# Efficient Personalized Community Detection via Genetic Evolution

Zheng Gao
Indiana University Bloomington
IN, United States
gao27@indiana.edu

Chun Guo
Pandora Media LLC
CA, United States
cguo@pandora.com

Xiaozhong Liu
Indiana University Bloomington
IN, United States
liu237@indiana.edu

## ABSTRACT

Personalized community detection aims to generate communities associated with user need on graphs, which benefits many downstream tasks such as node recommendation and link prediction for users, etc. It is of great importance but lack of enough attention in previous studies which are on topics of user-independent, semi-supervised, or top-K user-centric community detection. Meanwhile, most of their models are time consuming due to the complex graph structure. Different from these topics, personalized community detection requires to provide higher-resolution partition on nodes that are more relevant to user need while coarser manner partition on the remaining less relevant nodes. In this paper, to solve this task in an efficient way, we propose a genetic model including an offline and an online step. In the offline step, the user-independent community structure is encoded as a binary tree. And subsequently an online genetic pruning step is applied to partition the tree into communities. To accelerate the speed, we also deploy a distributed version of our model to run under parallel environment. Extensive experiments on multiple datasets show that our model outperforms the state-of-arts with significantly reduced running time.

## CCS CONCEPTS

• **Information systems** → *Community detection*; • **Computing methodologies** → *Genetic programming*.

## KEYWORDS

Personalized community detection, Graph mining, Network analysis, Genetic programming

## 1 INTRODUCTION

Community detection is an important topic in graph mining. By learning node community labels on the graph, we are able to detect node hidden attributes as well as explore the closeness between
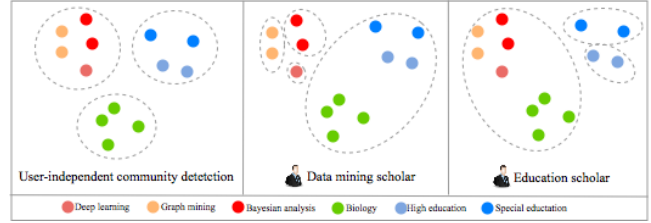
**Figure 1: An example of personalized community detection on a scholarly graph**

nodes [13, 40]. Conventional methods are mostly user-independent to detect communities solely relying on graph topological structure [37], generate semi-supervised communities with node constraints, or select top-K sub graphs as user-centric communities. These approaches are no longer enough to satisfy users with a pursuit of personalization, which makes involving user need into community detection to become an inevitable task. Specifically, from a user-centric viewpoint, the ideal communities should provide a high-resolution partition in areas of the graph relevant to the user need and a coarse manner partition on the remaining areas so as to best depict user need (*we also call it "query" in the rest of this paper*) in concentrated areas while fuzz irrelevant areas.

For instance, in Figure 1, two different scholars in *education* and *data mining* domains may consume the same scholarly graph differently because they may need more detailed community exploration in their own domains while generalized community information in other irrelevant domains (e.g., the *data mining* scholar needs more detailed communities such as *Deep Learning*, *Graph Mining*, and *Bayesian Analysis*. While an *education* scholar may need to generalize those communities as *Computer Science* or just *Science*).

As aforementioned, current investigations are still with limited scope. First, user-independent approaches solely consider graph topological structure without user need. For instance, [39] proposes a novel nonlinear reconstruction method by adopting deep neural networks to generate communities with the maximum modularity. [6] calculates the Jacaard similarity of neighborhood graphs to decide whether two nodes belong to the same community. Second, semi-supervised approaches detect communities restricted by pre-selected seed nodes. As different user needs refer to different seeds, each individual user requires a separate process to run the whole model completely to get personalized communities, which is inapplicable in real cases. [28] introduces a joint approach to decompose the matrices associated with multi-layer networks and prior information into a community matrix and multiple coefficient matrices. [20] defines a non-negativity and a latent sparsity

constraint to guide community detection. Third, sub-graph selection approaches only generate communities from the partial graph instead of the whole one. [21] designs two index infrastructures including a topology index and a metapath weight index to exploit top ranked subgraphs. Similarly, [43] indexes the graph and uses cluster coefficient as the criteria for sub-graph selection.

