A

# Description-driven Community Detection

SIMON POOL, Universiteit Utrecht, the Netherlands
FRANCESCO BONCHI, Yahoo! Research Barcelona, Spain
MATTHIJS VAN LEEUWEN, KU Leuven, Belgium

Traditional approaches to community detection, as studied by physicists, sociologists, and more recently computer scientists, aim at simply partitioning the social network graph. However, with the advent of online social networking sites, richer data has become available: beyond the link information, each user in the network is annotated with additional information, e.g., demographics, shopping behaviour, or interests. In this context it is therefore important to develop mining methods which can take advantage of all available information. In the case of community detection this means finding *good communities* (a set of nodes cohesive in the social graph) which are associated with *good descriptions* in terms of user information (node attributes).

Having good descriptions associated to our models make them understandable by the domain experts, and thus more useful in real-world applications. Another requirement dictated by real-world applications, is to develop methods that can use, when available, any domain-specific background knowledge. In the case of community detection the background knowledge could be a vague description of the communities sought in a specific application, or some prototypical nodes (e.g., good customers in the past), that represent what the analyst is looking for (a community of similar users).

Towards this goal, in this article we define and study the problem of finding a diverse set of cohesive communities with concise descriptions. We propose an effective algorithm that alternates between two phases: a hill-climbing phase producing (possibly overlapping) communities, and a description induction phase which uses techniques from supervised pattern set mining. Our framework has the nice feature of being able to build well-described cohesive communities starting from any given description or seed set of nodes, which makes it very flexible and easily applicable in real-world applications.

Our experimental evaluation confirms that the proposed method discovers cohesive communities with concise descriptions in realistic and large online social networks such as DELICIOUS, FLICKR, and LASTFM.

Categories and Subject Descriptors: H.4.3 [**Information Systems Applications**]: Communications Applications; H.2.8 [**Database Applications**]: Data Mining

General Terms: Algorithms, Experimentation, Theory

Additional Key Words and Phrases: Social networks, community detection, domain knowledge, description, social patterns, behavioral and demographic information.

**ACM Reference Format:**
Pool, S., Bonchi, F., van Leeuwen, M. 2013. Description-driven Community Detection. ACM Trans. Intell. Syst. Technol. V, N, Article A (January YYYY), 28 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

Author's addresses: S. Pool, Department of Information and Computing Sciences, Universiteit Utrecht, the Netherlands; F. Bonchi, Yahoo! Research, Barcelona, Spain; M. van Leeuwen (corresponding author), KU Leuven, Department of Computer Science, Celestijnenlaan 200a - bus 2402, 3001 Heverlee, Belgium; email: matthijs.vanleeuwen@cs.kuleuven.be.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

## 1. INTRODUCTION

One of the most prominent features of social and biological networks is the presence of communities, i.e., the organization of vertices in modules, with an high level of connectivity inside the modules and low connectivity among modules. As such, developing algorithms to detect communities in large networks has attracted the interest of physicists, sociologists, and more recently computer scientists (see [Fortunato 2010] for an extensive survey).

While there has been a tremendous effort in devising *community detection* algorithms, surprisingly very little has been done to make sense of the communities found. As stated by [Fortunato 2010], this is by far the most important open problem in the field. In fact, in the traditional and most studied setting, the input is just a social graph and the output is a *graph partitioning*. That is to say that nodes can belong to one and only one cluster, and no additional information is considered beyond the graph structure. However, with the advent of online social networking sites, richer data has become available: beyond the link information, each user in the network is annotated with additional information, e.g., demographical information, shopping behavior, or interests. As a consequence, various researchers have recently been relaxing one of these constraints, allowing their algorithms to exploit additional information and to detect overlapping communities. Our work collocates in this literature. In particular our work is driven by the following questions:

(1) *How can we use the available additional information to describe and understand the communities that we find?*
(2) *Can we exploit this information during the community detection process, in order to find more meaningful communities?*
(3) *When domain knowledge is available in terms of descriptions of communities of interest, can we let the mining process be driven by these descriptions?*

### 1.1. Motivations and objectives

It is well known that one of the most challenging tasks in data mining is to produce models which are not only accurate, but which are also *explainable* and *meaningful* to the domain experts. Indeed, data mining practitioners are aware of the fact that presenting a good machine learned model to a biologist or to a marketing specialist is not enough. Domain experts know their job and they want to understand why the model behaves the way it behaves. *They want to understand the model*.

That's not the whole story. Often the domain expert has some preconceived idea on the kind of patterns that the data mining model should produce, which is usually based on her knowledge of the application domain. Developing data mining methods that can keep in consideration the expert background knowledge of the domain, exploiting it to produce better models, has been one of the most important research problems since the early days of data mining.

In this paper we study the problem of finding communities that have "good" descriptions. In particular, we are given a social network consisting of 1) the social graph, and 2) auxiliary vertex information, in the form of attribute-value pairs assigned to each vertex in the graph. The goal is to find communities of vertices which are *cohesive* in the social graph, and have a *concise* description in the vertices' attribute space.

The basic assumption behind our goal is that *homophily* holds in the social network. Homophily, i.e. "love of the same", is the concept according to which individuals tend to associate and bond with similar others; where similar might be defined, e.g. on the basis of age, gender, class, organizational role, and so forth. This is often expressed by the adage "Birds of a feather flock together", and it has been observed by hundred sociological studies [McPherson et al. 2001]. If homophily holds in the data, then the users

of the social network are likely to be close (in a graph-distance sense) to users that are similar (w.r.t. vertex attributes). If this is the case, then we might succeed in our goal of finding cohesive communities with concise and meaningful, hence understandable, descriptions.

### 1.2. Application domain examples

The method we propose in this article is able to start either from some seed communities, or from some given descriptions. Our method keeps alternating between two phases, 1) maximizing community quality in the graph space, 2) inducing a description that matches the community as good as possible, and reshapes the community if no perfect match is found. In that case, another iteration of both phases follows. The ability to build a well-described cohesive community starting from either a given description or community is one of the main distinguishing features of our proposal. This feature makes our framework very flexible and applicable in real-world application scenarios as described next.

**Behavioral social targeting.** *Behavioral targeting*[1] uses information collected from an individual's web-browsing behavior, such as the pages they have visited or the searches they have made, to select which advertisements to display to that individual. Practitioners believe this helps them deliver their online advertisements to the users who are most likely to be interested. This is often done in conjunction with other forms of targeting, based on factors like geography, demographics or, when available, social context.

Indeed, conjoining behavioral and *social targeting* is an appealing perspective: on the one hand, one might want to target users similar to users that reacted well to a similar campaign in the past, on the other hand one might assume homophily, and target users that are socially close to these users. However, as discussed before, the marketing or advertising practitioners not only want to know the communities as a list of vertices, but they want to have a description of these communities, e.g. "30 to 35, east coast, Starbucks fans", or "20 to 30, with an iPhone, like blockbuster movies".

Moreover, a customer might have a high level description of few prototypical profiles of users that should be targeted by a campaign. Those prototypical profiles are nothing more than our descriptions, that can be used as seeds to start the community detection process. For instance, from the description "20 to 30, with an iPhone or with an iPad" a complete community and associated refined description could be mined.

Similarly, if a company knows which users where successfully targeted in previous campaigns, it could use these users as seeds in order to grow larger communities of potential customers.

**Recommender systems in social media consumption platforms.** Recommender systems are a powerful tool in social platforms for media consumption, such as Flixster[2] for movies, or Spotify[3] for music. In this context, the traditional collaborative-filtering based recommenders, which consider only user similarity defined on the basis of their past ratings, overlook a very important piece of information: the social network. Indeed, in a recent poll[4], 94% of the sample of Spotify users interviewed confirmed that they *"listen to a song because they saw a friend listening to it"*. This motivates the need for *community-aware recommender systems*. In our setting we can

[1] http://en.wikipedia.org/wiki/Behavioral_targeting
[2] http://www.flixster.com/
[3] http://www.spotify.com/
[4] http://mashable.com/2012/08/12/social-music-listening/

consider ratings or the number of plays as the users' attributes, and thus a community description represents what the users in that community like or dislike.

A user in these social media platforms typically belongs to more than one community: by presenting to him the various communities he belongs to, and which items are particularly popular in each community, we can give more effective recommendations. In fact, the user can easily collocate a new item in his own mental map of interests, thanks to the description of the community in which it is popular. The user can also relate such an item to a compact neighborhood of users, that goes beyond the distance-1 direct friends, but still remains in a close-knit, thus trustable, community of friends-of-friends.

Community-awareness can also help recommender systems to deal with the typical *cold-start* problem of collaborative filtering. Consider a new item that has not yet received any rating. We can create a short description of the item based on similar items, and then we can use this description as seed to grow communities around it: these communities can then receive the item and its description as recommendation.

### 1.3. Summary of contributions and article overview

To properly position this work relative to existing work, we discuss related work directly after the Introduction in Section 2. We constrain ourselves to discussing the work that is most relevant to ours, which we divide into two groups. The first group concerns methods for detecting overlapping communities, and the second concerns approaches that take more information than just the graph into account.

Subsequently, Section 3 studies the problem of description-driven community detection. Contrary to existing work, we propose a very *expressive query language* for our descriptions, i.e., disjunctions of (possible negated) conjunctions of constraints on the attribute data. With this rich language, we enable the description of both large and cohesive communities in terms of auxiliary user information even when homophily is not very distinctly present in the data. This is very likely to be the case for sparse data types such as tag data.

Furthermore, to quantify the cohesiveness of a community in the social graph, we propose an intuitive *community score* that is based on counting erroneous links, i.e. user relations that are either missing or obsolete with respect to the 'ideal' community given this subgraph. This score has a number of advantages. First, it is fast to compute and an elegant hill-climbing algorithm that finds a (local) optimum can easily be devised. Second, larger communities can potentially attain larger scores, which is in accordance with our goal to rank larger communities higher. Third and last, each community can be scored individually, independent of others, which not only facilitates paralellization but also ensures that identified communities can be overlapping.

To ensure concise descriptions, our aim is to induce queries that minimize *description complexity*. The last requirement is that all resulting communities should be substantially diverse. Based on these foundations and requirements, we formalize the *Diverse Top-k Descriptive Community Mining* problem.

Section 4 presents the heuristic *DCM algorithm*, which efficiently approximates the solution of this problem. It achieves this by starting from a set of candidate communities and improving these by alternating between two phases. The first phase consists of a fast hill-climbing algorithm that adds/removes vertices to/from a community, such that its community score is maximized. The second phase induces a concise description that matches the community as accurately as possible, and reshapes the community if no perfect match is found. In that case, another iteration of both phases follows.

A main advantage of the proposed two-phase algorithm is that a well-described cohesive community can be obtained from *any* given description or community. For the marketing/targeting and recommendation examples just mentioned above, this offers

numerous possibilities. Also note that it is very easy and efficient to let communities 'co-evolve' with the social network; simply use previously mined communities or descriptions as input, and let them adapt to the newly evolved network.

