Community Detection on Large Complex Attribute Network

Chen Zhe
Innovation Lab, PayPal
Singapore
zchen1@paypal.com

Aixin Sun
Nanyang Technological University
Singapore
axsun@ntu.edu.sg

Xiaokui Xiao
National University of Singapore
Singapore
xkxiao@nus.edu.sg

ABSTRACT

A large payment network contains millions of merchants and billions of transactions, and the merchants are described in a large number of attributes with incomplete values. Understanding its community structures is crucial to ensure its sustainable and long lasting. Knowing a merchant's community is also important from many applications - risk management, compliance, legal and marketing. To detect communities, an algorithm has to take advances from both attribute and topological information. Further, the method has to be able to handle incomplete and complex attributes. In this paper, we propose a framework named AGGMMR to effectively address the challenges come from scalability, mixed attributes, and incomplete value. We evaluate our proposed framework on four benchmark datasets against five strong baselines. More importantly, we provide a case study of running AGGMMR on a large network from PayPal which contains 100*million* merchants with 1.5*billion* transactions. The results demonstrate AGGMMR's effectiveness and practicability.

ACM Reference Format:
Chen Zhe, Aixin Sun, and Xiaokui Xiao. 2019. Community Detection on Large Complex Attribute Network. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3292500.3330721

1 INTRODUCTION

Network data in industry can be very complex with a large number of vertices of different types. Further, vertices may be associated with different types of attributes (*e.g.,* numerical and categorical) with missing values. We use PayPal payment network as an example. A vertex represents a merchant or consumer while an edge represents their transaction relationships. There are attributes associated with merchants describing their business but not consumers. Understanding community structures in such a network benefit many applications such as risk management, compliance management, and targeted marketing. However, finding communities in such an attributed network is not an easy task. Most state-of-the-art algorithms only consider either attributes or network topological information, but not both types of information. On the other hand, the optimal balance of using both types of information is unknown because community detection is unsupervised in nature. To make the task even more challenging, vertex attributes may provide contradictory information with topological structure [28].

In this paper, we propose a community detection framework named **AGGMMR** (Augmented Graph with Modularity Maximisation and Refinement) to detect communities from large-scale attributed graph. Our framework is designed to partition an attributed graph based on both its attributes and topological information through a greedy modularity maximisation model. It consists of three phases: (i) augmented graph construction and weight initialisation, (ii) weight learning with modularity maximisation, and (iii) modularity refinement. Specifically, an augmented graph is constructed through centre-based attributed clustering to retain attributes relationships between vertices. All attribute relationships are then transformed into edges in the augmented graph. Modularity maximisation model is then applied to partition the augmented graph, which now contains both attributes and topological information. Along with the partition, a fast learning algorithm is proposed to automatically adjust weights on those attributes relationships according to their contributions towards our objective. Because the result of a greedy modularity maximisation model is heavily affected by its processing order, we propose a modularity refinement algorithm to automatically optimise the result through a fast greedy search.

We evaluated AGGMMR on four open networks which are publicly available, CiteSeer, Cora, Terrorist Attack, and SinaNet networks. As a case study, we evaluate AGGMMR on a payment network from PayPal. The PayPal network consists of 100 *million* vertices and 1.5 *billion* edges. Among the 100 million vertices, 3 *million* are merchants which are described by 68 attributes. Running community detection algorithm on such a large attributed network is challenging for several reasons. First, the network is very large. Many existing algorithms are not scalable enough to be applied on this level of scale. Second, the network contains heterogenous vertices. There are both merchants and consumers, and business related attributes are only available for large merchants. Algorithms that require all vertices to share the same set of attributes cannot be applied. Third, there are both numerical and categorical attributes. Further, the categorical attributes can be many-value or multi-value (details in our case study).

Through the evaluation on public datasets and the large scale PayPal networks, we show the effectiveness of AGGMMR, and its superiority over 5 baselines Louvain [4], Distance Dynamics [31], NAGC [20], CODICIL [27] and K-means.

2 RELATED WORK

We review three categories of community detection algorithms: topological based, attribute based, and community detection using both types of information.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
KDD '19, August 4–8, 2019, Anchorage, AK, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330721

Topological Based Community Detection. A classic method that using topological information is graph cut, which partitions the network by minimising the cost from cut. Minimum cut and normalised cut are the most well studied algorithms in this area. Besides these two, edge betweenness can also be used to measure communities. One representative algorithm is Girvan-Newman algorithms [11]. However, due to high computational cost, those algorithms cannot handle large scale networks [22, 26, 33]. Label propagation [39] finds communities by propagating labels within the network. The algorithm is scalable, but recent research shows it is not robust when running on many common scenarios [36].

Modularity functions, introduced by Newman and Girvan [23] has been extensively used to measure communities. Although the modularity maximization problem is NP-hard, a large number of heuristic algorithms have been developed [10]. One representative algorithm is Louvain [4], which has been widely used in industry applications due to its scalability and high quality performance.

There are also solutions trying to partition communities from other perspectives. Those algorithms differ in the ways to measure distance based on network topological information, *e.g.,* random walk, graph drawing, and game theory [2, 3, 7, 8, 18, 21, 31, 34].

Attribute Based Community Detection. K-means is attribute based clustering that is widely used in research and industries for its simplicity and effectiveness. There are a huge number of extensions [1, 5, 32]. Examples include kernel method on K-means to identify nonlinear structure in data [29], and K-modes and K-prototypes that enable K-means to perform on categorical data [13].

Topology & Attributes Community Detection. Recently, a few algorithms are proposed to detect communities based on both topological and attribute information. Those algorithms can be grouped in three categories [9]: topological based, attribute based, and hybrid approaches.

Topological based approach maps attribute information to topological structures. Then the attributed network community detection problem is transformed into a pure topological clustering problem. The key is to correctly map the attributes and use them to construct an augmented graph. SA-Cluster [38] and Hetero-Attractor [35] use dummy vertices to map attribute values. Each attribute value is represented by a single dummy vertex on the graph. Those dummy vertices are connected to all vertices in the network to retain their attribute relationships. CODICIL [27], SAC2 [6] use attributes to find nearest neighbours for each vertex, then reconstruct graph by connecting each vertex with their neighbours to retain attribute information. Attribute based approach tries to integrate topological information together with attribute distance/similarity to cluster the graph. In the propagation algorithm [19], it is assumed vertices that belong to the same community should receive similar amount of content/attributes. So the problem is transformed into an optimisation problem that minimises the error between vertices' content propagation and expected content propagation in the respective communities. Communities can also be partitioned by embedding vertices through optimising both attribute distance and topology distance together [17]. The CESNA [37] algorithm utilizes generative model to predict community assignments. In [12], the method separates each attribute into a single layer of graph, then ensembles the results from each layer to get the whole graph clustering.

