traverses the different nodes in the network by randomly picking any neighbor of a node during the walk. The density of a node is essentially defined in an identical way to the page-rank computation.

DEFINITION 1. *The density of a node $i$ is defined as the steady-state probability that a random surfer on the network with a pre-defined set of reset probabilities visits node $i$ at any given transition.*

Intuitively, a random surfer on the network (upon entering a cluster), tends to get trapped in the cluster because the nodes in this dense region tend to have much higher visit probability than the surrounding nodes with lower visit probability. Therefore, a natural way of demarcating the boundaries of this cluster would be to exclude nodes from the cluster for which the density is below a given threshold, and then considering only connected regions of these high density nodes as candidates for a cluster.

Before discussing the clustering process in more detail, we will introduce the fundamentals of random-walk computation. In the random-walk process, at any given step, the random surfer either transitions to any node $j$ adjacent to $i$ with probability proportional to $p_{ij} = w_{ij} / \sum_{j \in I(i)} w_{ij}$, or it *resets* to a random node in the network with probability bias vector (or personalization vector) $\overline{\gamma}$. Thus, the conditional probability of transition to node $i$ (in case of a reset) is denoted by $\gamma(i)$. We will discuss the choice of bias vector $\overline{\gamma}$ slightly later, as it plays an important role in the supervision process. The relative probability of a reset is denoted by $\alpha$. Therefore, if we assume that the steady-state probability of node $i$ is denoted by $\pi(i)$, the random walk equation for node $i$ can be stated as follows:

$$\pi(i) = \sum_{j \in I(i)} p_{ji} \cdot \pi(j) \cdot (1 - \alpha) + \gamma(i) \cdot \alpha \tag{2.1}$$

We will use the values $\overline{\pi} = (\pi(1) \ldots \pi(n))$ as a proxy for the density of the node. In addition, since $\pi(i)$ represents a probability, we have $\sum_{i=1}^{n} \pi(i) = 1$.

We note that the above computation of density does not incorporate any class information into the estimation process. The use of the random walk framework for density estimation allows a seamless framework for controlling the level of supervision in the density estimation process. Specifically, this can be achieved by controlling the reset vector of the

random walk. In the unsupervised case, we assumed that a single random walk was performed, for which the value of the reset vector $\gamma(i)$ was equal to $1/n$ for each $i$. In this case, we perform $l$ different random walk computations in order to create $l$ different class-specific sets of densities.

Let $m_1 \ldots m_l$ be the number of nodes belonging to the $l$ different classes. Therefore, we have:

$$|N_s| = \sum_{i=1}^{l} m_i \tag{2.2}$$

The class-specific bias-vector $\eta_i$ for class $i$ contains a value of $1/m_i$ for nodes belonging to class $i$. Otherwise, the value of $\eta_i$ is set to 0. This class-specific bias vector can be used in order to determine the class-specific density $\overline{\pi^i}$. We note that the restart probability regulates the level of bias which is incorporated by the supervision process. If the restart probabilities are very low, and the (undirected) network is fully connected, then the steady state probabilities are very similar across the different nodes.

We note that the class-specific density $\overline{\pi^i}$ provides a different way to define the overall density of the clustering process. Specifically, we define the *dominant density* $\pi^*(j)$ of a node $j$ as its maximum density over all the different classes.

DEFINITION 2. (DOMINANT DENSITY) *The dominant density $\pi^*(j)$ of the node $j$ is defined as the maximum of $l$ different class-biased density values. In other words, we have:*

$$\pi^*(j) = max_i \pi^i(j) \tag{2.3}$$

We note that this definition of the dominant density will result in peaks of the value of $\pi^*(j)$ in regions of high density for specific classes. While it will still allow some mixing of nodes belonging to different classes, the clustering process will continue to be biased by the identity of the labels.

The partially supervised clustering process needs to combine the different class-specific densities effectively for the purposes of classification. For this purpose, we will use an approach which first determines the clusters separately for the different classes, and then combines them by using the structural behavior of the components found by the different classes. This will be described in detail in the next section.

The restart probability regulates the level of bias which is incorporated into the random walk vectors because of the supervision process. Thus, the supervision is still "partial" because the personalization vectors only bias the densities, whereas the actual cluster creation algorithm will allow the mixing of nodes belonging to different classes. The mixing of different node labels in the same cluster is likely to occur in cases, where they are placed in the same structural locality. Such partial supervision is especially useful, when the node labels may be incorrect, noisy, or may incompletely represent the classes in the data. In some cases, correlated (but different) node labels may belong to the same structural locality, and therefore *ought to be* placed in the same cluster in a partially supervised algorithm. In such cases, the density bias provides good hints for cluster localization, but is not an absolute guideline, since structural locality is allowed to trump class membership.