We report on extensive *experiments* in Section 5, performed on datasets obtained from three online social networks: DELICIOUS, FLICKR, and LASTFM. Both the quantitative results and the presented example communities confirm that the proposed method discovers cohesive communities with concise descriptions. Section 6 concludes the paper.

## 2. RELATED WORK

Communities have a long history in social sciences, but it was in 2002 that the seminal paper [Girvan and Newman 2002] triggered a lot of interest on the problem of *community detection*, which has since then been extensively studied, mainly in physics and computer science literature. Until recently most of such literature, of which Fortunato's survey [Fortunato 2010] provides a comprehensive coverage, has focussed on finding disjoint communities in simple graphs. That is to say that nodes can belong to one and only one cluster and no additional information is considered beyond the graph structure. In the following we review various proposals that have dropped these assumptions.

### 2.1. Overlapping communities

Nowadays it is widely understood and accepted that people in social networks rarely belong to only one community: for instance the same individual usually has family, friends, colleagues and several interest-based affiliations. This idea has also been explicitly implemented in `Google+` "circles" and `Facebook` "smart lists". Consequently, the study of overlapping community detection has received a growing amount of attention in the last years, and many new algorithms have been proposed.

A recent survey [Xie et al. 2013] categorizes algorithms for overlapping community detection in various classes: methods based on *clique percolation* [Palla et al. 2005; Palla et al. 2007; Kumpula et al. 2008]; methods that extend the idea of *label propagation* [Raghavan et al. 2007] to produce overlapping communities [Gregory 2010; Xie et al. 2011; Xie and Szymanski 2012]; agent-based and particle-based models [Chen et al. 2010; Breve et al. 2011]; methods based on local expansion and optimization [Baumes et al. 2005; Lancichinetti et al. 2009; Lancichinetti et al. 2011; Padrol-Sureda et al. 2010]. The two classes that are most relevant to our proposal are *link partitioning* methods and *stochastic generative* models, discussed next.

Link partitioning has recently gained popularity [Ahn et al. 2010; Evans and Lambiotte 2009]. Clustering links instead of nodes is a very appealing approach to obtain overlapping communities: it is simple, more understandable, and more realistic than simply having a soft (or fuzzy) assignment of nodes to communities. In fact, as highlighted before, while for nodes it is natural to belong to more than one community, links are usually explainable by co-affiliation to some topic/community (as in the famous *affiliation networks* [Lattanzi and Sivakumar 2009] model). [Evans and Lambiotte 2009] use the idea of applying normal node partitioning to the *line graph* of the given network, in order to obtain a link partitioning in the original network. [Ahn et al. 2010] use a simple hierarchical clustering of the links, where similarity among two links incident in the same node is defined based on the Jaccard coefficient of the neighborhoods of the other two nodes. [Kim and Jeong 2011] extend the *Infomap* method to the line graph which encodes the path of the random walk on the line network using the Minimum Description Length (MDL) principle.

While these methods are based on heuristic quality functions, in recent years approaches based on fitting generative models to the data have emerged. [Airoldi et al.

2008] introduce the *mixed membership block model*: this technique factorizes the adjacency matrix in a low dimensional space expressing patterns of directed social relationships between blocks of vertices. According to the generative process, for each pair of nodes group membership is sampled for both the source and the destination: the link is generated by sampling from the binomial distribution which encodes the probability of observing a directed connection between the considered groups. Since membership assignments are drawn independently for each possible link, users can belong to multiple groups.

A different generative model is proposed in [Ball et al. 2011]. The basic assumption is the existence of $n \times k$ parameters, where $n$ is the number of nodes and $k$ the number of communities, which specify the propensities of each vertex to have links of each possible label. The number of links of label $z$ between two vertices is then assumed to be distributed according to a Poisson distribution, parameterized by the product of the two vertex-label components.

### 2.2. When additional information is available

Most of the literature on community detection focuses on finding (either disjoint or overlapping) groups of nodes from a given graph. However, in online social networking sites, richer data is available. Beyond the mere link information, users might be annotated with, e.g., demographical information, shopping behavior, interests, tags and so on. Also the links might be labeled with a relationship type: e.g., family, friend, colleague and so on. Similarly nodes and links in biological networks are typically labeled with additional information. Recently researchers (mainly in the data mining community) have started proposing methods to discover communities in node-attributed graphs and in edge-labeled graphs.

[Wang et al. 2010] study the problem of discovering overlapping groups in social media. They propose a co-clustering framework based on users and tags. Users are not connected through a social network, instead they are implicitly connected by their common interests, as expressed by the tags they use. The method creates co-clusters of tags and users.

[Khan et al. 2010] introduce the concept of proximity patterns. Frequent patterns, i.e., patterns occurring frequently in the database, are a classic concept in data mining. Proximity patterns try to blur the boundary between the graph and description data. Proximity patterns do not involve only the attributes of a node, but instead they can be based, e.g., on one attribute of a vertex $u$, and one attribute of a vertex $v$, as long as an edge $(u, v)$ exists. Khan et al. develop a model called *Nearest Probabilistic Association* to define the frequency of a proximity pattern. This approach transforms the problem of finding communities in graph and attribute data into a traditional frequent pattern mining problem for which many efficient methods exist.

[Silva et al. 2010] study structural correlation patterns, which are dense subgraphs induced by a particular attribute set. Also their proposal combines aspects of frequent itemset mining and dense subgraph extraction. In particular, as dense subgraphs they propose to use $\gamma$-*quasi-cliques*. A $\gamma$-quasi-clique, with $0 < \gamma \leq 1$, is a set of vertices $V$ where every vertex $v \in V$ is connected to at least $\gamma(|V| - 1)$ vertices. The authors introduce the *SCORP* algorithm, that starts with mining all frequent item sets for all users. Then, for each frequent itemset it selects the set of vertices containing all items, it finds all quasi-cliques in this set, and tries to adjust the set of items selecting these vertices. Silva et al. also propose statistical significance measures that compare the structural correlation of attribute sets against their expected values using null models.

[Moser et al. 2009] introduce the problem of *finding cohesive patterns*. A cohesive pattern is defined as a connected subgraph whose density exceeds a given threshold.

Furthermore a cohesive pattern has, in a large enough subspace, homogeneous feature values. Integrating constraints into the frequent itemset mining problem reduces the number of patterns substantially. They developed an algorithm called *CoPaM* that efficiently finds the set of all maximal cohesive patterns.

[Zhou et al. 2009] use node attributes to augment the social graph by generating "attribute edges" between nodes that are similar on a given attribute. Then they mine communities in this augmented graph, using the neighborhood random walk model to estimate vertex closeness on the augmented graph.

Some other authors consider the case when additional information can be associated to the edges. In these cases, the information available beyond the social graph is usually materialized by creating multiple networks, and then there are essentially two ways of proceeding: 1) keeping the various networks separated, while trying to compare or combine the communities found in the various networks; 2) analyzing a single (possibly weighted) graph, obtained by combining the contributions of the different types of edges in some way.

The former approach is followed by [Tang et al. 2009]. They discuss two straightforward extensions of the modularity quality measure for the case of multidimensional networks: average modularity maximization (AMM) and total modularity maximization (TMM). The authors show that both methods are not robust enough to handle networks with noise, and therefore – inspired by PCA – they propose *Principal modularity maximization* (PMM) to overcome this limitation. The idea is to extract structural features from each dimension of the network, which also effectively removes the noise in that dimension. After that, cross-dimension analysis on the constructed data is applied to find the lower-dimensional embedding, such that the features extracted from all the dimensions are highly correlated to each other.

The latter approach is followed by [Atzmueller and Mitzlaff 2011]; they flatten multiple graphs into a single graph. In particular they work on data from the BibSonomy system. They merge the Friend-Graph with the Click-Graph, which contains an edge $(u, v)$ whenever $u$ clicked on a link contained in the user page of user $v$, and the Visit-Graph, where an edge $(u, v)$ is created if $u$ simply visited the user page of user $v$. This results in a single graph that is used for community mining. In addition to these multiple graphs, they also use tag information to obtain community descriptions. Because tag data is sparse, they use latent dirichlet allocation (LDA) to build topics, and assign topics to users (instead of, but based on, the tags that a user used). Using these topics as descriptive features, a standard subgroup discovery search is adopted to find good communities w.r.t. a local community quality measure, such as *local modularity* and *inverse conductance*.

[Bonchi et al. 2012] study the case in which each edge has one and only one label, e.g., `family`, `friend`, or `colleague`. They study the clustering problem of finding groups of nodes in such a way that the edges within the groups are as much as possible of the same kind.

[Barbieri et al. 2013] propose a generative stochastic model to detect communities from the social graph and a database of information propagations over the social graph.

Although there has been some work considering how to exploit additional information for community detection, no previous work offers the flexibility of the framework we propose in this paper, that is the possibility of building well-described and cohesive community starting either from a given description or a given seed community. As we discussed in the introduction this flexibility, that allows to inject background domain knowledge into the mining process, is advocated by applications and by domain experts. Another distinguishing feature of our framework, again motivated by the applications, is that we do not look for a complete coverage of the whole social graph with

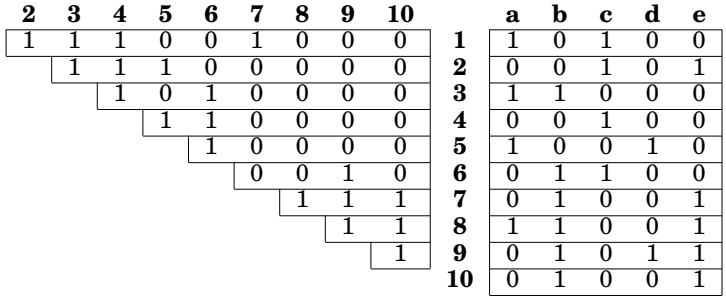



Fig. 1. Example attributed graph with 10 vertices (1–10) and 5 associated binary attributes (a–e).

(disjoint or overlapping) communities as most of the previous literature, instead we look only for few communities of interest, as we formalize in the next section.

## 3. PROBLEM DEFINITION

As stated in the introduction, the social networks we consider consist of 1) the social graph, and 2) auxiliary information, in the form of attribute-value pairs assigned to each vertex in the graph. More formally, we are given an *attributed graph* $G = (V, E, A)$, where $V$ is the set of vertices, $E \subseteq V \times V$ is the set of undirected edges[5], and $A = \{a_1, \ldots, a_l\}$ is a set of $l$ attributes. Without loss of generality we consider the case of binary attributes. Therefore $A$ is a binary matrix and each vertex $v \in V$ is associated with a binary vector $A(v) = [a_1(v), \ldots, a_l(v)]$. We denote the number of vertices by $N = |V|$, the number of edges by $M = |E|$.