Given a large scale network with complex attributes in our problem setting, existing algorithms are not (at least not directly) applicable. Those algorithms require all vertices to have same set of attributes. Besides that, how those algorithms adjust the weights between attributes and topological information in an unsupervised manner are not clearly discussed.

3 THE AGGMMR FRAMEWORK

Our AGGMMR framework is designed to partition an attributed graph based on its attributes and topological information, through a greedy modularity maximisation model.

Illustrated in Figure 1, the framework consists of three steps. We first construct an augmented graph through an center-based attributed clustering to retain attributes relationships between vertices. In second step, the modularity maximisation model is used to partition the augmented graph. A fast learning algorithm is proposed to automatically adjust weights on attribute relationships according to their contributions towards our objective. In the last step, we propose a modularity refinement algorithm to automatically optimise the result through a fast greedy search. This is to reduce the effect of processing order in the greedy modularity maximisation model in the previous step. Next, we detail the three steps.

3.1 Augmented Graph Construction

One way to retain attributes information on the graph is to plot all attribute values as dummy vertices and connect them with original vertices [38]. However, this method is applicable when attributes are all categorical and the number of attributes is relatively small.

Example 3.1. *Assume there are 100 attributes each with 10 different values in a graph consisting of* $1M$ *vertices. This method will generate* 100×10 *dummy vertices and* $100 \times 1M$ *dummy edges.*

Instead of directly using attribute values as a dummy vertex, we propose to summarize the values into attribute centers. Specifically, we perform center-based attribute clustering on all vertices that have attributes. A dummy vertex is created to represent the center of each attribute cluster (*i.e.,* **attribute center**) and is added to the graph. Then, **belongingness edges** between those attribute centers and their member vertices are added to the augmented graph to retain their attribute relationships.

Example 3.2. *Assume there is a graph with* $1M$ *vertices and* $10,000$ *attribute centers. Only additional* $10,000$ *dummy vertices and at most* $1M$ *belongingness edges are needed to construct an augmented graph. Useful attribute relationships are effectively captured in this much smaller augmented graph.*

Note that our construction method is not limited to a graph with only categorical attributes, but all types of attributes as long as the attributes are available for clustering. Our method is also flexible because all kinds of center-based attribute clustering algorithm can be easily adopted here, to cater to different application requirements.

To indicate the strength of belongingness relationship between each vertex and its nearest attribute center, we use *attribute distance* to initialize the weight of a belongingness edge. An **attribute distance** is the distance between each vertex and their nearest

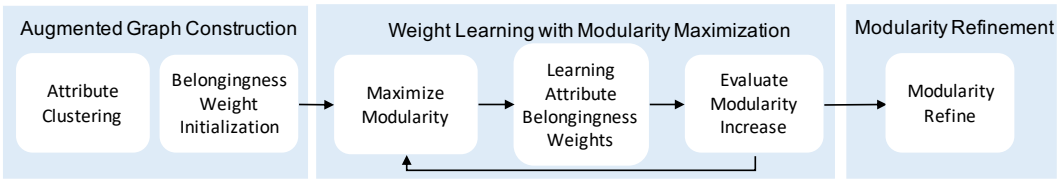


Figure 1: Flow of Framework

attribute center, calculated by the center-based attribute clustering algorithm. For example, we can use Euclidean distance if k-means algorithm is used to cluster attribute values. We denote attribute distance by $d(v_i, v_c)$ between vertex v_i and attribute center v_c.

In our implementation, we first compute Euclidean distances as attribute distances, and then map the distances into probability values. Specifically, we map Euclidean distances into higher dimensional space using RBF kernel.

$$P(v_i, v_c) = exp(-\frac{d(v_i, v_c)}{2\sigma^2}) \tag{1}$$

As the kernel distance embeds isometrically in a Euclidean space, the RBF kernel function is an effective metric for weighting observations [24]. Then the **weight initialization** on belongingness edges are by Equation 2.

$$w(v_i, v_c) = dt(v_i) \times P(v_i, v_c) \tag{2}$$

Here, $dt(v_i)$ is the weighted degree of vertex v_i in the topological graph, *i.e.,* the original graph before adding attribute centers as dummy vertices. This weighing scheme is designed to balance the weights between attribute information and topological information for each vertex in the augmented graph at the initial stage.

3.2 Weight Learning with Modularity Maximisation

In the augmented graph, both attributes and topological relationships are represented by edges. We therefore use a topological based clustering method to partition the augmented graph. Intuitively, densely connected vertices should be in a community as they share either strong attributes or strong topological relationships, or both. Modularity maximisation is one of such objective functions that have shown to be effective [9, 25, 31]. We adopt the modularity maximisation model from [23]

$$Q = \frac{1}{2m} \sum_{ij} [A_{ij} - \frac{da(v_i) \times da(v_j)}{2m}] \delta(v_i, v_j) \tag{3}$$

where m corresponds to the cardinality of edges in the graph, $da(v_i)$, $da(v_j)$ are the weighted degrees of vertices v_i and v_j in the augmented graph, respectively. A_{ij} is the ij-th component of the adjacency matrix of the graph, and A_{ij} equals to edge weight if vertices v_i and v_j are adjacent, and 0 otherwise. $\delta(v_i, v_j)$ equals to 1 when v_i and v_j belongs to the same community, and to 0 otherwise.

In our implementation, we adopt the Louvain [4] algorithm for modularity maximisation. At beginning, each vertex is assigned with an individual community. In every iteration, each vertex is compared with its neighbours' community assignments, and assigned to the one with maximum modularity gain. The computation of modularity gain is based on the weights of the edges and we refer to [4] for details.