## 3 Community Detection with Prior Knowledge

The density-based method in the last section can be used to encode significant prior knowledge about the underlying clusters, even when a small percentage of the labels are specified. Therefore, this can be used in order to meaningfully define clusters in a biased way, when the class-specific densities are used for the analytical process. We define the concept of a *density-connected cluster* with respect to the density vector $\overline{\pi}$. The concept of density-connected clusters is analogous to this concept [8] in the case of multi-dimensional data, except that we generalize it here to the case of structural data. We will first define the density-connected clusters with respect to general class probability vectors $\overline{\pi}$. Later we will see how supervision can be used in order to further improve the quality of the classification.

DEFINITION 3. (DENSITY CONNECTED PAIR) *Two nodes $i$ and $j$ are said to be density connected at level $\delta$ with respect to the density vector $\overline{\pi}$, if the following conditions are satisfied:*

- *$\pi(i) \geq \delta$ and $\pi(j) \geq \delta$*
- *There exists a path in the network $G$ denoted by $iq_1q_2 \ldots q_rj$, such that for each node $q_m$ on the path, we have $\pi(q_m) \geq \delta$*

We can generalize the concept of a density-connected pair of nodes to that of a density-connected set.

DEFINITION 4. (DENSITY CONNECTED SET) *A set of nodes $S$ in $G$ are density connected at level $\delta$ with respect to $\overline{\pi}$, if for every pair $i, j \in S$, the nodes $i$ and $j$ are density connected at level $\delta$ with respect to $\overline{\pi}$.*

Therefore, our goal is to determine density connected sets in the network in order to determine the underlying clusters. One challenge with using the density connected approach directly is that in typical networks, the clusters are of *widely varying sizes*, and correspondingly, the densities at the different nodes may vary by *orders of magnitude*. This can be easily noticed in page rank computations [4] on the web, in which the underlying page rank values of different nodes may vary wildly, and yet good clusters can be found at all these different levels of densities. Thus, by setting the density threshold $\delta$ too high, we may lose many relevant smaller

clusters in the sparser regions, whereas picking the density threshold low enough to capture such clusters may fabricate false clusters in denser regions.

We note that the aforementioned problem arises as a direct result of the heterogeneity [2] of networks in which different local regions of the data have different levels of density. Therefore, the use of a global analysis for the clustering process is likely to result in clusters which do not reflect the underlying trends in the data well. In order to handle the complications associated with this heterogeneity, we use a *residual network* approach in which we determine only the most significant network clusters in a given iteration for a given class. Then, we work with a residual network which does not include the excessively dense regions which have already been determined to be clusters. We then *iteratively* create the next most significant clusters, and keep removing nodes from the residual network, until it becomes "sufficiently" disconnected with only small components. The significant clusters discovered in these iterations are the *prototype clusters*, with which the remaining smaller connected components are either consolidated or declared outliers. We will first define the concept of a $\delta$-residual network formally.

DEFINITION 5. ($\delta$-RESIDUAL NETWORK) *The $\delta$-residual network of $G$ with respect to the density vector $\overline{\pi}$ is denoted by $G^{\delta}$, and is defined by removing all the nodes from $G$ for which the density is greater than $\delta$.*

We note that $G^1$ is equal to the original network $G$, because the density values are probabilities, and therefore all nodes have density values at most 1. The smaller the value of $\delta$, the more disconnected the residual network $G^{\delta}$ will be. Our density-based algorithm picks successively smaller values of $\delta$ in order to pick out more density-connected prototype clusters from the network in succession. The choice of $\delta$ is based on the statistical mean and variances of the densities of the different nodes, and an additional parameter $\beta$, which is known as the *slice factor*. The overall process of the algorithm uses four main phases:

- **Prototype Creation:** In this phase, prototype clusters are created separately for the different labels. This is achieved by slicing the label-specific density-based network repeatedly. We further note that most of the nodes are *not* included in the clusters for a particular class.
- **Prototype to Orphan Consolidation:** In this phase, the prototypes from different labels are consolidated together. Furthermore, the orphans are consolidated with the clusters to which they are best connected. At the end of this stage, some nodes may be missing from any cluster and other nodes may be present in multiple clusters.