In our model a community is identified by means of a query over the attribute matrix $A$. A query $Q$ is a logical formula consisting of conditions on the graph's attribute vectors. The exact form of the query language allowed in our framework will be discussed in Subsection 3.2.

We denote the universe of all possible queries over $A$ as $\mathbb{Q}_A$, and the size of a query $Q$, i.e., the number of constraints it contains, by $|Q|$. Given an attributed graph $G = (V, E, A)$ we define the function $g : \mathbb{Q}_A \to 2^V$, associating to each query $Q \in \mathbb{Q}_A$ the set of vertices which satisfy $Q$:

$$g(Q) = \{v \in V \mid \ A(v) \models Q\}.$$

Similarly we define the function $f : 2^V \to \mathbb{Q}_A$, associating a unique query to each set of vertices $C \subseteq V$. In particular, $f(C)$ is *the most concise query* satisfied by all vertices in $C$. That is, $f(C) = Q$ s.t. $g(Q) = C$ and there is no $Q' \in \mathbb{Q}_A$ other then $Q$ for which $g(Q') = C$ and $|Q'| < |Q|$. In other words, $f$ returns the shortest possible query that implies the given set of vertices.

Finally, it might happen that $Q$ is empty ($|Q| = 0$). We denote this special query with $\perp$: this is the most general query possible, with no constraints, and thus satisfied by any vertex.

*Example* 3.1. Figures 1 and 2 present an example attributed graph $G = (V, E, A)$ with $N = 10$ vertices, $M = 18$ edges, and $l = 5$ binary attributes.

Consider the set of vertices $U = \{7, 8, 9, 10\}$. This set of vertices is totally connected (a clique) and hardly connected to the rest of the network. Moreover $f(U) = b \wedge e$

[5]We consider undirected graphs for the sake of presentation, and consistent with most literature. However, we note that the generalization to directed graphs is straightforward.

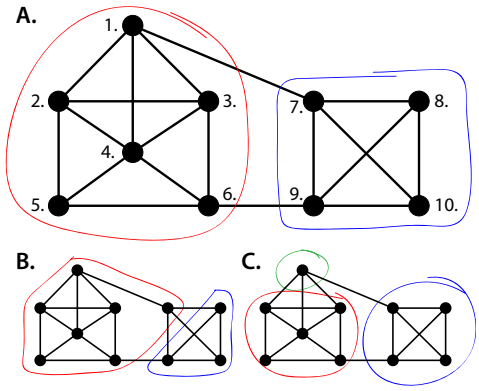



Fig. 2. Three different partitionings of the example graph depicted in Figure 1.

describes exactly all and only the nodes in $U$, i.e., $g(b \wedge e) = U$. Intuitively, this is a perfect example of a coherent community with a good description.

Consider now $T = \{1, 2, 3, 4, 5, 6\}$: intuitively, this is a good community too, at least in the graph space. However, when we take the most specific query satisfied by $T$, it turns out that $f(T) = \bot$. Obviously this is not a good description as $g(\bot) = V$. However we can note that the query $\neg b$ describes perfectly a subset of $T$ as $g(\neg b) = \{1, 2, 4, 5\}$. Another alternative is the query $\neg d$ which captures all but one elements of $T$, but it also describes one element not in $T$, i.e., $g(\neg d) = \{1, 2, 3, 4, 6, 9\}$.

In the example above we have seen that there simply is not always a good description corresponding to a good community in the graph space. Moreover, we have seen that queries can make different kinds of errors while describing a community: they might lose some elements of the community, or they might capture also elements which are not part of the community.

Hopefully, if homophily holds in the data, we will be able to find good communities with good descriptions. The first step, is to formally define what makes a community and a description "good".

### 3.1. Community Score

We next define our quality measure for individual communities. In order to produce this definition, we start from a global quality score for graph partitioning, and then we transform it to make it local to a single community. A graph partitioning for $G = (V, E)$ splits the graph into a set of non-overlapping communities $\mathcal{C} = \{C_1, \ldots, C_N\}$, where a community is a subset of the vertices. That is, $\bigcup_i C_i = V$, and $C_i \cap C_j = \emptyset$ for all $i, j \in [1, N]$ with $i \neq j$. Note that we do not consider queries and attribute vectors here, but only the graph structure as represented by $V$ and $E$.

As usual a graph partitioning can be regarded to be high-quality when it satisfies the following objectives: 1) the large majority of vertices *within* each community are directly linked by edges, and 2) there are hardly any edges *between* the communities. The simplest and neatest formalization of such intuition into a quality measure is to directly count the edges that are either missing or present where this shouldn't be the case.

For this, we consider two corresponding types of errors. The first error type counts the number of missing edges between vertices within a community $C$:

$$\epsilon_{within}(C, E) = |\{(v, w) \mid v, w \in C \wedge v \neq w \wedge (v, w) \notin E\}|$$

The second error type counts the number of edges between a given community $C$ and all other communities:

$$\epsilon_{between}(C, E) = |\{(v, w) \in E \mid v \in C \wedge w \notin C\}|$$

Together, this gives a global error measure. Given a graph partitioning $\mathcal{C}$ and graph $G$, *link error* $\epsilon$ is defined as:

$$\epsilon(\mathcal{C}, G) = \sum_{C \in \mathcal{C}} \epsilon_{within}(C, E) + \frac{\epsilon_{between}(C, E)}{2}$$

Note that since all errors between the communities are counted twice, once for each community, we divide this error by two. The globally optimal partitioning would be that partitioning that results in the *minimum link error*.

It is worth noting that this corresponds to a well-studied problem in theoretical computer science, known as the CLUSTER EDITING problem [Shamir et al. 2004] or CORRELATION CLUSTERING [Bansal et al. 2004]. The CLUSTER EDITING problem is defined as follows. Given an undirected graph $G$, and a non-negative integer $k$, can we transform $G$, by inserting and deleting at most $k$ edges, into a graph that consists of a disjoint union of cliques? The *cluster edit distance* for a graph $G$ is the smallest $k$ for which cluster editing is possible. In our terminology this corresponds to the best possible community score (i.e., the minimum link error) that we can achieve on a given graph $G$.

*Example* 3.2. Figure 2 shows three example graph partitionings, and their corresponding link errors are: 6 for A, 13 for B and 7 for C. Treating the entire graph as a single community would give an error of 26, considering each vertex as an independent community results in $\epsilon = 18$ (i.e., the number of edges $M$). Since lower link errors are better, Figure 2A contains the best partitioning, which is in accordance with our intuition when looking at the graph.

We next proceed to adapt minimum link error to a local function, as we want to evaluate communities on the basis of their own merits, and we are not interested in whole graph partitioning. On the contrary, we want to find some good communities that do not necessarily cover all vertices in the graph. Moreover, the communities might be overlapping.

The viewpoint we take is the situation where each vertex in a graph forms its own community, resulting in $N$ communities. Assuming that graph $G$ is sparse, this can be considered as a baseline partitioning, where each edge results in a *between-communities* error. For any given community $C \subseteq V$, this baseline error can be defined as:

$$\epsilon_{base}(C, E) = \sum_{v \in C} \epsilon_{between}(\{v\}, E)$$

Now, we would like to measure how much $C$ improves on this baseline, by counting the decrease in the number of errors. That is, we define the community score $S$ for a community $C$ and graph $G = (V, E)$ as follows:

$$S(C, G) = \epsilon_{base}(C, E) - \epsilon_{within}(C, E) - \epsilon_{between}(C, E)$$

In other words, we count the reduction in the number of errors by merging all *independent vertices* into a *single community*. The larger this reduction, the better. Note that negative values are possible, but this means that the community is really bad; treating each vertex as a single community would give a higher score. In what follows, we will sometimes omit $G$ and simply write $S(C)$.

A major distinctive feature of this community score over other measures such as modularity [Newman and Girvan 2004; Newman 2004], is that it is based only on counting errors, and these errors provide us all information needed to directly optimize a community. This allows our communities to be evaluated on their own merits (thus locally) and not in the context of a complete non-overlapping partitioning (thus globally), as assumed by modularity. Moreover, it should be noted that, since we compare a community $C$ to the situation where each vertex in $C$ forms a community by itself, larger communities potentially get higher scores than smaller communities, which is in accordance with our high-level goals.

*Example* 3.3. Let us consider the individual communities depicted in the graphs in Figure 2. We previously observed that the graph partitioning in Fig. 2A gives the minimum link error. We now observe that its individual communities also get the highest community scores: $S(\{1,\ldots,6\},G) = 18$ and $S(\{7,\ldots,10\},G) = 12$.

Let us have a close look at the computation of $S(\{1,\ldots,6\})$. The definition of $S(C,G)$ shows that we need to compute three terms. The first term, $\epsilon_{base}$, is the sum of the number of edges of each individual vertex in the community, i.e., $4{+}4{+}4{+}5{+}3{+}4 = 24$. The second term, $\epsilon_{within}$, is the number of missing edges within the community that prevent it from being a clique, i.e., $4$. Finally, $\epsilon_{between}$ is the number of edges that connect to any vertex that is not part of the community, $2$ in this case. This means that we have $S(\{1,\ldots,6\}) = 24 - 4 - 2 = 18$.

Small modifications to the communities achieving the maximum scores mentioned above result in lower scores. For example, if we consider the situation in Fig. 2B, we obtain $S(\{1,\ldots,7\}) = 15$ and $S(\{8,\ldots,10\}) = 6$. A single vertex can never be a high-scoring community, as its score is always 0, e.g. $S(\{1\}) = 0$.

Having defined the quality measure for communities, we can define the problem of discovering, from an attributed graph, the top-$k$ queries w.r.t. the scores of their corresponding communities.

PROBLEM 1. *Given an attributed graph $G = (V,E,A)$ and an integer $k$, find a set of $k$ queries $\mathcal{Q} \subset \mathbb{Q}_A$, such that no two queries $U \notin \mathcal{Q}$ and $Q \in \mathcal{Q}$ exist for which $S(g(U),G) > S(g(Q),G)$.*

Note that in practice, there may be high-scoring communities $C$ for which no corresponding query $Q \in \mathcal{Q}$ exists such that $g(Q) = C$. That is, rigid associations between queries and communities may not always exist, as homophily may not hold for all high-scoring communities. However, we are strictly interested in communities that have matching descriptions. To make interpretation possible, however, we prefer concise descriptions over lengthy ones. We next define the type of descriptions we allow in our model and we establish how to quantify their conciseness.

### 3.2. Description Language

Given the binary attribute matrix $A$, we build descriptions (or queries) starting from basic conditions of the form $a_i = 1$. Since each condition is of this form, we abbreviate this by simply denoting $a_i$. The queries we consider in this article are disjunctions of conjunctions over such conditions, and each of these conjunctions may contain negations, e.g., $(a_1 \wedge a_2) \vee \neg(a_3 \wedge a_4)$, $(a_1 \wedge a_2) \vee (\neg a_3 \wedge a_4)$, and $(a_1 \wedge a_2) \vee (a_3 \wedge \neg(a_4 \wedge a_5))$ are all valid queries. Note that e.g. the first example could also be written as $(a_1 \wedge a_2) \vee \neg a_3 \vee \neg a_4$. This alternative format would probably be easier to interpret by a domain expert, but since we induce the queries in the described form we refrain from this translation in the current paper for purposes of analysis.