On augmented graph, vertices are partitioned on both attributes and topological relationships. Since both types of relationships are represented by edges, there are three situations that two vertices are assigned a same community through modularity maximisation: (i) They are densely connected and they have strong attribute relationships. (ii) They are densely connected but they have trivial attributes relationships. (iii) They are not densely connected but their attributes relationships are strong enough to connect them.

In a real life network, some attribute relationships could be trivial for many communities, *e.g.,* the merchants who have loans are connected based on this 'haveLoan' attribute and this attribute is only useful when clustering business loans community. We want to minimise the influence from such attribute relationships when it is trivial. At the same time, we aim to increase the importance of meaningful attribute relationships. To this end, we propose an **unsupervised weight learning algorithm**, align with modularity maximisation objective, to automatically adjust the weights for attribute relationships according to their contributions in the clustering.

Assume most of vertices from a same attribute center have been assigned to the same community in an iteration, then it indicates that the attribute based relationship from this attribute center provides positive contribution in our community detection task. In opposite, if most of vertices from an attribute center have been assigned to a large number of different communities, then this attribute based relationship is very weak and might introduce noise to our task. The weights of belongingness edges to attribute centers therefore can be adjusted accordingly. To update weights of belongingness edges, we first calculate a clustering contribution score for each attribute center.

The **contribution score** for an attribute center, denoted by Θ_a is calculated through $\Theta_a = \frac{|V_a|}{|C_a|}$. V_a is the set of vertices that connect to this attribute center; C_a is the set of communities that the member vertices in V_a are assigned to, through modularity maximisation in the current iteration. The value of Θ_a is bounded between 1 to $|V_a|$ as $|C_a|$ varies from 1 to $|V_a|$. The more vertices an attribute center connects, the higher potential contributions this attribute center will have. That is, an attribute center connecting to 10,000 vertices and all its vertices distributed in the same community contributes more than an attribute center who connects only 10 vertices in the same situation.

To meet the constraint that the total edge weights does not change, *i.e.,* $\sum_{i=1}^{n} w_i^{t+1} = \sum_{i=1}^{n} w_i^t$, where w_i^{t+1} is the weight of a belongingness edge in iteration $t+1$, we redistribute the weights belongingness edges as follows:

$$w_i^{t+1} = \frac{1}{2}(w_i^t + \delta w_i^t) \qquad \delta w_i^t = \frac{\Theta_a}{\Sigma\Theta} \times W \qquad W = \sum w^t \quad (4)$$

Theorem 1. *In each iteration, the weights are adjusted towards the direction of increasing the modularity objective*

Rewriting the modularity maximisation (Equation 3) on this augmented graph, we have

$$\begin{aligned} Q &= \frac{1}{2m}(Q_s + Q_d) \\ Q_s &= \frac{1}{2m}\sum_{lk}[A_{lk} - \frac{da(v_l) \times da(v_k)}{2m}]\delta(v_l, v_k) \\ Q_d &= \frac{1}{2m}\sum_{ij}[A_{ij} - \frac{da(v_i) \times da(v_j)}{2m}]\delta(v_i, v_j) \end{aligned} \quad (5)$$

where v_l,v_k are vertices belonging to a same attribute center and v_i,v_j are vertices belonging to different attribute centers. $\delta(\cdot,\cdot)$ is the same as in Equation 3, and its value is 1 if the two vertices are in the same community and 0 otherwise.

The modularity of a graph is in proportion to the sum of differences between connections and expected connections from every pair of vertices that are in a same community. In the above equation, we use Q_s to represent the sum of modularity calculated from the pairs of vertices in a same attribute center and uses Q_d to represent the sum of modularity calculated from the pairs of vertices in different attribute centers. In weight learning, it only affects the modularity of Q_s while not the modularity of Q_d. Note that in each iteration, an attribute center will also been assigned to one of its member's communities according to its relationships with its members. When we increase the weights of attribute relationships from an attribute center, we are increasing A_{lk} between the member vertices to that attribute center. In this way, we increase Q_s more as most of vertices are likely to be assigned into the same community with their attribute center. In opposite, when we decrease the weights of attribute relationships, Q_d decreases less since most of vertices connect to that attribute center are assigned into different communities. After each iteration, the modularity will be compared with the value in the previous iteration. The learning algorithm converges once there is no more modularity increase in an iteration.

4 MODULARITY REFINEMENT

We now present the last step in AGGMMR framework, modularity refinement, to optimise our partition result from the greedy modularity maximisation model.

4.1 Drawbacks of Greedy Method

Greedy modularity maximisation reduces the computation cost significantly, however, the quality of result highly depends on the order of processing.

Greedy modularity maximisation process can be demonstrated from the perspective of dynamic programming. To find the community assignments which maximise the global modularity of a graph G, in each step, we assign a vertex into one of its neighbours' communities. Naturally, the sub problems of this dynamic programming is to find optimal community assignments for previous n vertices.

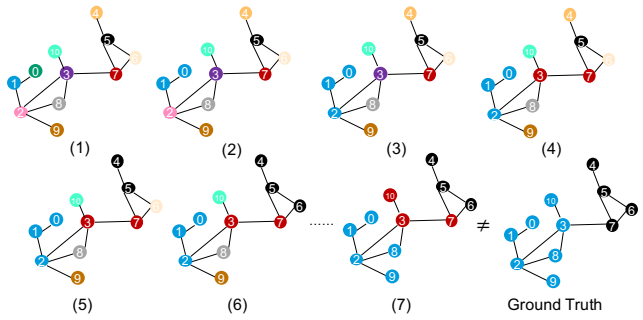


Figure 2: The effect of processing order in an example graph (edge weight not indicated in the graph, best view in color).

This recursion can be expressed as

$$Q[n] = max(Q_1[n-1] + q_{n-1,n}, Q_2[n-2] + q_{n-2,n} \ldots\ldots) \quad (6)$$

The $Q_k[n-k]$ represents the optimal community assignments for previous $n-k$ vertices while $q_{n-k,n}$ represents the later k vertices' assignments. In this greedy method, this recursion is

$$Q[n] = Q_1[n-1] + q_{n-1,n} \quad (7)$$

The community assignment of current vertex depends on the previous assignments. We assume the previous assignments for $n-1$ vertices are always the optimum solution even after we have assigned the current vertex. However, this is not true and the assignments for initial vertices are not always reliable. In first iteration, when we process the first half vertices, we have no information about remaining vertices' community assignments. That is, the greedy method only takes very limited information when it processes most of vertices in the network. This process may easily trap the result into a huge different local optimum due to a domino effect. In following iterations, even we have already obtained all vertices' community assignments, the situation will not get better due to a *mutual effect*.