To detect personalized communities on the whole graph, in this paper, we propose a **g**enetic **P**ersonalized **C**ommunity **D**etection (gPCD) model with an offline and an online step. Specifically, in the offline step, we convert the user-independent graph community to a binary community tree which is encoded with binary code. Subsequently, a deep learning method is utilized to learn low-dimensional embedding representations for both user need and nodes on the binary community tree. In the online step, we propose a genetic tree-pruning approach on the tree to detect personalized communities by maximizing user need and minimizing user searching cost simultaneously. The whole genetic approach runs in an iterative manner to simulate an evolutionary process and generate a number of partition candidates which are regarded as "chromosomes" in each genetic generation. Through the selection, cross-over and mutation process, successive chromosomes are bred as better personalized community partitions to meet with user need.

The contribution of this study is threefold.

- We address a novel personalized community problem and propose a model to generate different-resolution communities associated with user need.
- Our model contains an offline and an online step. The offline step takes charge of most calculation to enable an efficient online step: The construction of binary community tree has a time complexity of $O(\|V\|^2)$ in the worst case where $\|V\|$ denotes the number of vertices in the graph; Representation learning on both binary community tree and user need has the same time complexity as Node2vec [19]. The online genetic pruning step running under the parallel environment achieves $O(\frac{2^d KP}{M})$ time complexity where $d$ denotes the depth of the tree, $K$ denotes the community number, $P$ denotes the initialized population size in the genetic approach and $M$ denotes the number of Mappers/ Reducers in Hadoop Distributed File System (HDFS).
- We evaluate our model on a scholarly graph and a music graph. In our model, the offline step is separately calculated and keeps unchanged once constructed, while the online step guides the personalized community detection. Hence we only compare the online step results with baselines' performance. Extensive experiments shows our model outperforms in terms of both accuracy and efficiency.

## 2 LITERATURE REVIEW

The problem of exploring community structure in graphs has long been a central research topic in network science [5, 14, 27]. From user-centric viewpoint, existing community detection methods can be divided into three categories: user-independent models, semi-supervised models and top-K community selection models.

**User-independent Community Detection:** Models belonging to this category aims to generate communities solely relying on graph structure without considering any auxiliary information. As "modularity" is a classic metric to evaluate community quality [18], there are a bunch of works which try to generate community partitions by maximizing graph modularity [4, 8, 31]. Dynamic models can handle higher order structures with hierarchical communities [3]. Random walk dynamics are by far the most exploited track in community detection. For example, Infomap [34] detects communities by minimizing the description length for random walk paths. An extended Multilevel Infomap models reveals hierarchical community structure in a complex graph [35]. Recent works start to leverage deep learning methods for community detection . DeepWalk [32] learns node embeddings based on random walks and Kmeans is subsequently applied on node embeddings to detect communities. Node2vec [19] and edge2vec [16] are both extended models of DeepWalk which design a biased random walk to learn node embeddings better representing graph structure. [39] aims to maximize the graph modularity with a deep learning framework. [2] also uses neural networks to generate content-based communities for online social networks. Some other methods derived from statistical models are also able to detect communities. However, they are usually too time consuming to apply on large scale graphs [1, 22, 25].

**Semi-supervised Community Detection:** In this track, community detection models are restricted by a pre-defined constraint [17]. [20] using multi-aspect information of heterogeneous graph and a small ratio of node constraints to detect both non-overlapping and overlapping communities. [26] detects communities by actively selecting a small amount of links as side information to sharpen the boundaries between communities and compact the connections within communities. [30] proposes a new community measurement metric and a spectral model to detect communities. [15] introduces a new constrained community detection model based on Lagrangian multipliers to incorporate and fully satisfy the node labels and pairwise constraints. [36] designs a unified non-negative matrix factorization framework simultaneously for community detection and semantic matching by integrating both semi-supervised information and node content. [29] discusses the equivalence of the objective functions of the symmetric non-negative matrix factorization (SNMF) and the maximum optimization of modularity density first, and derives the community detection model from the equivalence.