**Algorithm** *FindDensityClusters*(*Graph:* $G$
*Min. Cluster Size:* $min\_thresh$, *SliceFactor:* $\beta$ );
**begin**
Determine density values $\overline{\pi_j}$ with
for the different classes $j$ with
the random-walk equations;
**for** each class $j$ **do**
$\mathcal{P}_j = \mathcal{R} = \{\}$; { Initialize the
prototype and orphan clusters };
{ **Phase I begins: Prototype Cluster Creation** }
**for** each class $r$ **do**
**begin**
$\delta_0 = 1$; $G'^{\delta_0} = G$; { Initialization }
$k = 0$; { Iteration Number }
**repeat**
$k = k + 1$;
Compute $\mu_k$ and $\sigma_k$ on $G'^{\delta_{k-1}}$;
$\delta_k = \mu_k + \beta \cdot \sigma_k$;
Determine all the density connected components $\mathcal{S}$ at
level at least $\delta_k$ in $G'^{\delta_{k-1}}$;
Add all connected components in $\mathcal{S}$ with at least
$min\_thresh$ nodes to $\mathcal{P}_r$ and the remaining to $\mathcal{R}_r$;
Compute residual network $G^{\delta_k}$;
Add any components in $G^{\delta_k}$ with less than
$min\_thresh$ nodes to $\mathcal{R}_r$ and let the
remaining network be $G'^{\delta_k}$;
**until**($\mathcal{P}_r$ did not increase in last iteration);
Remove any components from $\mathcal{R}_r$ for which
avg. class-specific density is less than unsup. density;
Remove any components from $\mathcal{P}_r$ for which
avg. class-specific density is less than unsup. density;
**end**
{ **Phase II: Prototype to Orphan Consolidation** } ;
**for** each class $r$ **do**
**begin**
**repeat**
Determine all components in $\mathcal{R}_r$ for which a
maximum number of outgoing links are connected
to one component in $\mathcal{P}_r$;
Consolidate the prototypes in $\mathcal{P}_r$ with their
matching components in $\mathcal{R}_r$, and remove these
components from $\mathcal{R}_r$;
**until**($\mathcal{R}_r$ did not reduce in last iteration);
$\mathcal{O} = \mathcal{O} \cup \mathcal{R}_r$;
**end**
{ **Phase 3: Inserting Missing nodes** }
Insert all missing nodes $\mathcal{O}$ globally over all
network components for the different classes
using a similar approach to Phase 2,
except that prototype set is the union over
all classes, and orphans are all individual
missing nodes;
{ **Phase 4** } **Final node re-assignment** }
Perform final hard or soft assignment of nodes
based on class-specific probabilities;
**end**

Figure 1: Finding Density-based Clusters

- **Missing Node Insertion:** In this phase, nodes which are missing from any cluster are re-inserted into their best matching cluster based on the density connectedness in the network. At this stage, no nodes are missing from any cluster, but some nodes may be present in many clusters.

- **Final Node Re-Assignment:** While overlapping clusters are quite common in network scenarios, we use a flexible approach in which nodes are either allowed to belong to a particular cluster with a given soft probability, or they belong to exactly one cluster.

Our algorithm, illustrated in Figure 1, is referred to as *CODEK*, which corresponds to COmmunity DEtection using prior Knowledge. The input to the algorithm is the network $G$, and a minimum threshold $min\_thresh$ on the component size of the prototype clusters. In addition, a *slice factor* $\beta$ is used, which regulates how aggressively the network is sliced based on the density distribution. As we will see later, this factor regulates the number of clusters found from the network.

The first phase of the algorithm partitions all the nodes into sets of *prototype clusters* separately for each class $r$. The set of prototype clusters for class $r$ are denoted by $\mathcal{P}_r$, (which is a set of cluster sets) or a set of *orphan clusters* denoted by $\mathcal{R}_r$. In the second phase of the algorithm, each of the orphan clusters for class $r$ will be assigned to either one of the prototype clusters or to the outlier set $\mathcal{O}$. Therefore, the first phase is referred to as the *prototype phase*, while the second phase is referred to as the *consolidation phase*.