Note that we opt for an expressive and therefore relatively complex query language, which makes our general framework suitable for any attribute data. In particular, this

language will be very useful in dealing with the very sparse data that we consider in this paper; in Section 3.5 we will argue why this is required, and in the experiment section we will empirically show that we indeed need these complex queries for tag data. However, depending on the goals of the user and the data at hand, a subset of this rich query language might be preferred, e.g. leaving out disjunctions and negations, and considering only conjunctions. This is possible within our framework and does not change the overall approach that we advocate, but it may be required to substitute the method that we use for query induction (see Subsection 4.4).

### 3.3. Description Complexity

When presenting a community description to a domain expert, it should not be overly complex. However, measuring complexity of a query can be done in several ways. The most obvious route seems to be based on the intuition that 'shorter is better', i.e., we could count the total number of constraints contained by a query and try to minimize this. This is not necessarily the best choice though, as multiple copies of the same constraint naturally occur in the expressive query language that we consider.

We observed that having the same constraint, or different constraints involving the same attribute, in several conjunctions of a disjunction does not give the impression of the query being much more complex. On the contrary, adding more constraints involving *different* attributes does make interpretation harder. We exploit these observations in the following definition of description complexity.

*Definition* 3.4. Given a query $Q$, define its description complexity $D$ as the number of (unique) attributes it contains:

$$D(Q) = |\{a_i \in A \mid a_i \text{ occurs in } Q\}|,$$

where an attribute *occurs* in $Q$ if it is used in at least one condition in the query.

Obviously, the goal is now to find queries such that their corresponding communities have high community scores $S$ and low description complexities $D$. To facilitate ranking of the results, we combine these two scores into a single overall objective score $\sigma$, which is defined as community score divided by description complexity. Although one could obviously combine the scores in different ways, we opted for this definition because it balances the two nicely, allowing larger communities (with larger community scores) to have slightly more complex descriptions. Preliminary experiments confirmed that this natural choice works well in practice.

Adding description complexity to the equation results in the following modified problem statement.

PROBLEM 2. *Given an attributed graph $G = (V, E, A)$ and an integer $k$, find the top-$k$ community-query pairs $(C, Q)$, where $C \subseteq V$, $Q \in \mathbb{Q}_A$, and top-$k$ is defined with respect to the overall score function $\sigma$:*

$$\sigma(C, Q) = \frac{S(C, G)}{D(Q)}$$

### 3.4. Diverse Descriptive Community Mining

The revised problem statement is a clear improvement over the initial one, as it includes both the community score and description conciseness. Together, this should give us concise queries that imply good communities in the graph. Still, there is one ingredient missing from our formalization: queries are only considered individually, which is likely to result in similar queries and communities.

Avoiding redundancy in the result set is essential to make interpretation by domain experts feasible. To achieve this, we consider the subgraphs implied by the queries.

That is, each unique community $C$ should be included only once in the result set, even if there are multiple queries of the same complexity that describe it. Taking this even one step further, we argue that all final communities should be sufficiently different from each other.

In particular, given two communities $C$ and $C'$ we consider a similarity measure $sim(C, C') \in [0, 1]$. Moreover, given a maximum similarity threshold $\eta \in [0, 1]$, we do not allow both communities to belong to the result set if $sim(C, C') > \eta$. Adding this additional constraint to the previous problem statement, this leads to the following and final problem.

PROBLEM 3 (DIVERSE TOP-K DESCRIPTIVE COMMUNITY MINING). *Given an attributed graph $G = (V, E, A)$, an integer $k$ and parameter $\eta$, find the top-$k$ community-query pairs $(C, Q)$, where $C \subseteq V$, $Q \in \mathbb{Q}_A$, and top-$k$ is defined with respect to the overall score function $\sigma$:*

$$\sigma(C, Q) = \frac{S(C, G)}{D(Q)},$$

*such that $sim(C, C') \leq \eta$ for each two communities $C$ and $C'$ in the diverse top-$k$.*

To make things more concrete, as communities are sets of nodes in this paper we consider the Jaccard index as similarity measure. Given two communities $C$ and $C'$, the Jaccard index is computed as $J(C, C') = \frac{|C \cap C'|}{|C \cup C'|}$. Observe that the additional constraint implies that the resulting set of community-query pairs is strictly speaking not a 'top-$k$', as there may be pairs with higher score $\sigma$ not selected for the result set.

### 3.5. Discussion: search space and naïve approaches

The search space of Problem 3 is very large, and exhaustive search is infeasible for realistically sized datasets. In fact we need to jointly explore two spaces: the one of all communities (subgraphs), and the one of all descriptions. These two spaces are intrinsically exponential in the size of the data. Moreover, $\sigma(C, Q)$ has no usable properties for pruning, e.g., it is not monotonic with respect to either the subgraph or query. As a result, it is not feasible to enumerate all possible subgraphs and compute the corresponding lowest-complexity description, nor to enumerate all possible queries and compute the highest-scoring corresponding subgraph, nor to traverse the joint query-subgraph search space in an efficient way.

It might appear that a possible solution would be to consider simpler query languages. Such a language might even exhibit some monotonicity property, and therefore make smart traversal of the search space possible. For example, let us assume that we only allow conjunctions of attribute-value pairs, i.e., *itemsets*. Queries would thus be restricted to the form '$a_i = 1 \wedge a_j = 1$'. A first naïve approach could be to mine all frequent itemsets [Agrawal et al. 1993] from the attribute-value data, consider these as candidate queries, and compute $\sigma(C, Q)$ for each candidate query $Q$ such that $C$ consists of the set of nodes that contain $Q$. Unfortunately, in practice this approach suffers from three problems: 1) the set of frequent itemsets is usually large, and many itemsets are very similar and hence redundant; 2) due to a variety of reasons, attribute-value data in social networks is often very sparse and itemsets are not powerful enough to describe semantically meaningful communities; 3) although frequent itemsets provide a description, it is usually not a characteristic description that discriminates it from the rest of the social network. Similarly, frequent itemsets are combinations of attribute-values that often occur together, but in realistic datasets these are unlikely to satisfy the homophily requirement discussed before.

A second naïve approach would be to mine *emerging patterns* [Dong and Li 1999] given a subgraph $C$. In this case, emerging patterns are those itemsets that are (very) frequent within $C$, but (very) infrequent in its complement. Although this solves the third problem of the frequent itemset approach, i.e., it does provide discriminative descriptions, the other two problems remain. Furthermore, one would have to mine emerging patterns for each candidate subgraph $C$, which is a time-consuming effort, and then select/construct a suitable description from the large set of resulting patterns. In the next section we will also show empirically that simple query types lead to very inaccurate descriptions for subgraphs with high community scores.

Hence, we conclude that the query language needs to be more powerful for real-world applications, and we arrive at the much more expressive language that we use in this paper. This obviously comes with its own disadvantages: by allowing both disjunctions and negations, the description search space also becomes very large and enumerating and testing all possible queries becomes infeasible. Hence we resort to a heuristic search scheme that alternately optimizes the query and corresponding subgraph, as we explain in the next section.

## 4. THE ALGORITHM

In this section we introduce our heuristic algorithm that approximates the solution of the *Diverse Top-k Descriptive Community Mining* problem. Our iterative method is inspired by the *exceptional model mining* paradigm [Leman et al. 2008], and in particular by EMDM [M. van Leeuwen 2010], a generic algorithm for finding interesting subgroups in large datasets. The method we propose, dubbed DCM for 'Descriptive Community Mining', alternates between maximizing the community score and inducing a fitting concise description. We will first introduce the overall algorithm, after which we will explain the individual steps in more detail.

### 4.1. The DCM Algorithm

The pseudo-code of the overall DCM algorithm is given in Algorithm 1. It starts with a set of candidate communities, which can be obtained in different ways. We will discuss several strategies in the next subsection. Starting from each individual candidate community, the algorithm iteratively improves it until convergence (lines 2-6).

Each iteration consists of two important steps, where each step reshapes the community to optimize one of the two main criteria (i.e., community quality and description conciseness). To optimize the first criterion, we employ a novel and purely local hillclimbing procedure to transform the community such that its graph-based score is maximized (line 4). For the second criterion, a description induction scheme is used to induce a concise query, to which the community is adapted accordingly (line 5). Since both steps alter the community, they are alternately executed until the community no longer changes. Note that it is important to perform the equality check *after* performing the two steps, to avoid cycling through exactly the same actions forever. After converging, the query corresponding to the resulting community (denoted $Q(C)$) is added to the intermediary result set (line 6).

When all candidate communities have been individually processed, a simple post-selection scheme is performed to select a diverse top-$k$ communities (line 7). The aim of this step is to eliminate communities that are mostly overlapping, as motivated in Section 3. All communities are ordered according to their overall scores (descending), after which they are considered one by one for inclusion in the final result set. Each community is included in this set if the Jaccard index measured on the vertex sets between the candidate and all communities selected so far is no larger than the given parameter $\eta$. In other words, communities that share too many vertices with a selected higher-score community are eliminated.

**ALGORITHM 1:** DCM

**Input:** Attributed graph $G$, parameters $k$ and $\eta$, and a set of candidate communities $\mathcal{C}$.
**Output:** An approximation of $\mathcal{Q}$, the top-$k$ communities.

```
1. Q ← ∅
2. for all C ∈ C do
3.    while C changes do
4.       C ← MAXIMIZE_COMMUNITY_SCORE(C)
5.       C ← FIND_CONCISE_QUERY(C)
6.    end while
7.    Q ← Q ∪ Q(C)
8. end for
9. Q ← SELECT_DIVERSE_TOP_K(Q, k, η)
10. return Q
```

### 4.2. Candidate communities

The DCM algorithm presented in the previous subsection starts from a set of candidate communities and iteratively improves each of these. Hence, it is important to start with a suitable set of candidates. As discussed earlier, one of the main advantages of our method is the ability of growing communities starting from small seeds of nodes, even from a single node, or from preliminary descriptions. We next discuss several approaches to obtaining initial candidates, based on what type of a-priori knowledge is available.

*Domain knowledge available.* In many situations there is already background information available on the communities that one is interested in: such knowledge can be expressed as one or more queries, representing an initial description of the communities of interest. Consider for example the targeting application that we mentioned in the Introduction. In this case, there might be user profiles available that were constructed from, e.g., users that reacted well to previous campaigns. These user profiles could then be converted to queries. Alternatively, a domain expert might have other knowledge that can be directly translated to queries. Given a set of queries, or even just a single query, these can be applied to the current dataset. Each individual query results in a (partial) candidate community that can serve as input to the DCM algorithm.