Assume vertices v_b and v_c are processed after vertex v_a, and both are assigned into the same community with v_a. When we re-evaluate v_a's community assignment, the community assignments of vertices v_b and v_c will back affect vertex v_a's community re-assignment by keeping it from re-assigning to other communities. We define such effect as **mutual effect**. We use an example in Figure 2 to further illustrate this.

In Figure 2, the number on each vertex represents its order of processing. Each colour represents an individual community assignment. At step (1), each vertex is initialised to its own individual community. Then every vertex is compared with its community assignment with neighbours' communities according to its modularity gain. When processing v_3 in step (3), it has neighbour vertices v_2, v_7, v_8 and v_{10}. At that moment, vertices v_8, v_9 and v_{10} are in their individual communities because they have not been processed.

In the ideal result, indicated by ground truth, vertices v_8, v_9 and v_{10} are in the same community with v_2 according their connectivity. Thus v_3 should also be assigned to the same community for its dense connections with v_2, v_8, v_9 and v_{10}. However, at step (3), we do not have those information. In the end, v_3 is assigned to the same community with v_7. This assignment also influences further

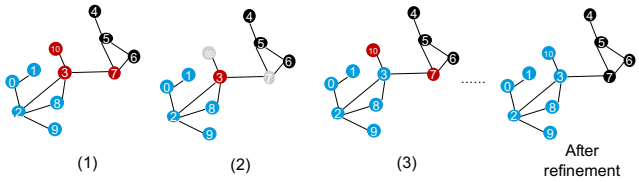


Figure 3: Example of Refinement (best view in color)

assignments after we processing v_3. As in this example, vertices v_3, v_7, and v_{10} eventually form a community in step (7).

This scenario happens frequently since the earlier assignment of a vertex, the less information can be used for that assignment. A single edge with large weight will easily trap two vertices into a bad assignment when we do not have enough information. In a payment network, one occasional transaction involving a large amount will trap two merchants into the same community and thus affect further assignments.

4.2 Modularity Refinement

Completely getting rid of the aforementioned scenario *i.e.,* finding the optimum solution for a modularity maximisation problem, is np-hard and not practical. However, we can do an effective greedy refinement to optimise the result as much as possible. The idea of refinement is to give each vertex a second chance to re-assign its community after we have the whole picture of assignments, and at the same time to eliminate mutual effect when re-assigning a vertex.

In refinement, all vertices' community assignments will be re-evaluated in the same order from previous iteration. When re-evaluating a vertex v_i, v_i will be compared with three types of neighbours: (i) a neighbour has the same community assignment as v_i, assigned before v_i's assignment, (ii) a neighbour has same community assignment as v_i, assigned after v_i, and (iii) the neighbour has different community assignment from v_i.

We temporarily mask a neighbour who is assigned the same community after v_i an individual community. The masked vertices are those whose community assignments are directly affected by the current vertex v_i in the greedy modularity maximisation process. Note that those vertices processed after v_i but have different community assignments from v_i are not masked. Now, we re-evaluate v_i's community assignment. If v_i's assignment is changed to one of its neighbors, say v_n's assignment, because of largest modularity gain, then we have two possible cases. If v_n is a masked neighbor, then v_i keeps its original assignment, *i.e.,* v_i's community assignment is unchanged during the re-evaluation. However, if v_n is not a masked neighbor, then v_i will be re-assigned to v_n's community.

Figure 3 provides an example to illustrate how refinement works. In ground truth, vertices v_3, v_{10} belong to the blue community on the left, while vertex v_7 belongs to black community on the right. In step (1), we have community assignments of all vertices from greedy modularity maximisation, and vertex v_3 is assigned to a community with vertices v_7 and v_{10}. When we re-evaluate vertex v_3 in step (2), vertices v_7 and v_{10} will be temporarily masked to individual communities, because they originally were processed after v_3 and at the same time they are in the same community as v_3.

Table 1: Statistics of the four open networks

Dataset	Vertices	Edges	Attributes
Citeseer	3,312	4,732	3,702 binary attributes
Cora	2,708	5,429	1,433 binary attributes
Terrorist	1,293	3,172	106 attributes
SinaNet	3,490	30,282	LDA topics

After the masking, there will be no longer mutual effects between vertices v_3, v_7, and v_{10}. After re-evaluation, v_3 is re-assigned to the blue community on the left, since joining into it produces larger modularity gain than joining either community of v_7 or of v_{10}. The relationship between either v_7 or v_{10} to v_3 is not strong enough to continue keeping v_3 in their communities once we eliminate the mutual effect.

To summarize, through this refinement process, we eliminate the mutual effect but retain all other useful information on the graph as much as possible. If a vertex is re-assigned to one of those masked neighbours' community again, it indicates these vertices have strong relationships to bound them in the same community.

4.3 Time Complexity Analysis

The whole framework is fast and scalable with a time complexity of $O(n \log n)$, where n is the number of vertices. For augmented graph construction phase, the running time required largely depends on the chosen attribute based clustering algorithm. The good thing is, the attribute based clustering algorithm is able to run separately on attributes. The time complexity for our framework itself is linear to the number of attribute centers and belongingness edges. In the phases of modularity maximisation and modularity refinement, it has the same time complexity with Louvain algorithm, runs in $O(n \log n)$. For the phase of weight learning, it runs in $O(n)$ as it updates weights for every vertex iteratively.

5 EXPERIMENT ON OPEN NETWORKS

We conduct experiments to evaluate AGGMMR on four publicly available networks, against five baselines. All the four open networks come with ground truth for validation.

5.1 Datasets and Evaluation Metrics

Open Network Datasets. Listed in Table 1, **CiteSeer** is the network extracted from CiteSeer digital library [30]. Each vertex represents a publication and an edge indicates a citation relationship. The binary attributes attached to each vertex show whether a word in the vocabulary is present or absent. The **Cora** citation network also consists of scientific publications, and the binary attributes of each vertex indicate whether a word in the vocabulary is present or absent [30]. In the **Terrorist** attack network, each vertex represents an attack while the edges connect co-located attacks. Each attack is described by a set of 106 attributes. The **SinaNet** is a microblog user relationship network extracted from sina-microblog website. Each vertex represents a user and each edge represents a relationship. Attributes are extracted by LDA topic model that represent users' topic distribution [15].