The algorithm starts off by computing $\overline{\pi}_r$ for the network $G$ with the use of the random walk computation specific to the class $r$. The prototype set $\mathcal{P}_r$ and orphan set $\mathcal{R}_r$ are both initially set to null for the class $r$. Let $\mu_1$ and $\sigma_1$ be the mean and standard deviation of the initial density vector $\overline{\pi_r}$, which is specific to the class $r$. We use this to set the initial density threshold $\delta_1 = \mu_1 + \beta \cdot \sigma_1$. All the density connected nodes in the graph $G$ are removed, and each connected component is added to the prototype set $\mathcal{P}_r$, if it contains more than $min\_thresh$ nodes. Otherwise, it is added to the orphan set $\mathcal{R}_r$. For the next iteration, we reconstruct $G^{\delta_1}$ by removing all the density connected sets which were added to $\mathcal{P}_r$ and $\mathcal{R}_r$ from the network together with their incident edges. The removal of such nodes may also result in the further disconnection of the network into smaller components. If some of these components have less than $min\_thresh$ points, then they are added to the orphan set as well. Therefore, the network $G^{\delta_1}$ is adjusted to remove these nodes to create an even smaller network $G'^{\delta}_1$. In the next iteration, we compute $\mu_2$ and $\sigma_2$ in the (adjusted) $\delta_1$-residual network $G'^{\delta_1}$. The new threshold is then set to $\delta_2 = \mu_2 + r \cdot \sigma_2$. We note that since the high density nodes in the network have already been removed, we will have $\delta_2 < \delta_1$. Thus, we may find new nodes which were not $\delta_1$-density connected in $G$, but may be $\delta_2$-density connected in $G'^{\delta_1}$. As before, we remove the $\delta_2$-density connected components from $G^{\delta_1}$, and add them either to the prototype set or the orphan set. This process is repeated in order to successively construct $G'^{\delta_k}$, which continues to become smaller and more disconnected. This process is repeated until in a given iteration the remaining network is either the null set, or no node in $G'^{\delta_k}$ has density greater than $\delta_{k+1} = \mu_{k+1} + \beta \cdot \sigma_{k+1}$. The residual network is discarded. In many cases, this residual network may contain a majority of the nodes in the network which are not relevant to that particular class. Thus, each set of the set of prototype clusters $\mathcal{P}_r$ is biased towards the class $r$.

The first phase of the algorithm thus creates a set of prototype clusters $\mathcal{P}_r$ and orphan clusters $\mathcal{R}_r$ for the class $r$. The second phase of the algorithm builds around these prototype clusters by adding orphan clusters to them. For each orphan cluster, we determine the prototype cluster to which it has the maximum number of connecting links. After scanning all the orphan clusters for linkage with prototype clusters, we merge the assigned orphaned cluster with their respective prototype clusters (and also add the connecting links) in order to increase the size of the prototype clusters. At the same time, the assigned orphaned clusters are removed from the set $\mathcal{R}_r$. We note that the increase of the size of the prototype clusters in $\mathcal{P}_r$ may result in some new components in $\mathcal{R}_r$ becoming significantly connected to the (expanded) clusters in $\mathcal{P}_r$. Therefore, this entire process is repeated iteratively, until either no orphan cluster is assigned in an iteration, or the set $\mathcal{R}_r$ becomes empty.

At this stage, many nodes are included in multiple clusters, and other nodes are not included in any cluster. In fact, at the end of the first phase, it is possible that significant parts of the network may not be included in any cluster at all. However, since the class-based prototypes provide excellent reference points to build the clusters around, it is possible to insert the remaining nodes *in reference to* these high quality prototypes. Even when most of the nodes are not included in any prototype, this approach is still able to find better clusters for such nodes than a purely unsupervised method which uses no prior information at all. As in the case of the second phase, an iterative approach is used in which the data points are repeatedly inserted in clusters, based on the cluster to which they have the highest connectivity. Such an approach may sometimes merge multiple prototype clusters (possibly generated from different classes), when they are connected to such clusters. This is actually quite reasonable, because cluster overlaps may exist among the different classes. Such overlaps need to be properly accounted for in the final cluster creation process.

After the third phase, many nodes may belong to multiple clusters. Such overlaps among different clusters are quite common in many real networks. On the other hand,

in many applications, it may be desirable to perform hard-partitioning, in which each node belongs to only a single cluster. Therefore, we use two variations in order to perform the final assignment of data points to clusters:

- **Soft Partitioning:** For nodes which belong to multiple clusters, each occurrence of a node in a particular cluster is associated with a density value. This density value was the random walk probability of that node at the time it was separated out from the network as a prototype or orphan cluster. The probability of a node belonging to that cluster is estimated as the fractional random walk probability for that instance. We refer to this variant of our algorithm as CODEK-S.

- **Hard Partitioning:** In this case, each node is assigned to its best matching cluster. The best matching cluster is picked by using the highest density for that node. This version of the algorithm is called CODEK-H.

The resulting hard or soft partitioning is reported as the final result found by the algorithm.

## 4 Experimental Results

In this section, we will present a number of experimental results illustrating the effectiveness of the proposed technique. We will study both the soft and hard partitioning approaches compared to several well established baselines.