*Partial communities available.* In other situations, it might be that no descriptive information about the target communities is known. However, there could be information concerning specific nodes, or sets of nodes, that are expected to be part of a community. All nodes or node sets can be directly used as seed candidates for the algorithm. For example, the algorithm may have been run on the network in a previous state, and the domain expert aims to have the changes in the network reflected in (some of) the previously identified communities. Or, as a second example, we could use the set of users that reacted well to a previous campaign as input, without using their profiles to generalize.

*No a-priori knowledge available.* When there is no background information available, our aim is to identify a diverse set of (possibly overlapping) communities that covers a substantial part of the network. To find such a diverse set of communities, the simplest and most naïve approach would be to consider each individual vertex $v \in V$ as a candidate community. However, for large graphs this would require testing a very large amount of candidates, of which many are likely to result in (almost) identical communities.

To avoid this problem, we instead select a limited set of candidates that are uniformly distributed on the graph. The candidate set is limited by imposing a minimum geodesic distance $p$ between each pair of candidates. From all vertices $V$, we pick a maximum subset $\mathcal{C} \subset V$ such that the shortest path between each two vertices $v_i, v_j \in \mathcal{C}$ (with $v_i \neq v_j$) consists of at least $p$ edges. By varying the minimum distance parameter $p$, the number of candidates can be restricted (and therefore the runtime of the algorithm), while hopefully not giving in on the quality and diversity of the final resulting communities. The experiments will have to show how well this approach works in practice.

The procedure just described gives us candidate communities of a single vertex. However, since *all* (connected) vertex pairs have community score 2, starting from individual vertices would require randomly adding a second vertex, which may be suboptimal. Hence, we use the attribute vectors to make informed guesses about the best vertex to add to the initial candidates, as follows. We use a distance function $d$ on the vertices' attribute vectors to compute the distance to all neighbours of a candidate vertex $v$, and then form a pair of $v$ together with its nearest neighbour $NN$, i.e.

$$NN(v, G) = \underset{w \in V \text{ s.t. } (v,w) \in E}{\operatorname{argmin}} d(v, w).$$

As distance function $d$ we use the complement of the Jaccard index, because it is well suited for the binary tag data we consider. Each $\{v\} \in \mathcal{C}$ is replaced by $\{v, NN(v, G)\}$ to form the final candidate set.

### 4.3. Finding Cohesive Communities

As explained earlier in this section, the main part of the DCM algorithm consists of two alternating phases. This subsection describes the first of these two phases, which transforms the graph corresponding to a community such that its community score is maximized. For this, we introduce a hillclimbing method, which is summarized in Algorithm 2. $\Delta S(a)$ denotes the relative change in $S(C, G)$ caused by applying modification $a$.

MAXIMIZE_COMMUNITY_SCORE is a greedy hill-climbing algorithm that considers, in each step, all possible actions that either ADD or REMOVE a single vertex, and chooses the one that maximizes $S(C, G)$. Hence, this procedure considers only the *structure* of the graph, i.e., not the attribute data. The algorithm starts by constructing a list of all possible modifications, i.e., all vertices that are currently in $C$ can be REMOVEd (line 2), and all vertices that are not in $C$ but have a direct neighbor in $C$ can be ADDed (line 3). Considering only direct neighbors implies that we only consider additions that keep the community connected; unconnected communities are hardly communities and therefore undesirable. From all possible actions, we choose the best one (line 4), i.e., the one that maximizes the change in community score $\Delta S$, which we apply only if it improves the score (lines 5-6). That is, we iteratively select that action that results in the largest increase in score, until no considered action results in a positive change in community score. When multiple actions result in the same score, one that adds a vertex is chosen at random (growing the community is preferred to shrinking it).

Whenever a (local) maximum of the community score is found, the hill-climbing algorithm stops. Note that this scheme makes it very unlikely that a vertex that is essential to community connectedness is removed, even though we do not explicitly forbid this.

*Example* 4.1. Using the example graph in Figure 2, if we apply Algorithm 2 to any connected subgraph consisting of 5 or less vertices, we *always* end up with one of the two communities that form the minimum link error partitioning in Figure 2A. When

**ALGORITHM 2:** MAXIMIZE_COMMUNITY_SCORE

**Input:** Graph $G$, community $C$.
**Output:** Community $C$ modified to maximize $S(C, G)$.

1. **repeat**
2. $\quad \mathcal{A}ctions = \{\text{REMOVE}(v) \mid v \in C\}$
3. $\quad \mathcal{A}ctions = \mathcal{A}ctions \cup \{\text{ADD}(v) \mid v \in (V \setminus C) \wedge \exists w \in C : (v, w) \in E\}$
4. $\quad best \leftarrow \underset{a \in \mathcal{A}ctions}{\text{argmax}}\ \Delta S(a)$
5. $\quad$ **if** $\Delta S(best) > 0$ **then**
6. $\quad\quad C \leftarrow \text{ModifyCommunity}(C, best)$
7. $\quad$ **end if**
8. **until** $\Delta S(best) \leq 0$
9. **return** $C$

all possible subgraphs are tested as input for the algorithm, the results show that only two other local maxima exist: $\{4, \ldots, 10\}$ and $\{1, \ldots, 10\}$ (the entire graph). This demonstrates the robustness and strength both of this method and the community score.

### 4.4. Finding Concise Queries

By finding communities with (locally) maximum community scores, we address one objective of the *Diverse Top-k Community Mining* problem. The next step is to find a corresponding concise query $Q$ for a community $C$. The key idea of our approach is that this can be easily interpreted as a binary 'classification' problem. That is, a query $Q$ is required to distinguish community $C$ from the remainder of the network. Hence, $Q$ should be a discriminative description that 'predicts' whether a vertex belongs to the community or not, using only the attribute vector data. Given a community, such a description can be induced with a suitable classification method. Since it is unlikely that the induced classifier perfectly matches the community, the resulting classifier is used to reshape the community through prediction.

More formally, each vertex in $G$ has an attribute vector (its *features*) and a boolean *class label* representing community membership; a vertex is either part of community $C$ or it is not. Hence there is a mapping $\text{Dom}(a_1) \times \ldots \times \text{Dom}(a_l) \mapsto \{0, 1\}$, which we approximate from the attribute and community membership data by means of a classification algorithm. The induced classifier is subsequently used to alter the community through prediction: for each attribute vector, it is predicted whether the corresponding vertex should be part of the community or not. The set of vertices that get a positive prediction forms the new, revised community, which is returned by FIND_CONCISE_QUERY. An important side-product of the method is the query that was induced by the classification algorithm.

At first sight, a logical choice might seem to employ a well-known off-the-shelf classifier such as, e.g., SVMs. However, our goal is to find interpretable queries, which makes SVMs and other 'black-box' classifiers unsuitable. Decision trees and rule-based classifiers, on the contrary, seem more appropriate choices. Unfortunately, preliminary experiments that we performed with C4.5 [Quinlan 1993] and CART [Breiman et al. 1984] showed that no accurate classifiers could be induced on the sparse tag data that we consider in the experiment section.

Closer inspection of the data, the communities and the induced decision trees revealed that *individual tags* are hardly informative with respect to community membership. In other words, individual tags can not be used to predict whether a user is a member of a community or not. However, when considering *combinations of tags* by means of frequent itemsets [Agrawal et al. 1993], analysis demonstrated that the at-

tribute data belonging to the community is substantially different from the rest of the attribute data. That is, it should be possible to develop a *pattern-based* approach to finding concise queries, which considers co-occurrences of multiple attribute-values at the same time.

State-of-the-art methods in *supervised pattern mining* [Bringmann et al. 2010] aim to find small sets of patterns that are predictive with respect to a class label. That is, when the data is partitioned according to the occurrences of the patterns in a high-scoring pattern set, the resulting parts should be relatively pure with respect to the class label. In supervised pattern mining, heuristics are unavoidable due to the huge search spaces, i.e., in general all possible subsets of all possible patterns have to be considered. Hence, the method we will use is also a heuristic, and we cannot guarantee to find queries that optimally minimize description complexity.

The method we use is based on the ReMine algorithm [Zimmermann et al. 2010], which recursively partitions the data by splitting the data on the most informative pattern for each partition. Each pattern is some description of a recurring structure in the data, which can be verified for each object in the database; a pattern either occurs in an object or not. Using this occurrence information, each pattern naturally induces a split of the data into two parts. With $k$ patterns, the data can be partitioned in up to $2^k$ parts. Informativeness is quantified by information gain with respect to the class label. By mining informative patterns locally for each individual part and using all of them to globally partition the entire dataset, the ReMine algorithm achieves a good balance between accuracy and runtime. This perfectly suits our needs.

ReMine is a very generic algorithm that can be applied to different types of data. Since we consider data consisting of binary tag attributes, we will use itemsets as patterns, i.e., an itemset is a subset of the full set of tags (attributes). Each itemset is essentially a conjunction of conditions, and a set of such itemsets mined by ReMine can be transformed to a query $Q$. The only modification we make is that we use a (heuristic) beam search to approximate each maximally informative pattern, instead of using exhaustive search as this turned out to be practically infeasible. As ReMine aims to mine few patterns that accurately discriminate between classes, we expect to obtain good approximations of the low complexity queries we are after.

On a high level, our adaptation of the ReMine algorithm works as follows. It starts with the full dataset, and searches for a pattern that splits the dataset into two parts such that information gain with respect to community membership is maximal. For this, a standard beam search on itemsets is used, which is parametrized by a *beam width* and *maximum depth* (which equals the maximum size an itemset is allowed to have). After a pattern has been found and the data is split into two parts, the algorithm recurses on each of the new parts. An approximation for the maximally informative pattern is found for each part. After each recursion, all newly found patterns are added to the pattern set and used to determine the new global partitioning of the data. The number of recursions is the last parameter of the algorithm.

After the adapted ReMine algorithm has found a set of itemsets, it is transformed into a query $Q$. Next, $Q$ is used to 'query' the complete graph $G$ and $C$ is replaced by all resulting vertices, i.e., the newly predicted community. In other words, the existing $C$ is replaced by $C_Q$. Unless the induced classifier had 100% accuracy, this operation adds/removes vertices to/from the community. This may break a community's connectedness, but when this happens we are likely to observe a big (negative) impact on the community score.

## 5. EXPERIMENTS

In this section we present the results obtained from extensive experiments on datasets obtained from three different social networks.

For evaluation, we developed a parallelized prototype implementation of the DCM algorithm in C#. All experiments were executed on a Windows machine with quadcore CPU and hyperthreading support, using 8 threads. Our implementation is publically available on our website[6], together with two of the datasets that we used. Since we did not know in advance whether the communities would always converge, we imposed a maximum number of iterations per candidate. That is, at most five iterations of maximizing community score and finding a concise query were allowed. In all the experiments we used $\eta = 0.25$.