Table 2: Evaluation on Open Network Datasets

	Purity						Fscore					
Dataset	Louvain	DDynamic	K-means	NAGC	CODICIL	AGGMMR	Louvain	DDynamic	K-means	NAGC	CODICIL	AGGMMR
Citeseer	0.7355	0.5041	0.4457	0.7271	0.6760	**0.7656**	0.7250	0.5896	0.5098	0.7001	0.6595	**0.7501**
Cora	0.7651	0.5194	0.3811	0.7858	0.7991	**0.8342**	0.7567	0.4878	0.3992	0.7736	0.7800	**0.8238**
Terrorist	0.7318	0.7318	0.5209	0.7069	0.7349	**0.7504**	0.5498	0.5534	0.5192	0.5265	0.5660	**0.5892**
SinaNet	0.2506	0.3265	0.7684	0.5484	0.6550	**0.8131**	0.1924	0.3380	0.7462	0.4610	0.6051	**0.7653**

FScore and Purity. The evaluation metrics used in this paper are Fscore (or F1 measure) and Purity. F1 is the harmonic mean of precision and recall. Precision measures the ratio of vertices with ground truth labels in a detected community, and recall measures the coverage of vertices with the ground truth label in a detected community. Purity is the fraction of the vertices belonging to the community label. It is averaged in a weighted way over different communities in order to create a composite community purity measure. A good community detection method should obtain both high Purity and high Fscore.

5.2 Comparison Methods

We compared the proposed AGGMMR with Louvain [4, 16], Distance Dynamics [31], NAGC [20], CODICIL [27] and K-means. **K-means** is the most widely used and efficient attribute based clustering algorithm. **Louvain** is one of the most widely used modularity based method which is not only well known in research communities but also in industries due to its good scalability and quality performance. AGGMMR also utilizes Louvain for modularity maximization on the augmented graph constructed in the first step. **Distance Dynamic** is a state-of-the-art algorithm detecting community naturally through vertices' interaction dynamics. **NACG** is the state-of-the-art non-negative matrix decomposition based community detection algorithm that takes both attributes and topological information into consideration. **CODICIL** is a three-step framework that transforms attributed graph into a nearest neighbour based augmented graph, and then finds communities with coherent attributes and topological structure. Compared with traditional matrix decomposition and generative method, CODICIL is more efficient and scalable. To summarize, Louvain and Distance Dynamics only utilise topological information for partition while K-means only uses attribute information for partition. NACG and CODICIL partition the graph by utilising both types of information.

Since K-means algorithm requires K as input. We set K the same value as it is in ground truth and also tested those K values detected by algorithms that do not need number of clusters as input in order to compare with them. For CODICIL algorithm, we set parameter N as 50 which is the value suggested by authors, and we perform parameter search for α from 0.1 to 0.9 to balance the weights from two types of information.

5.3 Results

From the results reported in Table 2, AGGMMR outperforms all five baselines on all four datasets, on both measures. The results also show that all methods based on both attributes and topological information have better and consistent performance in general. Due to the existence of inconformity between attributes and topological information, a method that focuses only on a single type of information may achieve a higher quality result on some datasets. For instance, on SinaNet network, attributes are user's topic distribution which are directly related to users' labels. K-means algorithm achieves a much better results than other baselines because of the usefulness of the attributes for clustering.

6 CASE STUDY ON PAYPAL APAC NETWORK

After showing good results on open networks, we now present a case study on a large scale network from PayPal. We illustrate how our framework can be adapted to handle complex attributes on large complex networks.

PayPal is a global leading company that provides international payment services. It has a unique two-sided payment business with a large network connecting both consumers and merchants. The goal of our case study is to uncover merchant communities in PayPal's APAC network using both attributes and topological information.

6.1 PayPal Networks

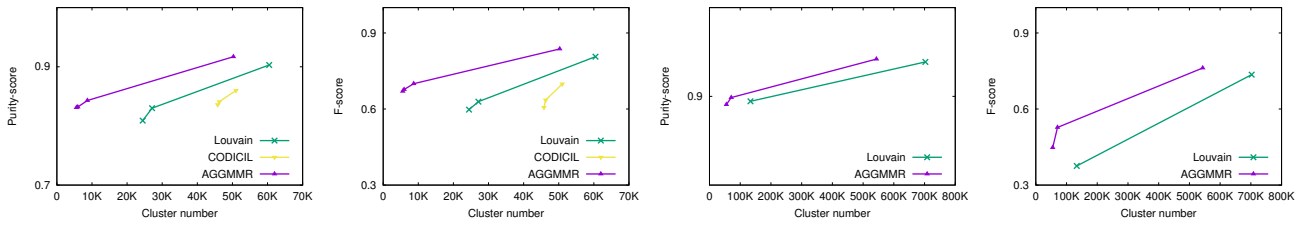
We use two PayPal internal datasets with (partial) ground truth labels in this case study. The **Philippine Network** consists of $1.5million$ users with $2million$ transactions between those users. All merchants in this network are located in Philippine while the consumers are from all over the world. It is a subset of the APAC network which contains a large number of freelancer businesses. The **APAC Network** consists of $100million$ users with $1.5billion$ transactions between those users. This network includes all merchants located in the APAC region while the consumers are from all over the world.

A user (*i.e.,* a vertex in the network) here can be either a merchant or a consumer. The edges are weighted by the amount of the transaction between the two users. We leverage a set of 68 attributes associated with majority of merchants in our case study.[1] Those attributes contain general information about merchants' business and the PayPal products they have used. However, not all attributes are available for all merchants, *i.e.,* there are many missing attributes and missing values as expected.