**4.1 Data Sets** We tested our approach on the *DBLP* and *IMDB* data sets.

**DBLP Data Set:** We downloaded the *DBLP* XML file from [19], on January 23, 2012. We extracted the title, author and venue information for several published document types, including, articles, proceedings, inproceedings, and thesis. We further cleaned the *DBLP* data set to remove documents with missing meta-information such as author, date, and conference proceedings name. After cleaning, the *DBLP* dataset contained 1,008,883 distinct authors and 1,810,117 documents. We constructed a *DBLP* co-authorship network using this data, which contained 1,008,883 nodes and 3,383,570 edges. An edge $(i, j)$ in this relationship represents co-authorship between authors $i$ and $j$. The average degree per node was 1.86.

**IMDB Data Set:** The *IMDB* data set is an international collection of movies, documentary and short films. It contains actors, actresses, crew, plot and many more details of each movie. The data set was downloaded from [18]. The television serials were excluded from our data set, and we focused on the remaining 37% of movies, which translated to 840,542 movies. A co-actor network was constructed in an analogous way to *DBLP*. To reduce the effect of uninformative nodes and edges, we choose only actors who had acted in at least 2 movies and edges which correspond to at least 2 co-acting relations. The size of the resulting network was, 423,281 nodes with 3,625,196 edges. The average node degree was 8.56.

Each of these data sets had class labels, which were used for supervision. For *DBLP*, we extracted the list of top 10 conferences from the 22 different areas of computer science as grouped in $academic.research.microsoft.com$. We counted the number of conference publication for each author in each of these domains and picked the maximum publication domain as the ground truth community for the author. Similarly for *IMDB*, we counted the number of movies acted by each actor in top 3 genres: *Short, Drama, and Documentary*. As the movie participated in several genres, unlike a publication in *DBLP*, we split the credit equally to all the participating genres for that movie. The actor is then assigned a genre as the community label based on *dominant* behavior.

**4.2 Evaluation Measures** We tested our algorithm for clustering effectiveness using the average cluster purity measure. We also measure the distribution of cluster sizes and average cluster entropy to measure the quality of the communities obtained.

We measure the cluster purity of a cluster as the fraction of the nodes belonging to the dominant label. This is averaged in a weighted way over different clusters in order to create a composite cluster purity measure.

A disadvantage of this measure is that it depends only on the *dominant* label of the cluster, and ignores the distribution of non-dominant labels. The cluster entropy for cluster $i$ is measured as follows:

$$h_i = 1 - \sum_{l=1}^{L} p_{il}^2 \tag{4.4}$$

Here, $L$ is the total number of ground truth communities, and $p_{il}$ is the fraction of class labels of community $l$ in cluster $i$. The cluster entropy is then averaged over the different clusters. The cluster purity effectiveness measure can be computed in a comparable way for both the hard and soft versions of the algorithm, by appropriately assigning portions of "shared credit" for each data point to the appropriate clusters, when a data point belongs to multiple clusters.

As mentioned earlier, the extreme unevenness of cluster sizes is a concern for many algorithms. While some methods such as *METIS* are able to force this via hierarchical partitioning, this continues to be an issue for many of the algorithms which use flat partitioning. Therefore, we measured the variance in cluster sizes as follows:

$$V = \sum_{i=1}^{K} f_i^2/K - 1/K^2 \tag{4.5}$$

Here, $f_i$ is the ratio of size of cluster $i$ to the number of nodes in the network, and $K$ is the number of clusters. Note

that Equations. 4.5 and 4.4 are applicable only for hard partitioning algorithms.

**4.3 Baselines** We use three well-established graph clustering baselines of different types to compare against our algorithm. They are *METIS* [15], *GRACLUS* [16] and *SSC*[17].

- **METIS:** This is a multi-level $k$-way graph partitioning algorithm. We used the implementation available from [15]. We specify the number of partitions of the graph as the input to the program.
- **GRACLUS:** This is a fast kernel-based multi-level graph clustering algorithm. We used the code from [16]. All parameters were set to their default values, except the number of clusters.
- **SSC:** This is a fast Sparse Spectral Clustering [17] (SSC) approach proposed for graph clustering. We used the *SSC* algorithm implementation available in [17]. The parameter $\sigma$ was set to $0.1$.