**Top-K Community Selection:** Some prior works also exploit on social graphs to find top-K groups of nodes that are relevant to a user query. [23] addresses the problem of forming a team of skilled individuals based on a given task, while minimizing the communication cost among the members of the team. In order to find the top K most relevant subgraphs given a user query, [21] introduces an offline approach generating two index structures for the network: a topology index, and a graph maximum metapath weight index. An online novel top-K approach is subsequently applied to exploit these indexes for answering queries. To solve the same problem, [43] designs a balanced tree (G-Tree) to index the large graph first and then proposes the rank matching (RM) algorithm to locate the top-k matches of query Q by pruning on the balanced tree. [42] develops a new graph distance measure using the maximum common subgraph (MCS), which is more accurate than the feature based measures, to find top-K similar graphs for
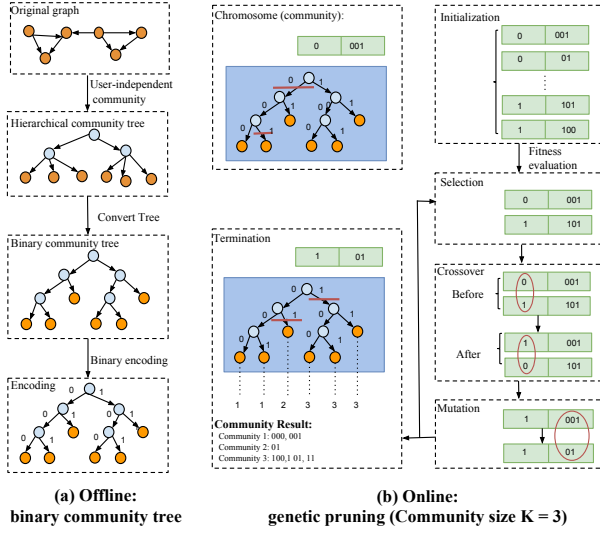
**(a) Offline:**
**binary community tree**

**(b) Online:**
**genetic pruning (Community size K = 3)**

**Figure 2: The framework of gPCD model.** (a) refers to the offline construction step on the original graph and (b) refers to the online genetic pruning step to generate personalized communities.

a user query and offers an optimization approach to accelerate running speed.

## 3 GENETIC PERSONALIZED COMMUNITY DETECTION

Our proposed gPCD model contains an offline and an online step. Figure 2 shows the pipeline of the whole framework. The offline step first encodes the user-independent binary community tree (*Section 3.1*), and subsequently learns embedding representations for both user need and nodes on the binary community tree (*Section 3.2*). The online step introduces the genetic personalized community detection approach (*Section 3.3*). To accelerate the running speed, a distributed version of gPCD model is also deployed on HDFS (*Section 3.4*). To disambiguate the notations mentioned in this section to better explain our gPCD model, some commonly used notations can be found in Table 1.

### 3.1 Offline Community Tree Index

One challenge of solving personalized community detection problem is the computational cost due to the complexity of personalization and graph structure. In order to reduce the online workload, most of computation cost is put into the one-time offline step whose time cost can be excluded from the online personalized community detection step. Thus, we first convert the graph into a binary community tree offline to retain user-independent community information.

We employed the Infomap algorithm [35] to generate the user-independent communities solely based on the graph $G(V, E)$. Infomap algorithm simulates a random walker wandering on the graph and indexes the description length of his random walk path via multilevel codebooks. By minimizing the description length based on the map equation below, community structures are formed

| Notations | Descriptions |
|---|---|
| $G(V, E)$ | Original graph $G$ with vertex set $V$ and edge set $E$ |
| $T_c(N^c, L^c)$ | The hierarchical community tree generated from graph $G(V, E)$ with node set $N^c$ and link set $L^c$. Each node $N_k^c \in N^c$ denotes a group of vertices belonging to $V$. |
| $T_b(N^b, L^b)$ | The binary community tree reconstructed from $T_c(N^c, L^c)$. Each node $N_k^b \in N^b$ denotes a group of vertices belonging to $V$. |
| $B$ | The binary codebook for $T_b(N^b, L^b)$. Particularly, $B_k \in B$ denotes the binary code of both $N_k^b \in N^b$ and $L_k^b \in L^b$ where $L_k^b$ is the link points to node $N_k^b$. |

**Table 1: Commonly used notations in gPCD model**

for the graph.

$$L(M) = q_\frown H(Q) + \sum_{i=1}^{m} p_\cup^i H(\mathcal{P}^i) \tag{1}$$

where $L(M)$ is the description length for a random walker in the current community $M$. $q_\frown$ and $p_\cup^i$ are the jumping rates between communities and within the $i_{th}$ community. $H(Q)$ is the frequency-weighted average length of codewords in the global index codebook and $H(\mathcal{P}^i)$ is frequency-weighted average length of codewords in the $i_{th}$ community codebook. Followed by this equation to partition communities into sub-communities , a hierarchical community tree $T_c(N^c, L^c)$ is constructed from the original graph $G(V, E)$.