6.2 Handling Complex Data Types

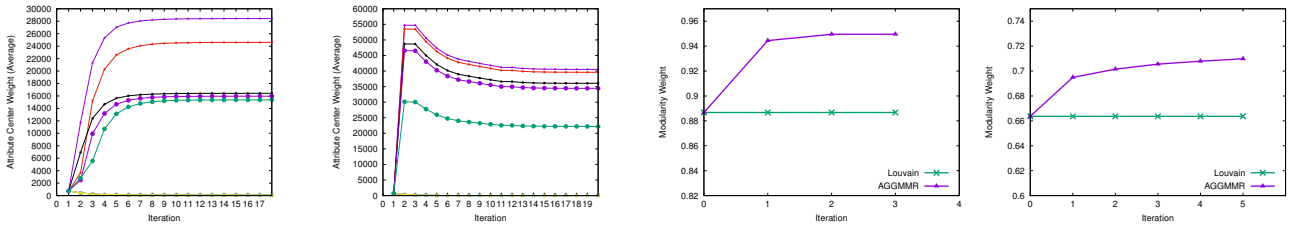
Attributes in PayPal network are in various different data types. For example, a merchant's business region is a categorical value while its payment volume through a specific PayPal product is numerical.

[1]The attributes are produced internally. Due to privacy consideration, we masked the schema of those attributes.



(a) Purity on PayPal's Philippines Network (b) Fscore on PayPal's Philippines Network (c) Purity on PayPal's APAC Network (d) Fscore on PayPal's APAC Network

Figure 4: Community Quality Comparisons on PayPal's Network



(a) Weight Learning on Philippines Network (b) Weight Learning on APAC Network (c) Modularity Refinement on Philippines Network (d) Modularity Refinement on APAC Network

Figure 5: Weight Learning and Modularity Refinement on PayPal's Network

Clustering on attributes with mixed types is challenging. Nevertheless, our framework is flexible enough to adapt different methods for attribute clustering. In this case study, we use K-prototype algorithm to cluster attributes and construct the augmented graph. k-prototype extends k-means clustering algorithm and is efficient for clustering large data sets with mixed attribute types [14].

Besides mixed data types, in our data set, there are two additional special data types that are unable to be processed directly by traditional data processing algorithms. One is **many-value categorical attribute** and another is **multi-value categorical attribute**. The former refers to attributes that contain a large cardinality of values. For example, the value of "country code" may contain more than one hundred country codes. One hot encoder on this type of attribute leads to sparse latent dimensions which decreases clustering performance. The latter, multi-value categorical attribute refers to attributes that contain multiton values. One example is attribute "product bundle". Each value of it is a set of singleton values such as *product A, product B*. Those complex data types need to be specially handled before it can be used for clustering. We extends k-prototype algorithms for this purpose.

The k-prototype algorithm clusters data against k prototypes instead of k means. Each prototype is represented by a vector which is a combination of numerical attributes and categorical attributes. In each iteration, it updates numerical attributes by their means while updates categorical attributes by their modes. In k-prototype, the distance between a vertex v_i and a prototype v_p is defined by

$$d(v_i, v_p) = \sum_{j=1}^{m_r} (v_{ij}^r - v_{pj}^r)^2 + \gamma \sum_{j=1}^{m_c} \delta(v_{ij}^c, v_{pj}^c) \tag{8}$$

where m_r is the number of numerical attributes, v_{ij}^r and v_{pj}^r are values of a numeric attribute of v_i and v_p respectively. m_c is the number of categorical attributes and v_{ij}^c and v_{pj}^c are values of a categorical attribute. γ is a weight balancing the two types of attributes. $\delta(v_{ij}^c, v_{pj}^c) = 0$ if $v_{ij}^c = v_{pj}^c$, and $\delta(v_{ij}^c, v_{pj}^c) = 1$ otherwise.

Modified distance measure. Simply uses 1 to indicate the difference between two different categorical values (*i.e.,* $\delta(v_{ij}^c, v_{pj}^c)$) fails to reflect the attribute's value distribution. Further, this equation is inadequate to handle many-value and multi-value categorical attributes.

Example 6.1. *Suppose a categorical attribute have 3 possible values: A, B, and C. Among 10 vertices, 8 vertices have value A while 1 vertex has value B and another has value C. Intuitively, the difference between value A and value C (or A and C) should larger than the difference between value B and value C if we consider their distributions.*

To retain categorical value distribution, and to handle multi-value and many-value categorical attributes, we normalize attribute values as follows in this case study.[2]

- Numerical attributes are normalize by z-score normalization.
- Categorical attributes are encoded by one hot encoder and normalized by z-score normalisation (Normalization may loss interpretability but depends on application).
- For multi-value and many-value categorical attributes, we normalise each singleton value by z-score normalisation, and store each value as a ⟨categorical value, z-score⟩ pair. A multi-value attribute is stored as a set of key-value pairs.

[2]Other forms of normalization than z-score normalization can be applied as well. Whether to normalize a categorical attribute depends on the needs from application.

The distance between a vertex v_i and a prototype v_p is redefined as:

$$d(v_i, v_p) = \sum_{j=1}^{m_r} \|(\hat{v_{ij}^r} - \hat{v_{pj}^r})\| + \sum_{j=1}^{m_c} \|(\hat{v_{ij}^c} - \hat{v_{pj}^c})\| \delta(v_{ij}^c, v_{pj}^c) + \sum_{j=1}^{m_u} \mathbb{J}(\hat{v_i^u}, \hat{v_p^u}) + \sum_{j=1}^{m_a} \|(\hat{v_{ij}^a} - \hat{v_{pj}^a})\| \delta(v_{ij}^a, v_{pj}^a) \quad (9)$$

In above equation, $\hat{}$ denotes normalized values. v^u is a value of multi-value attribute and v^a represents a value of many-value attribute. With respect to the original distance, we use the difference of normalized values between two categorical values to represent their distance, instead of 1.

For multi-value attributes, each value is a set of key-value pairs. We calculate their distances using weighted Jaccard distance $\mathbb{J}$.

$$\mathbb{J}(\hat{v_i}, \hat{v_p}) = 1 - \frac{\sum_{x \in \hat{v_i} \cap \hat{v_p}} w(x)}{\sum_{y \in \hat{v_i} \cup \hat{v_p}} w(y)} \quad (10)$$

Here $w(x)$ is the normalized value of x. Weighted Jaccard distance, values in the range of [0,1], measures the dissimilarity between two multi-value attributes.

Prototype Updating for Categorical Variable The original k-prototype algorithm updates a categorical attribute of a prototype in two steps: (i) calculate the frequency for all categories, and (ii) assign the prototype the category with highest frequency.