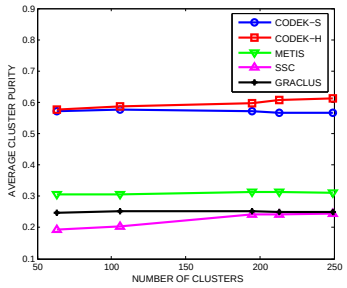
**4.4 Effectiveness results** We present the effectiveness results for *DBLP* and *IMDB* in terms of the average cluster purity and cluster entropy measured against various cluster sizes. As discussed earlier, our algorithm *CODEK* has two variants, soft (*CODEK-S*) and hard (*CODEK-H*) partitioning. We set the random walk restart probability to $0.1$ and stopping criteria threshold to be $10^{-4}$ and $10^{-6}$ for *DBLP* and *IMDB* data sets respectively. The parameters $min\_thresh$ is set to $50$ and number of exposed labels are $1375$ and $4200$ for *DBLP* and *IMDB* respectively for any bias class. Note that this set of labels is *very small* compared to the size of the entire network (about $2.9\%$). The number of clusters was controlled by using the slice factor parameter, as discussed in sensitivity analysis section. For other algorithms, the same number of clusters were used as an input parameter, in order to obtain results which could be reasonably compared with one another.

Figures 2(a) and 2(b) show the effectiveness of the proposed approach *CODEK-H* and *CODEK-S* using the average cluster purity. In both *DBLP* and *IMDB* data sets, both the soft and hard versions of the proposed approach consistently outperform the baselines. In *DBLP*, the average cluster purity of *CODEK-H* is at least twice that of the baselines. The soft version *CODEK-S* performs especially well for large cluster sizes. Note that, the baseline methods do not perform consistently in the same order in both data sets. For example, in *DBLP*, *METIS* performs better, whereas in *IMDB*, *SSC* performs better than *METIS*. However, we see that our proposed approach consistently performs better because of its use of local density connected clusters. This robustness is significant; if the performance of a clustering algorithm is erratic over different data sets, it cannot be trusted to perform effectively in different scenarios. We have also presented the results with the use of our other measure corresponding to cluster entropy. The results for *DBLP* and *IMDB* are shown in Figures 2(c) and 2(d) respectively. One can see that the average entropy of our method was better than all the baselines in *DBLP* and much better in reasonably large clusters in *IMDB*.

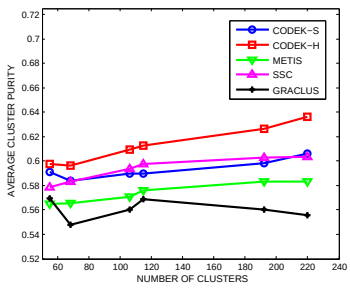
One of the challenges of community detection algorithms is their tendency to have uneven cluster sizes. While some algorithms such as *METIS* avoid this with a hierarchical approach of always dividing the largest cluster, this comes at the expense of purity. In order to examine how balanced the different clusters are in different scenarios, we compared the cluster size distribution by plotting the top-10 cluster sizes ranked in descending order of size. The 10-th cluster is replaced by an average of the cluster sizes over the remaining clusters (excluding the top nine). These cluster size distributions are shown in Figures 2(e) and 2(f) for both the *DBLP* and *IMDB* data sets. One can see that *METIS* produces similar size clusters, as it is by the design of the algorithm. The *GRACLUS* and *SSC* algorithms as noted earlier produces highly uneven size clusters. Furthermore, these algorithms behave in an unpredictable way over different data sets. In *IMDB*, *GRACLUS* performs slightly better than in the case of the *DBLP* data set. We also plot the overall variance in cluster sizes, over different granularity in partitioning. This summarizes the entire cluster sizes rather than the top 10, and is shown in Figures 2(g) and 2(h). In both *DBLP* and *IMDB*, our algorithm closely competes with METIS (which produces even clusters by design), whereas the other algorithms behave poorly on at least one of the data sets.

**4.5 Sensitivity analysis** As our algorithm has a few important parameters the controls the size of clusters, the number of clusters and the amount of supervision, we perform sensitivity analysis of these parameters in this section. In Figures 2(i) and 2(j), we show the sensitivity of changing the slice factor and the effect in the number of clusters generated by *CODEK*. In general, as the slice factor increases, the number of clusters generated in the prototype clustering phase decreases. Hence, the resulting number of clusters is also lower. We also vary the $min\_thresh$ factor to show the effect of this parameter on the number of clusters. As the $min\_thresh$ parameter is lowered the number of clusters that are density connected with in $min\_thresh$ factor increases, resulting in a larger number of clusters. This rate of decrease in the size of clusters for increasing slice factor is different for different data sets, depending the sparsity of the graph. For *DBLP*, the graph is much more sparse compared to *IMDB* and hence the slope of the curves are much more steeper.