In $T_c(N^c, L^c)$, each parent node can have multiple child nodes which can be regarded as a community partition on the parent node. For instance, a node $N_k^c \in N^c$ from $T_c(N^c, L^c)$ represents a community of vertices. Its $m$ child nodes $\{N_{k_1}^c, N_{k_2}^c, ..., N_{k_m}^c\}$ represent $m$ sub-communities of vertices from $G(V, E)$ where we have $\bigcap_{i=1}^{m} N_{k_i}^c = \varnothing$ and $\bigcup_{i=1}^{m} N_{k_i}^c = N_k^c$.

In order to achieve an efficient personalized community detection in the following online step, we convert the hierarchical community tree $T_c(N^c, L^c)$ to a binary community tree $T_b(N^b, L^b)$ for index. Specifically, for $m$ child nodes of a parent node $N_k^c$, a bottom-up approach is proposed to merge a selected pair of sibling nodes as a new node in an iterative manner. The approach runs until all $m$ child nodes merged together to form the parent node $N_k^c$. To avoid an unbalanced tree where small communities are always left to merge with huge communities in the end, we first select the node with the smallest community size among all sibling nodes in each merging step. It is merged with its sibling node with the largest normalized linked weight (Please refer to Figure 2(a)). The normalized linked weight function $w(\cdot)$ between two nodes $N_i^c$ and $N_j^c$ is defined as:

$$w(N_i^c, N_j^c) = \frac{N_i^c \odot N_j^c - \frac{\mathcal{D}(N_i^c) \cdot \mathcal{D}(N_j^c)}{2\|E\|}}{\|N_i^c\| \|N_j^c\|} \tag{2}$$

where $N_i^c \odot N_j^c$ denotes the number of edges linked between vertices in node $N_i^c$ and $N_j^c$, which can be interpreted as the linkage strength between them; $\|N_i^c\|$ is the number of vertices inside node $N_i^c$; $\mathcal{D}(N_i^c)$ is the out-degree of node $N_i^c$ (the total number of edges

linked to other nodes) and $\|E\|$ is the total number of edges in the original graph $G(V, E)$. $\frac{\mathcal{D}(N_i^c) \cdot \mathcal{D}(N_j^c)}{2\|E\|}$ denotes the random linkage strength between node $N_i^c$ and $N_j^c$. The $w(\cdot)$ function calculates how much that two nodes are better connected beyond random connection and is normalized by node size. Given the node $N_i^c$ with the smallest community size and all its sibling node set $S$, The merging step can be formulated as:

$$N_j^c \Leftarrow \underset{N_j^c \in S}{\operatorname{argmax}} w(N_i^c, N_j^c)$$
$$N_*^c = N_i^c \bigcup N_j^c \tag{3}$$

The bottom-up process will stop until all child nodes are merged together to form the parent node. In the end, the hierarchical community tree $T_c(N^c, L^c)$ is fully converted to a binary community tree $T_b(N^b, L^b)$ with user-independent community information. The node size $\|N^b\|$ as well as the link size $\|L^b\|$ in $T_b(N^b, L^b)$ is at most $2\|V\|$ which is smaller than the size of original graph $G(V, E)$. If we consider to form the binary community tree with only $k$ levels, the size of $T_b(N^b, L^b)$ can be even smaller.

For running time analysis, calculating normalized linked weight takes constant time. In each merging step, node pair selection takes linear time. Therefore, in the worst case, the time complexity of binary community tree construction is $O(\|V\|^2)$ where the depth of the hierarchical community tree $T_c(N^c, L^c)$ is 1 and each vertex in $G(V, E)$ forms a single-vertex community.

To encode the nodes and links on $T_b(N^b, L^b)$ as binary code, the root node is encoded as 'null' first. For a parent node $N_k^b$ with its left child node $N_{k_l}^b$ and right child node $N_{k_r}^b$, the binary code of a child node and the related link defined in the Notation Table 1 is calculated as:

$$B_{k_i} = \begin{cases} B_k + \text{"0"}, & i = \text{"l"} \\ B_k + \text{"1"}, & i = \text{"r"} \end{cases} \tag{4}$$

For instance, if the node $N_k^b$ is with binary code "00," its left child node's binary code is "000" while the right child node's binary code is "001." The link $L_k^b$ that points to $N_k^b$ also has the binary code "00".