The greedy algorithm used to maximize the community score has no parameters. The adapted ReMine algorithm, used for inducing queries, was run with the following settings: *beam width* set to 6, *maximum depth* (equal to maximum size of an individual pattern) set to 4, and *number of ReMine recursions* set to 4.

### 5.1. Data

We use three datasets obtained from different social networks. In each dataset, users are represented by vertices in the attributed graph, and user relations are represented by edges between those vertices. A user's attribute vector is a binary vector that represents information about what tags a user used to annotate resources. This might be considered as an expression of a user's interests.

— DELICIOUS: this is a publicly available dataset from the HetRec 2011 workshop[7]. It has been obtained from the DELICIOUS social bookmarking system. Its users are connected in a social network generated from DELICIOUS' "mutual fan" relations. Each user has bookmarks, tag assignments, i.e., [user, tag, bookmark] tuples, and contact relations within the dataset's social network. The tag assignments were transformed to attribute data by taking all tags that a user ever assigned to any bookmark and assigning those to the user.
— LASTFM: our second dataset also comes from the HetRec 2011 workshop. It contains data from the LASTFM online music system. The social network is defined by the "friend" relation available in LASTFM. Each user has a list of most listened music artists and tag assignments, i.e., [user, tag, artist] tuples. As with DELICIOUS, the tag assignments were transformed to attribute data by taking all tags that a user ever assigned to any artist and assigning those to the user.
— FLICKR: this is a sample from a dump of the internal database of the popular FLICKR photo sharing platform. The social network is defined by the "contact" relation of FLICKR. As this relation is directed, we considered its undirected version by simply removing all directions; two vertices are connected with an undirected edge if at least one undirected edge exists between them. Each user has a list of tags associated that she used at least 5 times. To reduce data sparsity we limit our attention to tags that have been used by at least 50 users, and we select only users having a vocabulary of more than 100 and less than 5000 tags. Moreover, we used Wikipedia redirects for matching different ways of referring to the same entity in one single tag (e.g., *New York City*, *NYC*, *New York NY*). For a detailed description of this pre-processing we refer the reader to our previous work [van Leeuwen et al. 2009].

The basic statistics of the resulting datasets are listed in Table I.

[6] www.patternsthatmatter.org/implementations
[7] http://ir.ii.uam.es/hetrec2011/

Table I. Datasets. For each dataset the number of vertices $N$ (users), the number of undirected edges $M$ (user relations), the total number of attributes $l$ (tags), and the attribute density (average number of tags per user) are given.

| *Dataset* | $N$ | $M$ | $l$ | Att.density |
|---|---|---|---|---|
| DELICIOUS | 1,861 | 7,664 | 1,350 | 52.5 |
| FLICKR | 100,267 | 3,781,947 | 16,215 | 116.7 |
| LASTFM | 1,892 | 12,717 | 11,946 | 17.9 |

Table II. Number of candidate communities for different minimum geodesic distances $p$.

| *Dataset* | Minimum distance $p$ | | | | | |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 |
| DELICIOUS | 1861 | 508 | 267 | 180 | 1 | 1 |
| FLICKR | 100267 | 31992 | 7214 | 2006 | 175 | 62 |
| LASTFM | 1892 | 741 | 217 | 101 | 1 | 1 |

### 5.2. Reducing the candidate set

In Subsection 4.2 we described that there are (at least) three different ways to establish the initial set of candidate communities. Which method to use mostly depends on the amount of prior knowledge available for the application at hand. To avoid the potential introduction of subjective bias in this evaluation, we restrict ourselves to the most generic case in which we assume *no prior knowledge*. This implies that we will use the third and last method to generate our candidate set, which consists of vertices uniformly distributed across the graph. The number of candidates depends on the minimum geodesic distance $p$: the larger $p$, the fewer candidates. Since runtime of the algorithm increases linearly with the number of candidates, it is of interest to investigate the effect of $p$ both on the number of candidates and the resulting communities found.

Table II presents the number of initial candidates for a range of values for $p$ for each of the three datasets. When the minimum distance $p = 0$, the number of candidates equals the number of vertices in the graph. However, as $p$ increases, the number of candidates rapidly decreases. Since we expect communities to cover substantially larger regions than vertex pairs, it is expected that (slightly) fewer candidates should still maintain most high-score communities in the final result set. This hypothesis is assessed with the experiment shown in Figure 3.

Each of the plots depicts how community score evolves for the top-$k$ communities found for different settings for $p$. In particular for DELICIOUS and FLICKR, the scores hardly decrease as $p$ is increased from $0$ to $3$, indicating that (almost) the same communities are found from the substantially smaller sets of candidates. This is a obviously a good result that we will exploit. In the remainder of this section, we will experiment with $p = 3$ for DELICIOUS and FLICKR, and with $p = 2$ for LASTFM.

### 5.3. Comparing community score to existing measures

We introduced the community score as a novel measure to quantify how much a given subgraph forms a community. We have argued that it has several advantages, e.g., it allows for a simple and fast hillclimbing procedure and overlapping communities, but we have not yet investigated how it relates to existing measures. We therefore perform an empirical comparison to three of the best-known 'community measures', i.e., inverse conductance, intra cluster density, and modularity.

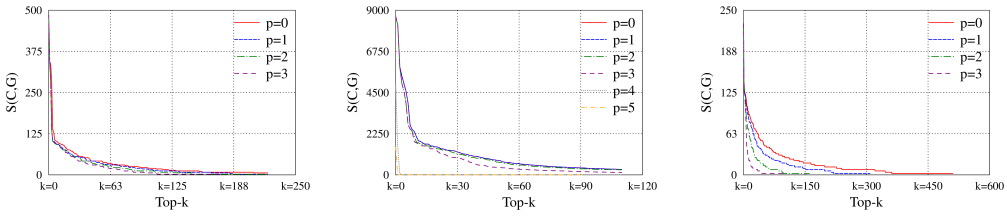
Figures 4, 5, and 6 each present three individual runs of the MAXIMIZE_COMMUNITY_SCORE algorithm, one for the top-1 community of each dataset. Note that although we here present runs for only one community per dataset, we have observed similar behavior for all top-ranking communities that were identified; these plots are representative of the general trends. In each of the plots, the x-axis represents the number of steps made by the hillclimbing algorithm, where each step is a single ADD/REMOVE action. As the algorithm optimizes community score $S$ through hillclimbing, the red line representing $S$ increases monotonically.

The first figure, Figure 4, shows that a higher community score $S$ also implies a higher inverse conductance. As for the other observations that we make here, this is generally the case for all runs that we inspected. This makes perfect sense, as inverse conductance can be regarded as the ratio between the number of edges inside the community and the number of edges leaving the community. This is clearly related to our community score, but the important difference is that $S$ grows as a community becomes larger. This can be observed from the higher score for the FLICKR example; that dataset is the largest and (thus) also contains the largest communities. This is a deliberate choice: larger communities are generally preferred over smaller ones.

The relation with intra cluster density is rather different, as can be observed from Figure 5. Since intra cluster density is the ratio between the number of edges within a community and its corresponding maximum (if the subgraph were a clique), it has a preference for very small communities. The attained densities are generally around 0.5, which is much higher than expected from a random subgraph, indicating that the community score and hillclimber together succeed in finding dense regions of the graph. However, optimizing for intra cluster density would give very different results, because smaller communities would be ranked higher.

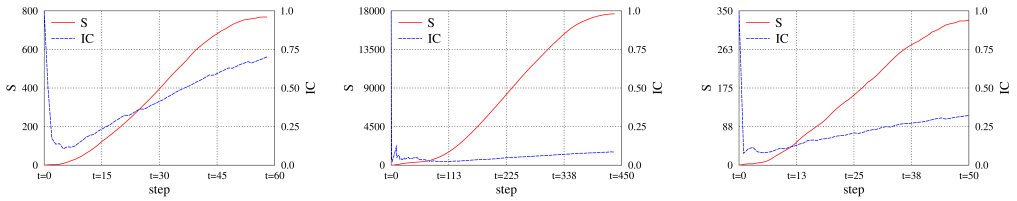
Finally, Figure 6 presents a comparison to modularity, arguably the most often used 'community measure'. Unfortunately, a direct comparison is impossible, because modularity is a *global* measure, i.e., it only works on a complete partitioning of a network into two or more non-overlapping communities. Since community score is a *local* measure, working only on individual communities, we performed this comparison as follows. For each step, we took the community and created a complete partitioning by adding a second community consisting of all nodes not in the primary community, and modularity was computed for the resulting two-community partitioning.

The resulting plot clearly shows that higher community scores imply higher modularities. This indicates that high-scoring communities found with the MCS algorithm score relatively good with respect to modularity. However, optimizing for modularity would yield different results, even when considering a local, greedy algorithm similar to ours, for three reasons. First, modularity considers both the numbers of edges within the community and within the remainder of the network, whereas community score only considers the community being optimized. Second, community score explicitly penalizes edges between the community and the rest of the network, which modu-



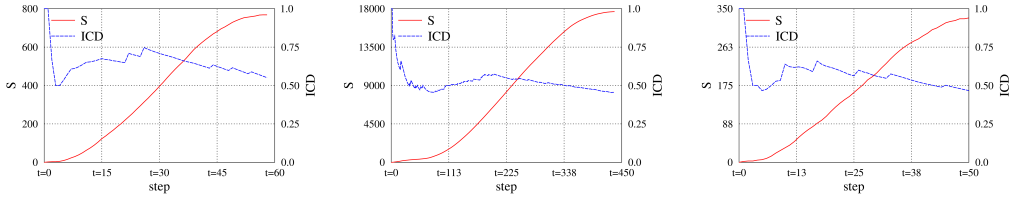

a. DELICIOUS b. FLICKR c. LASTFM

Fig. 3. The effect of candidate reduction on the top-k community scores.



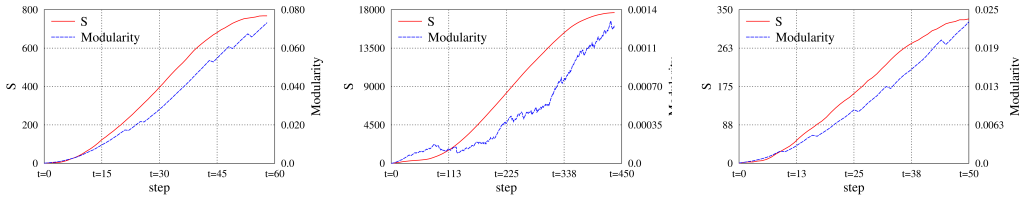

a. DELICIOUS b. FLICKR c. LASTFM

Fig. 4. Community score and inverse conductance.




a. DELICIOUS b. FLICKR c. LASTFM

Fig. 5. Community score and intra cluster density.




a. DELICIOUS b. FLICKR c. LASTFM

Fig. 6. Community score and modularity.

larity does not. Third, modularity contrasts the fractions of edges within communities to those fractions expected if the edges were distributed at random. Community score only uses the current network as reference, rather than a random null model, which helps to avoid the resolution limit from which modularity suffers and also simplifies computations.