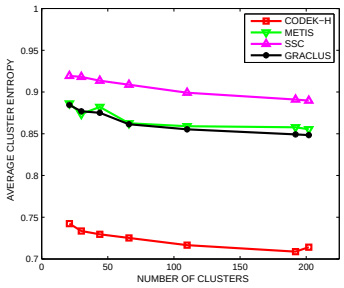
The effect of supervision level on the average cluster purity for the two data sets is illustrated in Figures 2(k) and 2(l). In each case, the slice factor was chosen so as



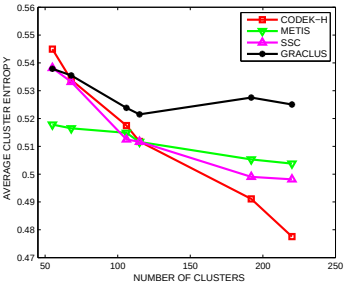

(a) DBLP Avg. Cluster Purity



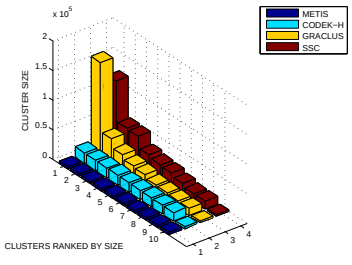

(b) IMDB Avg. Cluster Purity



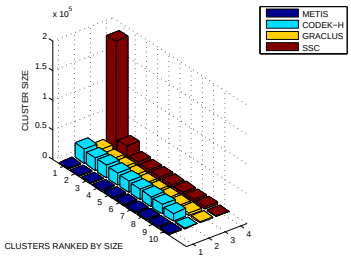

(c) DBLP Avg. Cluster Entropy



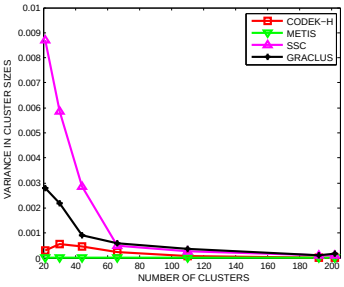

(d) IMDB Avg. Cluster Entropy



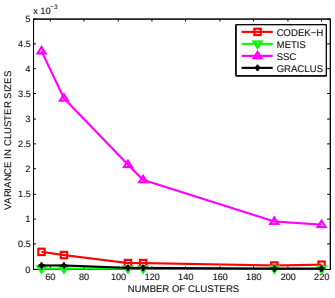

(e) DBLP Cluster Size Distribution



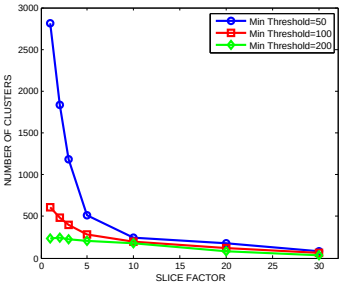

(f) IMDB Cluster Size Distribution



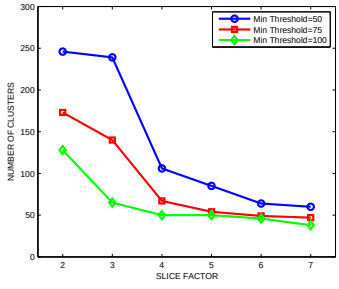

(g) DBLP Cluster Size Variance



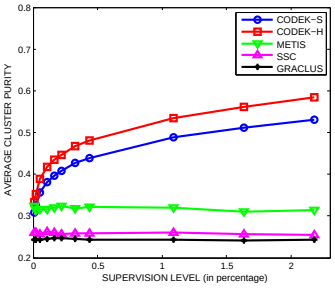

(h) IMDB Cluster Size Variance



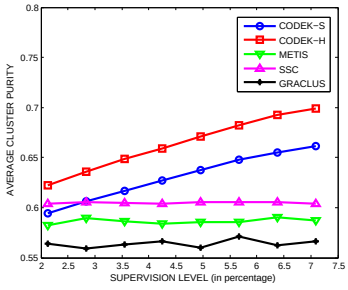

(i) DBLP Slice Factor Analysis




(j) IMDB Slice Factor Analysis




(k) DBLP Supervision Sensitivity




(l) IMDB Supervision Sensitivity

Figure 2: Experimental Results for *DBLP* and *IMDB* data sets

| Method | DBLP | IMDB |
|---|---|---|
| *CODEK-S* | 3.610 | 2.561 |
| *CODEK-H* | 3.642 | 2.574 |
| *METIS* | 0.126 | 0.085 |
| *GRACLUS* | 0.862 | 0.938 |
| *SSC* | 35.992 | 7.694 |

Table 1: Running time reported for baselines and *CODEK*

to hold the number of clusters to about 250, though this cannot be exactly controlled. *We note that as much as 2% of supervision is sufficient for our algorithm to perform significantly better than baseline methods. This suggests that a very small amount of prior information can have drastic results.* The behavior of *CODEK-H* and *CODEK-S* algorithms are very similar in both data sets to the level of supervision. As the supervision increases the algorithm performs better in terms of the average cluster purity. As the baseline algorithms are not supervised, the corresponding cluster purity is almost horizontal, but with some variations because of varying number of clusters (which was chosen to be the same as the *CODEK* algorithm).