This updating scheme can be directly extended to many-value and multi-value attributes. For multi-value attribute, the value of a prototype is a set of singleton values. It means that we need to choose multiple singleton values as a set, to update it in an iteration. For example, given 4 attributes, each has its 4, 5, 4, 3 singleton values respectively, listed in a column.

$$\begin{bmatrix} c_{1,1} & c_{2,1} & c_{3,1} & c_{4,1} \\ c_{1,2} & c_{2,2} & c_{3,2} & c_{4,2} \\ c_{1,3} & c_{2,3} & c_{3,3} & c_{4,3} \\ c_{1,4} & c_{2,4} & c_{3,4} & \\ & c_{2,5} & & \end{bmatrix}$$

If k is 3, we assign 3 prototypes with 4 multi-value attributes as:[3]

$p_1 = \{\{c_{1,1}, c_{1,3}\}, \{c_{2,1}, c_{2,2}\}, \{c_{3,2}\}, \{c_{4,1}, c_{4,3}\}\}$,
$p_2 = \{\{c_{1,2}\}, \{c_{2,3}, c_{2,4}\}, \{c_{3,2}\}, \{c_{4,2}\}\}$,
$p_3 = \{\{c_{1,4}\}, \{c_{2,2}\}, \{c_{3,3}, c_{3,4}\}, \{c_{4,3}\}\}$

A singleton value is considered frequent if it is shared by majority vertices in a cluster. Based on this intuition, multi-value attribute can be updated in two steps: (i) calculate frequencies for all singleton values of one multi-value attribute, and (ii) assign to the prototype the set of singleton values where each value is shared by more than half vertices in the cluster.

In other words, when a value is shared by more than half of vertices in a cluster, it will be updated to the prototype because it is considered a common feature to that cluster. One concern is that whether we can avoid updating two mutual exclusive singleton values to a single prototype at the same time. The answer is positive and stated below.

[3]The values in this example are randomly assigned for purpose of illustration.

Lemma 1. *Two frequent values updated to prototype will never be mutual exclusive.*

It can be proved by contradiction. Two singleton values are mutual exclusive if they are unable to appear in a same multi-value attribute instance. Let us assume two singleton values v_1 and v_2 are mutual exclusive and both of them have been chosen to update to a prototype in a cluster. So there are more than half vertices containing value v_1 and more than half vertices containing value v_2. To satisfy both conditions, at least one vertex shall contain both v_1 and v_2, which contradicts with our assumption. As a result, our model does not take mutual exclusive values when updating a multi-value attribute for a prototype.

6.3 Analysis of Clustering Results

There are many challenges when we run community detection algorithms on real PayPal network compared to running on open data sets. The attributes in PayPal's network provide very important information for clustering. However, not all merchants have all attributes available and at the same time there are aforementioned special attribute types. Those methods that solely rely on attributes and methods require all vertices to have the same set of attributes are not directly applicable on this network.

In our case study, we compare AGGMMR with Louvain and CODICIL. Louvain is known for its scalability and it only uses topological information for partition. So it can be directly applied to our network. For CODICIL, instead of using union edges to replace all original edges, we calculate union edge for vertices who have this set of attributes and left other edges with their original weights. With this minor modification, CODICIL is able to run on PayPal's network without any additional assumptions. However, due to the large complexity for calculating nearest neighbours for each vertex in CODICIL, we did not manage to run it successfully on our APAC data set, the data with 100 million vertices.

The Purity and FScore results plotted in 4 show that AGGMMR yields much better results than the two baselines. As Louvain algorithm neglects attributes information, it only detects community with dense connections. The CODICIL algorithm does not perform as well as expected even it considers both types of information. There might be two possible reasons. First is that CODICIL algorithm prunes information on the graph to reduce computation cost. As the result, a lot of information is pruned if we force it to run on a very large graph. The second possible reason is that it does not well handle the noise from the unconformity between attributes and topological information in our network. Simply combining both types of information may not always given better performance. AGGMMR tends to cluster communities that conform from both types of information; it is robust to handle these scenarios. Our internal evaluation of communities detected by AGGMMR shows a strong goodness of fit with existing professional knowledge.

6.4 Analysis of Weight Learning

Figures 5(a) and 5(b) report weight changing in the weight learning phase on PayPal network. Since belongingness edge weights are changed based on each attribute center, we selected 10 attributes center's average weights (δw_i^t in Equation 4) to analyze their updating by iterations. Half of them are attribute centers that have

largest positive weight changes and others are the centers that have largest negative weight changes. Those centers with largest positive weight changes are those centers whose members are distributed in very small number of communities by modularity maximisation. This is consistent with our expectation. If an attribute center contains large number of members, and most of them concentrate on a small number of communities, it indicates the attributes relationship for this attribute center provides a very positive contribution in clustering. The attribute centers with largest negative weight changes are those centers whose members are distributed into different communities. The connections from those attribute centers provide trivial information or even noise. These attribute centers would be adjusted to much smaller weights automatically in our weight learning phase.

6.5 Analysis of Modularity Refine

Figures 5(c) and 5(d) show the trend of modularity refinement when running AGGMMR on PayPal network. The modularity score from Louvain method is also plotted as a baseline for comparison. We observe that the modularity score is continuously increasing through the whole process of our refinement, and there always exists a significant modularity increase at the first iteration of refinement. It illustrates the fact that in most of cases, a small number of false assignment caused by poor processing ordering would result a huge different result. Such cluster assignments can be effectively refined through our greedy refinement process in a small number of iterations.

7 CONCLUSION

In this paper, we propose a practical framework for community detection based on attributes and topological information for large network with mixed types of attributes. The attributes information can be effectively captured and transformed into an augmented graph through attribute clustering. After that, a weight learning process is designed to automatically adjust weights from those attributes information according to their contributions to clustering. A greedy based modularity maximisation algorithm is adopted to partition the graph based on both types of information and finally a modularity refinement process is introduced to optimise the result effectively. Experiments on different open data networks and real industry case studies on PayPal's large networks demonstrate that our proposed framework is effective and practical in industry applications.