### 5.4. Accurate community descriptions

For the DCM algorithm to work well, it must be able to induce reasonably accurate descriptions, so that several iterations of community score maximization and finding concise queries lead to high-scoring communities with accurate descriptions. Furthermore, we previously argued that we need the relatively complex query language for the application and data that we consider. We now empirically investigate whether this is indeed the case.

From all communities (subgraphs) that result from a single iteration of the MCS algorithm on all initial candidates, we pick the top-10 with respect to community score for each dataset. We induce several queries on each of these top-scoring communities and compare them. To measure how accurate the different community descriptions are, we quantify their performance in terms of *precision* and *recall*, as is usual for classification tasks. We compare the following approaches:

Table III. Description accuracy for different types of queries. Average community sizes and scores are shown for the top-10 communities used for these experiments. Furthermore, precision $p$ and recall $r$ (average $\pm$ standard deviation) of the queries induced for these communities are given for each of the three evaluated query types.

| *Dataset* | *Comm.avg* | | *Most frequent itemset* | | *Single conjunction* | | *DCM query* | |
|---|---|---|---|---|---|---|---|---|
| | $\|C\|$ | $S$ | $p$ | $r$ | $p$ | $r$ | $p$ | $r$ |
| DELICIOUS | 19.7 | 233 | $.23 \pm 0.39$ | $.54 \pm 0.15$ | $1.0 \pm 0.0$ | $.32 \pm 0.18$ | $.98 \pm 0.02$ | $.75 \pm 0.15$ |
| FLICKR | 181 | 8940 | $.06 \pm 0.08$ | $.82 \pm 0.10$ | $.71 \pm 0.17$ | $.31 \pm 0.09$ | $.995 \pm 0.01$ | $.84 \pm 0.08$ |
| LASTFM | 28.2 | 210 | $.04 \pm 0.01$ | $.71 \pm 0.04$ | $.96 \pm 0.07$ | $.26 \pm 0.10$ | $.98 \pm 0.03$ | $.61 \pm 0.17$ |

*Most frequent itemset.* We mine all itemsets that have the *maximal frequency* (or support) within the community. This is often a single item, but there can also be several itemsets with the same maximal frequency, of which some consist of multiple items. All are considered as queries, and the one that gives the highest precision is chosen (recall is the same for all most frequent itemsets).
*Single conjunction.* The query type considered in this case is of the form: $[\neg]a_i \wedge \ldots \wedge [\neg]a_k$. That is, a query is a conjunction of attributes (items) that may each be negated or not. These queries are constructed in a greedy fashion: starting from an empty query, in each iteration the condition that improves the F1-score most is added, and this is repeated until F1-score cannot be improved. F1-score is the harmonic mean of precision and recall; we tried balancing these two differently and also tried using information gain as measure, but were not able to obtain structurally better results than the ones presented.
*DCM queries.* This case represents the full query language that we consider in this paper, and queries are induced by ReMine. For a fair comparison, for these experiments we restricted the beam width to 1. This makes the search equally greedy to the *Single conjunction* case.

The results are shown in Table III, together with average community sizes and scores for the top-10 communities used for each experiment. For the *Most frequent itemset* approach, the recall values are often reasonably high. This is to be expected, since we explicitly search for a query that frequently occurs in the community. However, as we conjectured before, something that is frequent within a community is likely to be frequent in the entire dataset. This leads to very low precision, and basically means that the induced queries describe not only the communities but much larger parts of the database. This illustrates that it is essential to look for *discriminative* descriptions.

The results obtained with the *Single conjunction* approach show that much higher precisions can be obtained when one does look for discriminative queries. Unfortunately, when only considering *individual tags* the search procedure is unable to induce descriptions that have both high precision and high recall. Note that this is not due to the specific search procedure that we used; we obtained similar results with alternative heuristics. This result implies that individual tags are not informative enough and we really need to consider multiple tags at once, i.e., itemsets. This is what our adaptation of ReMine does, and the precision and recall results demonstrate the benefit of this approach. Precision is almost perfect, with very low standard deviation. Recall is not perfect, but this might be due to a lack of homophily; it may very well be that small parts of a subgraph with high community score cannot be matched by a description.

### 5.5. The overall algorithm

After having explored candidate generation, the community score with its hillclimber, and the description induction step, we now turn to the complete DCM algorithm.

Table IV. Summary of DCM results and all identified communities. For each experiment, the minimum geodesic distance $p$ is given, resulting in $|\mathcal{C}|$ candidates. The DCM algorithm results in $|\mathcal{Q}|$ queries and corresponding communities. For each of the result sets, the following averages per community are given: number of vertices $|C|$, community score $S$, number of patterns in query $|Q|$, description complexity $D$, and combined score $\sigma$.

| *Dataset* | *Experiment* | | | *Community averages* | | | | |
|---|---|---|---|---|---|---|---|---|
| | $p$ | $\|\mathcal{C}\|$ | $\|\mathcal{Q}\|$ | $\|C\|$ | $S$ | $\|Q\|$ | $D$ | $\sigma$ |
| DELICIOUS | 3 | 180 | 167 | 7.3 | 39.0 | 3.1 | 9.6 | 88.10 |
| FLICKR | 3 | 1453 | 848 | 11.0 | 190.2 | 4.7 | 10.4 | 966.50 |
| LASTFM | 2 | 217 | 134 | 8.9 | 39.5 | 4.2 | 6.5 | 21.3 |

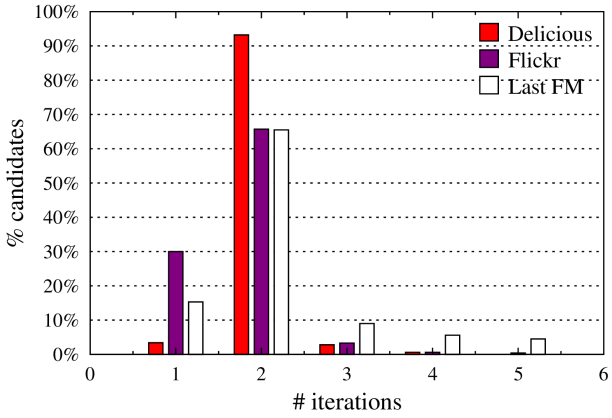



Fig. 7. Number of iterations until 'convergence'.

The overall results are summarized in Table IV. A first important observation is that the 'keep ratio' from initial candidates to diverse community sets is quite high for the used candidate sets: 180 candidates result in 167 diverse communities for DELICIOUS, respectively 1453/848 for FLICKR and 217/134 for LASTFM. On average the communities are not very large, but that's partly due to the fact that we consider the average of all results; e.g., the top-25 has larger communities on average.

The queries found for the communities generally consist of relatively few patterns, with averages ranging from 3.1 to 4.7 depending on dataset. Average description complexity $D$, i.e., the number of unique attributes used in a query, ranges up to 10, which implies that most queries are not too complex for human inspection and interpretation. Finally, the high average values for $\sigma$ reveal that there there are communities that have a very high community score $S$, as can be confirmed by inspecting the top-$k$.

At the start of this section we explained that we imposed a maximum number of iterations of five, to ensure a stopping criterion. Figure 7 shows that this maximum hardly had an effect on the final results, as by far most candidate communities have converged to a stable state already after two iterations. This indicates that the community hillclimber and description induction method are robust, resulting in stable solutions within a few iterations.

Runtimes of the DCM algorithm, presented in Table V, are low for the smaller datasets, but quite high for the largest dataset. However, if we consider the time

Table V. Runtime. MCS time is the time needed for the community score hillclimber, whereas DCM time is the full time needed for the algorithm.

| *Dataset* | *MCS time (s)* | *DCM time (s)* |
|---|---|---|
| DELICIOUS | 0.5 | 47 |
| FLICKR | 1493 | 219373 |
| LASTFM | 0.7 | 82 |

Table VI. Example community properties

| *Community* | $\|Q\|$ | $D$ | $\|C\|$ | $S$ | top-$k$ |
|---|---|---|---|---|---|
| FLICKR 1 | 33 | 32 | 58 | 1188 | 168 |
| FLICKR 2 | 25 | 30 | 125 | 3647 | 117 |
| FLICKR 3 | 46 | 39 | 75 | 2011 | 148 |
| LASTFM 1 | 3 | 5 | 9 | 27 | 42 |
| LASTFM 2 | 4 | 4 | 7 | 18 | 43 |

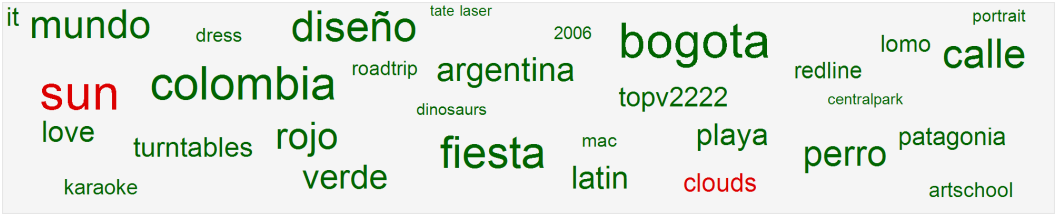
needed for the Maximize_Community_Score (MCS) method, we observe that this is only a fraction of the total time. The single reason for this is that even the heuristic implementation of ReMine needs quite some time to induce good descriptions, because the tag data that we consider is sparse and identifying informative patterns is tough. In other words, the MCS routine is very fast and can be easily performed on large-scale graphs, but scalability of the description induction method depends much on the data at hand. In very sparse tag data homophily is not always perfect, and therefore ReMine has difficulties coming up with reasonable queries in little time. This might be much faster with different data though, when the homophily assumption holds better. Note that runtimes are given in CPU time, not in wall clock time; wall clock times are approximately 8 times shorter due to parallelization.

### 5.6. Anecdotal communities

To conclude the experiment section, we take a closer look at some of the individual communities found by the DCM algorithm. A first important observation to make is that all result sets contain *overlapping* communities, which shows that our goal to find potentially overlapping communities has been achieved. By using the Jaccard-based post-selection, communities that are clearly redundant are removed, while others are maintained. Pairwise overlaps of up to about 10% of the vertices regularly occur in each of the datasets. In case of Flickr, in practice this means that sometimes around 50 vertices occur in two communities consisting of roughly 400-500 vertices.

Five example communities, three from FLICKR and two from LASTFM, are shown in Figure 8 and Table VI. The table presents basic properties such as query and community sizes, whereas the figure shows tag clouds which were created to improve interpretability. The tag clouds were created from the queries by drawing each occurring tag once, where font size is relative to information gain of that individual tag, i.e., how much information with respect to community membership would be gained by using that individual tag to split the full network. In addition, when a tag occurs in a negated conjunction, it is depicted in red instead of green. Providing an objective evaluation for these example queries is inherently hard, but on first sight they seem to characterize clear interests/communities.