## 3.2 Community and User Need Representation

Node2vec [19] helps to learn fixed-length embeddings for both user need and communities. It simulates random walks on the graph $G(V, E)$ and learns the vertex embedding by optimizing the sequential relationships from random walk paths. In the end, each vertex $V_k$ in graph $G(V, E)$ has a vector representation as $\vec{V}_k$. Each node $N_k^b$ on the binary community tree $T_b(N^b, L^b)$ refers to a vertex community $C_k$ in the graph $G(V, E)$. Its representation $\vec{C}_k$ is calculated as the averaged embedding of all vertices inside the community. In the end, the binary community tree $T_b(N^b, L^b)$ represents the hierarchical community partition of Graph $G(V, E)$. Each node $N_k^b$ on the tree is indexed with three attributes: a group of vertices from graph $G(V, E)$, a binary code $B_k$, and an embedding representation $\vec{C}_k$.

On the other hand, User need (query) $I$ can also be represented by a combination of $t$ different vertices $\{V_1, V_2...V_t\}$ in the graph

$G(V, E)$. In this study, two different scenarios for user need representation are offered:

**Vertex-based Query**. User need can be directly represented by the vertices based on the generation probability $P(V_k|I)$ between them. Hence the user need representation $\vec{I}$ is calculated as:

$$\vec{I} = \sum_{k=1}^{t} P(V_k|I) \cdot \vec{V}_k \tag{5}$$

For instance, in a music sharing network, each vertex $V_k$ denotes a music and a user listing history can be used to reflect the user need $I$. $P(V_k|I)$ therefore can be regarded as the probability that a music being listened by the user.

**Text-based Query**. Under this scenario, user need $I$ is represented as a text query, and each vertex $V_k$ in the graph $G(V, E)$ also contains textual content. From language model viewpoint, each vertex importance weight is the query likelihood $P(I|V_k)$, and the user need can is the weighted average of vertex embedding:

$$\vec{I} = \frac{\sum_{k=1}^{t} P(I|V_k) \cdot \vec{V}_k}{\sum_{k=1}^{t} P(I|V_k)} \tag{6}$$

In either case, user need is conceptualized as an embedding with the same dimension as the node embeddings on the binary community tree. It enables very efficient online personalized community detection in later steps. And running Node2vec takes most of the time in this step.

## 3.3 Online Genetic Pruning

The whole process, as the Figure 2 shows, is to generate communities by pruning the constructed binary community tree. After each cut on a link, the original tree will be separated into two sub-trees. After a specific number of cuts to the links on the tree, a fixed number of communities with different resolutions are detected. By applying genetic selection, crossover, and mutation steps, our model converges to the optimized solution efficiently with a clear-defined fitness function. The details are shown in the following paragraphs.

### 3.3.1 Genetic Representation.

A chromosome is formed by a set of genes $\{g_1, g_2, ..., g_{K-1}\}$, and each gene $g_i$ holds a cut link $L_i^b$ in the binary community tree $T_b(N^b, L^b)$. Since communities can be created by cutting links on the offline tree, a chromosome can be represented as a generated community partition of the original graph $G(V, E)$ in this way. To constrain a chromosome so that it can be decoded to a fixed number of communities, four **Cutting Rules** are necessarily to be applied:

- **Rule 1**: If a link $L_i^b$ is picked to cut on the binary community tree $T_b(N^b, L^b)$, its pointing node $N_i^b$ will be retrieved and all the vertices within it form a community.
- **Rule 2**: If a link $L_i^b$ and its ancestor link $L_j^b$ are stored in the same chromosome, all vertices in $L_i^b$'s related node $N_i^b$ are a subset of vertices in $L_j^b$'s related node $N_j^b$. In this case, the two cut links generate two communities where community $C_i$ is all vertices in $N_i^b$ and community $C_j$ is the remaining vertices in $N_j^b$ but not in $N_i^b$. It can be formulated as $C_i =$

$\bigcup_k \{V_k | (V_k \in N_i^b)\}$ and community $C_j = \bigcup_k \{V_k | (V_k \in N_j^b) \cap (V_k \notin N_i^b)\}$.