REFERENCES

[1] D. Arthur and S. Vassilvitskii. 2007. k-means++: The advantages of careful seeding. In *Proc. ACM-SIAM symposium on Discrete algorithms*. 1027–1035.
[2] N. Barbieri, F. Bonchi, and G. Manco. 2013. Influence-based network-oblivious community detection. In *ICDM*. 955–960.
[3] Y. Bian, J. Ni, W. Cheng, and X. Zhang. 2017. Many Heads are Better than One: Local Community Detection by the Multi-Walker Chain. In *ICDM*. 21–30.
[4] V. D Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. 2008. Fast unfolding of communities in large networks. *J. of statistical mechanics: theory and experiment* 2008, 10 (2008).
[5] X. Cai, F. Nie, and H. Huang. 2013. Multi-View K-Means Clustering on Big Data. In *IJCAI*. 2598–2604.
[6] T. Dang and E. Viennet. 2012. Community detection based on structural and attribute similarities. In *Int'l conf. on digital society*. 7–12.
[7] H. Dev. 2014. A user interaction based community detection algorithm for online social networks. In *SIGMOD*. ACM, 1607–1608.
[8] M. Ester, H.-P. Kriegel, J. Sander, and X Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*. 226–231.
[9] I. Falih, N. Grozavu, R. Kanawati, and Y. Bennani. 2018. Community detection in Attributed Network. In *WWW*. 1299–1306.
[10] A. Geyer-Schulz and M. Ovelgönne. 2014. The randomized greedy modularity clustering algorithm and the core groups graph clustering scheme. In *German-Japanese Interchange of Data Analysis Results*. Springer, 17–36.
[11] M. Girvan and M. EJ Newman. 2002. Community structure in social and biological networks. *PNAS* 99, 12 (2002), 7821–7826.
[12] Y. Huang and H. Wang. 2016. Consensus and multiplex approach for community detection in attributed networks. In *IEEE GlobalSIP*. 425–429.
[13] Z. Huang. 1997. Clustering large data sets with mixed numeric and categorical values. In *PAKDD*. 21–34.
[14] Z. Huang. 1998. Extensions to the k-means algorithm for clustering large data sets with categorical values. *DMKD* 2, 3 (1998), 283–304.
[15] C. Jia, Y. Li, M. B Carson, X. Wang, and J. Yu. 2017. Node Attribute-enhanced Community Detection in Complex Networks. *Scientific Reports* 7, 1 (2017), 2626.
[16] Jure Leskovec and Rok Sosič. 2016. SNAP: A General-Purpose Network Analysis and Graph-Mining Library. *ACM Transactions on Intelligent Systems and Technology (TIST)* 8, 1 (2016), 1.
[17] Y. Li, C. Sha, X. Huang, and Y. Zhang. 2018. Community Detection in Attributed Graphs: An Embedding Approach. In *AAAI*.
[18] S. Lim, J. Kim, and J.-G. Lee. 2016. BlackHole: Robust community detection inspired by graph drawing. In *ICDE*. 25–36.
[19] L. Liu, L. Xu, Z. Wangy, and Enhong C. 2015. Community detection based on structure and content: a content propagation perspective. In *ICDM*. 271–280.
[20] S. Maekawa, K. Takeuch, and M. Onizuka. 2018. Non-linear Attributed Graph Clustering by Symmetric NMF with PU Learning. *arXiv preprint arXiv:1810.00946* (2018).
[21] A. Mahmood and M. Small. 2016. Subspace based network community detection using sparse linear coding. In *ICDE*. 1502–1503.
[22] M. EJ Newman. 2004. Detecting community structure in networks. *The European Physics Journal B* 38, 2 (2004), 321–330.
[23] M. EJ Newman. 2006. Modularity and community structure in networks. *PNAS* 103, 23 (2006), 8577–8582.
[24] J. M Phillips and S. Venkatasubramanian. 2011. A gentle introduction to the kernel distance. *arXiv preprint arXiv:1103.1625* (2011).
[25] A. Prat-Pérez, D. Dominguez-Sal, and J.-L. Larriba-Pey. 2014. High quality, scalable and parallel community detection for large real graphs. In *WWW*. 225–236.
[26] M. J. Rattigan, M. Maier, and D. Jensen. 2007. Graph Clustering with Network Structure Indices. In *ICML*.
[27] Y. Ruan, D. Fuhry, and S. Parthasarathy. 2013. Efficient community detection in large networks using content and links. In *WWW*. 1089–1098.
[28] P. I. Sánchez, E. Müller, U. L. Korn, K. Böhm, A. Kappes, T. Hartmann, and D. Wagner. 2015. Efficient algorithms for a robust modularity-driven clustering of attributed graphs. In *SDM*. 100–108.
[29] B. Schölkopf, A. Smola, and K.-R. Müller. 1998. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation* 10, 5 (1998), 1299–1319.
[30] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93.
[31] J. Shao, Z. Han, Q. Yang, and T. Zhou. 2015. Community detection based on distance dynamics. In *KDD*. 1075–1084.
[32] X.-B. Shen, W. Liu, I. W Tsang, F. Shen, and Q.-S. Sun. 2017. Compressed K-Means for Large-Scale Clustering.. In *AAAI*. 2527–2533.
[33] H. Shiokawa, Y. Fujiwara, and M. Onizuka. 2013. Fast algorithm for modularity-based graph clustering. In *AAAI*. 1170–1176.
[34] P. L. Szczepanski, A. S. Barcz, T. P. Michalak, and T. Rahwan. 2015. The Game-Theoretic Interaction Index on Social Networks with Applications to Link Prediction and Community Detection. In *IJCAI*. 638–644.
[35] X. Wang, J. Song, K. Lu, and X. Wang. 2017. Community detection in attributed networks based on heterogeneous vertex interactions. *Applied Intelligence* 47, 4 (2017), 1270–1281.
[36] Y. Yamaguchi and K. Hayashi. 2017. When does label propagation fail? a view from a network generative model. In *IJCAI*. 3224–3230.
[37] J. Yang, J. McAuley, and J. Leskovec. 2013. Community detection in networks with node attributes. In *ICDM*. 1151–1156.
[38] Y. Zhou, H. Cheng, and J. X. Yu. 2009. Graph clustering based on structural/attribute similarities. *PVLDB* 2, 1 (2009), 718–729.
[39] X. Zhu and Z. Ghahramani. 2002. *Learning from labeled and unlabeled data with label propagation.* Technical Report CMU-CALD-02-107. Carnegie Mellon University.