**4.6 Efficiency results** The efficiency of our algorithm is comparable to that of the baseline algorithms. In Table 1, we compare the runtime for our algorithm versus baselines for both data sets. We present the run-time for both soft and hard partitioning of our algorithm. In terms of run time, *METIS* performs the best. Our algorithm is highly comparable to *GRACLUS*. The *SSC* method is an expensive eigenvector-based method. Hence, the algorithm is extremely slow for large cluster sizes. The run time numbers in Table 1 are reported in minutes for a cluster size of 202 and 220 for *DBLP* and *IMDB* respectively. The hard partitioning algorithm requires a small amount of extra time in the final phase because of the process of determining assignments of nodes to clusters.

## 5 Conclusions and Summary

Many network clustering algorithms frequently create imbalanced clusters of very low quality. This paper proposes a method for incorporating prior knowledge in network clustering applications. The results show that even a tiny amount of prior knowledge provides significant information for clustering. The resulting clustering is balanced, robust, and generally of much higher quality than state-of-the-art community detection methods in the literature. At the same time, it is able to retain reasonable efficiency.


## Acknowledgements

Research was sponsored by the Army Research Laboratory cooperative agreement number W911NF-09-2-0053 and Defense Advanced Research Project Agency (DARPA) agreement number W911NF-12-C-0028. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory, DARPA or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.


## References


[1] C. Aggarwal, H. Wang, *Managing and Mining Graph Data*, Springer, (2010).

[2] C. C. Aggarwal, Y. Xie, P. Yu. *On Community detection in locally heterogeneous networks*, SDM Conference, (2011).

[3] S. Basu, M. Bilenko, R. J. Mooney. *A probabilistic framework for semi-supervised clustering.* KDD, (2004).

[4] S. Brin, L. Page. *The anatomy of a large-scale hypertextual search engine.* WWW, (1998).

[5] D. Chakrabarti, R. Kumar, A. Tomkins. *Evolutionary clustering.* KDD, (2006).

[6] Y. Chi, X. Song, D. Zhou, K. Hino, B. L. Tseng. *Evolutionary spectral clustering by incorporating temporal smoothness.* KDD, (2007).

[7] A. Clauset, M. Newman, C. Moore. *Finding community structure in very large networks.* Phys. Rev. E 70, (2004).

[8] M. Ester, H.-P. Kriegel, J. Sander, X. Xu. *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise.* KDD, (2006).

[9] X. Ji, W. Xu. *Document Clustering with Prior Knowledge.* SIGIR, (2006).

[10] J. Leskovec, K. J. Lang, A. Dasgupta, M. W. Mahoney. *Statistical properties of community structure in large social and information networks.* WWW, (2008).

[11] X. Wang, I Davidson. *Flexible Constrained Spectral Clustering*, KDD (2010).

[12] T. Yang, R. Jin, Y. Chi, S. Zhu. *Combining link and content for community detection: a discriminative approach.* KDD, (2009).

[13] Y. Zhou, H. Cheng, J. X. Yu. *Graph clustering based on structural/attribute similarities.* VLDB, (2009).

[14] X. Zhu, Z. Ghahramani, J. D. Lafferty. *Semi-supervised learning using gaussian fields and harmonic functions.* ICML, (2003).

[15] G. Karypis, V. Kumar. *Multilevel k-way Partitioning Scheme for Irregular Graphs.* JPDC, 48(1), (1998), pp. 96–129. `http://glaros.dtc.umn.edu/gkhome/metis/metis/download`

[16] I. Dhillon, Y. Guan, B. Kulis. *Weighted Graph Cuts without Eigenvectors: A Multi-level Approach.* IEEE TPAMI, 29(11), (2007), pp. 1944–1957. `http://www.cs.utexas.edu/users/dml/Software/graclus.html`

[17] W-Y. Chen, Y. Song, H. Bai, C-J. Lin, E. Chang. *Parallel Spectral Clustering in Distributed Systems.* IEEE TPAMI, 33(3), (2011), pp. 568–586. `http://sourceforge.net/projects/spectralcluster/files/`

[18] `ftp://ftp.fu-berlin.de/pub/misc/movies/database/`

[19] `http://dblp.uni-trier.de/xml/`