- **Rule 3**: Sibling links can't be stored in the same chromosome, and it is not allowed to store duplicated links in a chromosome.
- **Rule 4**: The depth's upper bound is set to be $d$, which means all eligible cut links should be located in the first $d$ depth on the binary community tree. It avoids to generate super tiny communities and hugely reduces the genetic searching scope on cut links.

By applying the cutting rules to the online pruning process, we ensure a $K$ community partition can be retrieved from a chromosome with $K - 1$ cut links.

### 3.3.2 Initialization.

Initially, the model generates a given number $P$ chromosomes as the seed "chromosome population". And each iteration in the genetic approach breeds a new "generation" of chromosome population. In order to ensure a chromosome is an encoder of a $K$ community partition, $K - 1$ links will be randomly picked (on the binary community tree) following the cutting rules and stored in the related genes of a chromosome.

### 3.3.3 Fitness Function.

As each chromosome can be decoded as a community partition, it is important to measure the quality of each generated chromosome (how well the generated communities can satisfy user need). The measurement is hosted in a fitness function.

In our model, the fitness function simulates the user searching behavior on the graph given the community partition. For instance, a user can be more likely to pick the most relevant communities while avoiding the redundant information already selected. With the help of the offline step, the relevance score of node $N_i^b$ (community $C_i$) towards user need $I$ can be calculated with the cosine similarity $cos(\vec{I}, \vec{C_i})$, and the information redundancy can be $\sum_{C_j \in S_c} cos(\vec{C_j}, \vec{C_i})$ where $S_c$ is the set of communities that the user have already picked from the communities decoded from the target chromosome. Following this, we use a greedy selection approach to iteratively rank and pick communities given a chromosome (community partition) until all communities are picked:

$$\underset{C_i}{\text{argmax}} \; \lambda \cdot cos(\vec{I}, \vec{C_i}) - (1 - \lambda) \cdot \frac{\sum_{C_j \in S_c} cos(\vec{C_j}, \vec{C_i})}{\|S_c\|} \quad (7)$$

where $C_i$ is the candidate community to be picked and $\|S_c\|$ is the number of communities already been picked. $\lambda$ is a parameter controls whether user prefers to obtain new useful information or to avoid redundant information.

For chromosome quality evaluation, a query-generated vertex ranking list $l_q$ is first created by retrieving top $n$ vertices relevant to the query (user need) with the largest cosine similarities on embeddings of graph $G(V, E)$. We store the top $n$ vertex ranking label $R(l_q) = \{1, 2, ..., n\}$ as the pseudo ground truth. On the other hand, given the $k_{th}$ chromosome $ch_k$ in the current chromosome generation, we can also retrieve the community-generated ranking of each vertex $V_k \in l_q$ from the sequentially selected communities decoded by the chromosome. We assign the ranking label on each

vertex $V_k$ based on the following formula:

$$\sum_{V_j \in l_q} \Phi(\delta(V_j) < \delta(V_k)) + 1 \quad (8)$$

$V_j$ refers to all vertices in $l_q$. $\delta(V_j)$ shows the ranking (selection sequence) of the community which $V_j$ belongs to. $\Phi$ is a binary operator to determine whether $V_j$ satisfy the condition $\delta(V_j) < \delta(V_q)$. This formula helps to construct the community-generated ranking label $R(l_c)$. For instance, when $n = 3$, we have a query-generated ranking list $l_q = \{V_1, V_2, V_3\}$ and its related ranking label $R(l_q) = \{1, 2, 3\}$. Given a chromosome where the community of $V_1$ and $V_2$ is the same and selected before $V_3$, we can generate the related community ranking label $R(l_c) = \{1, 1, 3\}$ with the same vertex sequence of $l_q$.

Then, we define the fitness function $f(\cdot)$ to evaluate the chromosome $ch_k$. As we have the query-generated ranking label $R(l_q)$ (ground truth) and community-generated ranking label $R(l_c)$ from $ch_k$, we calculate their Kendall's $\tau$ correlation coefficient as the fitness score $f(ch_k)$ of chromosome $ch_k$ where higher score means the chromosome $ch_k$ can generate better personalized communities to meet with user need.

$$f(ch_k) = 1 - \frac{\sum_{i=1}^n R(l_{ci}) \cdot R(l_{qi})}{\sum_{i=1}^n R(l_{ci})^2 \cdot \sum_{i=1}^n R(l_{qi})^2} \quad (9)$$

where $R(l_{ci})$ is the $i_{th}$ vertex ranking in community-generated ranking label $R(l_c)$ and $R(l_{qi})$ is the $i_{th}$ vertex ranking in query-generated ranking label $R(l_q)$. Kendall's $\tau$ is a widely used metric to evaluate the correlation between two lists where higher score means stronger correlation. Thus, higher fitness score reflects that the generated community ranking ($R(l_c)$) can better meet with user need ($R(l_q)$).