The first LASTFM example indicates that there is a small group of fans of the band Queensryche that do *not* use tags ‘metal’, ‘metallica’, ‘rock’, and ‘metalcore’. Considering the second LASTFM community, for example, rapper Eminem has multiple songs and even a record label that all involve the term ‘shady’, and Eminem and G-Unit are similar artists. They may have performed together in Stuttgart, which would explain

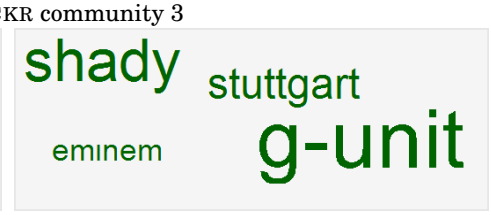



a. FLICKR community 1


b. FLICKR community 2


c. FLICKR community 3


d. LASTFM community 1




e. LASTFM community 2

Fig. 8. Tag clouds for different communities

why there is a community of LASTFM users that has this particular combination of tags. Looking at Table VI, we can see that this community is relatively small and consists of only 7 users. In general, the communities found from LASTFM are quite small. The reason for this is that LASTFM's tag data is extremely sparse, as can be witnessed from Table I; density of the tag data is only 0.15%.

The tag data of FLICKR is considerably denser, and this results in substantially larger communities. The biggest example community is FLICKR 2, which consists of 125 users. It has a community score of 3647, and 30 different tags are used in the query. In total, the query is formed by 25 itemsets. In the top-$k$ this particular result was found on rank 117. This may not seem high, but in total 1859 communities were found. Hence, all example communities were in the top 10%. Better pre-processing of the data would probably give descriptions that are easier to interpret. This was beyond the scope of this paper though.

## 6. CONCLUSIONS

Motivated by the needs of real-world applications we introduce a new framework for finding $k$ possibly overlapping communities together with their corresponding descrip-

tions, where $k$ is a given number. The discovered communities maximize a quality function combining cohesiveness of the communities with conciseness of their descriptions, and are diverse among each other in terms of the node sets they consist of.

We achieve this by means of a fast and effective algorithm that alternates between two phases: a hill-climbing phase that grows communities, and a description induction phase which is based on techniques from supervised pattern set mining. Our framework has the nice feature of being able to build a well-described, cohesive community starting from any given description or seed set of nodes. This provides a way to exploit existing background knowledge in the mining process, and makes our framework very flexible and easily applicable in many real-world scenarios. For instance, it can also be applied in dynamic contexts in which the social network structure and the associated information keep evolving: in this case, our framework can simply use previously mined communities or descriptions as starting input, and let them adapt to the newly evolved network.

Our experiments on three real-world online social networks, i.e., DELICIOUS, FLICKR, and LASTFM, confirm that the proposed framework achieves good quality communities and descriptions.

## ACKNOWLEDGMENTS

This research is partially supported by a Rubicon grant of the Netherlands Organisation for Scientific Research (NWO), by the Torres Quevedo Program of the Spanish Ministry of Science and Innovation, co-funded by the European Social Fund, and by the Spanish Centre for the Development of Industrial Technology under the CENIT program, project CEN-20101037, Social Media (`http://www.cenitsocialmedia.es/`).

## REFERENCES

AGRAWAL, R., IMIELINKSI, T., AND SWAMI, A. 1993. Mining association rules between sets of items in large databases. In *Proceedings of the SIGMOD'93*. ACM, 207–216.

AHN, Y.-Y., BAGROW, J. P., AND LEHMANN, S. 2010. Link communities reveal multiscale complexity in networks. *Nature 466,* 7307, 761–764.

AIROLDI, E. M., BLEI, D. M., FIENBERG, S. E., AND XING, E. P. 2008. Mixed membership stochastic blockmodels. *Journal of Machine Learning Research 9*, 1981–2014.

ATZMUELLER, M. AND MITZLAFF, F. 2011. Efficient descriptive community mining. In *Proc. 24th Intl. FLAIRS Conference*. AAAI Press.

BALL, B., KARRER, B., AND NEWMAN, M. E. J. 2011. An efficient and principled method for detecting communities in networks. *Phys. Rev. E 84, 036103*.

BANSAL, N., BLUM, A., AND CHAWLA, S. 2004. Correlation clustering. *Machine Learning 56,* 1-3, 89–113.

BARBIERI, N., BONCHI, F., AND MANCO, G. 2013. Cascade-based community detection. In *Sixth ACM International Conference on Web Search and Data Mining, (WSDM'13)*.

BAUMES, J., GOLDBERG, M. K., KRISHNAMOORTHY, M. S., MAGDON-ISMAIL, M., AND PRESTON, N. 2005. Finding communities by clustering a graph into overlapping subgraphs. In *IADIS AC'05*. 97–104.

BONCHI, F., GIONIS, A., GULLO, F., AND UKKONEN, A. 2012. Chromatic Correlation Clustering. In *KDD*.

BREIMAN, L., FRIEDMAN, J., OLSHEN, R., AND STONE, C. 1984. *Classification and Regression Trees.* Wadsworth.

BREVE, F. A., ZHAO, L., QUILES, M. G., PEDRYCZ, W., AND LIU, J. 2011. Particle competition and cooperation for uncovering network overlap community structure. In *ISNN*.

BRINGMANN, B., NIJSSEN, S., TATTI, N., VREEKEN, J., AND ZIMMERMAN, A. 2010. Mining sets of patterns. In *Tutorial at ECML PKDD'10*. `http://www.cs.kuleuven.be/conference/msop/`.

CHEN, W., LIU, Z., SUN, X., AND WANG, Y. 2010. A game-theoretic framework to identify overlapping communities in social networks. *Data Min. Knowl. Discov. 21,* 2, 224–240.

DONG, G. AND LI, J. 1999. Efficient mining of emerging patterns: Discovering trends and differences. In *Proceedings of the SIGKDD'99*. 43–52.

EVANS, T. S. AND LAMBIOTTE, R. 2009. Line graphs, link partitions, and overlapping communities. *Phys. Rev. E 80*, 016105.

FORTUNATO, S. 2010. Community detection in graphs. *Phys. Rep. 486*.

GIRVAN, M. AND NEWMAN, M. 2002. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences 99,* 12, 7821–7826.

GREGORY, S. 2010. Finding overlapping communities in networks by label propagation. *New Journal of Physics 12,* 10, 103018.

KHAN, A., YAN, X., AND WU, K.-L. 2010. Towards proximity pattern mining in large graphs. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. SIGMOD '10. ACM, New York, NY, USA, 867–878.

KIM, Y. AND JEONG, H. 2011. The map equation for link communities. *Phys. Rev. E 84, 026110*.

KUMPULA, J. M., KIVELÄ, M., KASKI, K., AND SARAMÄKI, J. 2008. Sequential algorithm for fast clique percolation. *Phys. Rev. E 78*, 026109.

LANCICHINETTI, A., FORTUNATO, S., AND KERTESZ, J. 2009. Detecting the overlapping and hierarchical community structure of complex networks. *New Journal of Physics 11*, 033015.

LANCICHINETTI, A., RADICCHI, F., RAMASCO, J. J., AND FORTUNATO, S. 2011. Finding statistically significant communities in networks. *PLoS ONE 6,* 4, e18961.

LATTANZI, S. AND SIVAKUMAR, D. 2009. Affiliation networks. In *STOC*. 427–434.

M. VAN LEEUWEN. 2010. Maximal exceptions with minimal descriptions. *Data Min. Knowl. Discov. 21,* 2, 259–276.

LEMAN, D., FEELDERS, A., AND KNOBBE, A. 2008. Exceptional model mining. In *Proceedings of the ECML/PKDD'08*. Vol. 2. 1–16.

MCPHERSON, M., SMITH-LOVIN, L., AND COOK, J. 2001. Birds of a feather: Homophily in social networks. *Annual Review of Sociology. 27:41544*.

MOSER, F., COLAK, R., RAFIEY, A., AND ESTER, M. 2009. Mining cohesive patterns from graphs with feature vectors. In *SDM* (2009-06-12). SIAM, 593–604.

NEWMAN, M. 2004. Fast algorithm for detecting community structure in networks. *Phys. Rev. E 69, 066133*.

NEWMAN, M. AND GIRVAN, M. 2004. Finding and evaluating community structure in networks. *Phys. Rev. E 69, 026113*.

PADROL-SUREDA, A., PERARNAU-LLOBET, G., PFEIFLE, J., AND MUNTÉS-MULERO, V. 2010. Overlapping community search for social networks. In *ICDE*. 992–995.

PALLA, G., DERENYI, I., FARKAS, I., AND VICSEK, T. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*.

PALLA, G., FARKAS, I., POLLNER, P., DERENYI, I., AND VICSEK, T. 2007. Directed network modules. *New J. Phys. 9,* 6.

QUINLAN, J. 1993. *C4.5: Programs for Machine Learning*. Morgan-Kaufmann.

RAGHAVAN, U. N., ALBERT, R., AND KUMARA, S. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E 76*, 036106.

SHAMIR, R., SHARAN, R., AND TSUR, D. 2004. Cluster graph modification problems. *Discrete Applied Mathematics 144,* 1-2, 173–182.

SILVA, A., MEIRA, JR., W., AND ZAKI, M. J. 2010. Structural correlation pattern mining for large graphs. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*. MLG '10. ACM, New York, NY, USA, 119–126.

TANG, L., WANG, X., AND LIU, H. 2009. Uncoverning groups via heterogeneous interaction analysis. In *The Ninth IEEE International Conference on Data Mining, ICDM'09*. 503–512.

VAN LEEUWEN, M., BONCHI, F., SIGURBJÖRNSSON, B., AND SIEBES, A. 2009. Compressing tags to find interesting media groups. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM'09,*. 1147–1156.

WANG, X., TANG, L., GAO, H., AND LIU, H. 2010. Discovering overlapping groups in social media. In *The 10th IEEE Int. Conf. on Data Mining (ICDM)*.

XIE, J., KELLEY, S., AND SZYMANSKI, B. 2013. Overlapping community detection in networks: the state of the art and comparative study. *ACM Computing Survey 45,* 4.

XIE, J. AND SZYMANSKI, B. K. 2012. Towards linear time overlapping community detection in social networks. In *PAKDD*.

XIE, J., SZYMANSKI, B. K., AND LIU, X. 2011. Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In *ICDM Workshops*. 344–349.

ZHOU, Y., CHENG, H., AND YU, J. X. 2009. Graph clustering based on structural/attribute similarities. *PVLDB 2,* 1, 718–729.

ZIMMERMANN, A., BRINGMANN, B., AND RÜCKERT, U. 2010. Fast, effective molecular feature mining by local optimization. In *Proceedings of the ECML PKDD'10*. 563–578.