Moreover, it is clear that the fitness function aims to separate all top $n$ vertices in different communities to get the optimal case. It matches our research goal to generate high resolution communities on vertices which are more relevant to user need. As the number of community is a given number $K$, it also leads to a coarser manner partition on the remaining less relevant vertices. On the other hand, the binary community tree $T_b(N^b, L^b)$ and the Cutting Rule 4 naturally preserve the community structure and unite the most relevant vertices in the same community. Hence the whole genetic approach is a gambling process. The final chromosome result is the equilibrium case to detect communities both contain graph topological structure and meet with user need.

### 3.3.4 Selection.

We select the superior chromosomes from current chromosome population based on their fitness scores. The probability that the $k_{th}$ chromosome $ch_k$ is picked can be calculated via the Softmax normalization function $p(ch_i) = \frac{exp(f(ch_k))}{\sum_{i=1}^P exp(f(ch_i))}$. Then, the Fitness Proportionate Selection method [12] is applied to randomly select $P$ chromosomes into chromosome pairs based on probability distribution. In order to enhance optimization efficiency, we also use elitism selection to ensure the best chromosome in the current generation will always be selected to the next generation.

### 3.3.5 Crossover.

To reach global optimum community partition efficiently, given a pair of chromosomes, the crossover operation can randomly exchange part of the genes in both chromosomes to produce a new pair of chromosomes with a certain crossover rate.

In order to make sure that the newly generated chromosomes meet the cutting rules, an **Exchange Rule** is defined to restrict gene exchange: If gene $g$ contains link $L_g^b$, $g$ can't do crossover process with genes that contain either link $L_g^b$ or its sibling link $L_g^{b'}$. This rule can help avoid having duplicated links or sibling links stored together in the newly generated chromosome (To satisfy Cutting rule 3).

After $m$ random numbers are selected from $\{1, 2, ..., K - 1\}$ as exchanged gene position indexes, genes located in the chosen positions of two chromosomes will exchange the stored link restricted by the Exchange Rule.

### 3.3.6 Mutation.

Mutation operation is applied to avoid local optimization. If a chromosome is chosen to mutate, a gene within the chromosome will be randomly picked, and its stored link will be changed to another link restricted by the Exchange Rules. An example is illustrated in Figure 2(b) where the link stored in the second gene is changed from "001" to "01".

### 3.3.7 Termination.

After $T$ iterations, the whole process stops and the current best chromosome is retrieved as the final result. Choosing the number of $T$ is dependent on the task. In order to decode the final chromosome to the related community partition, all genes in the chromosome are sorted in an ascending order based on the binary code of their stored cut link. Vertices whose binary codes start with the same cut link's binary code will be assigned to the same community label. And its later assigned community label can overwrite the previous assigned community label. For instance, if there are a vertex with binary code "0011" and two cut links with binary code "00" and "001", the vertex will be assigned to a community label "00" first, and its community label is overwritten by "001" afterwards. The Termination step in Figure 2(b) also illustrates a vivid example. In this way, the binary code of the binary community tree can help to decode the final chromosome into communities in an efficient way.

## 3.4 Distributed gPCD

To enhance the online step efficiency, a MapReduce framework [11] is utilized to enable the distributed genetic evolution. Figure 3 depicts the personalized community detection under a MapReduce framework. The chromosome collection is either originally initialized from binary community tree or obtained from the last generation. It contains the whole chromosome population in the central depository. In its first "Splitter" process, all chromosomes are split into $M$ groups based on their hash values and sent out to related $M$ Mappers to calculate the "Fitness" scores. In the same Mapper, after all chromosomes are assigned fitness scores, based on their scores, a Combiner groups all chromosomes together and random select equal number of chromosomes with duplicated as the "Selection" step. All the selected chromosomes are sent to $R$ reducers (we set $R = M$ arbitrarily in order to better represent time complexity) to form pairs for the "Crossover" and "Mutation" step,
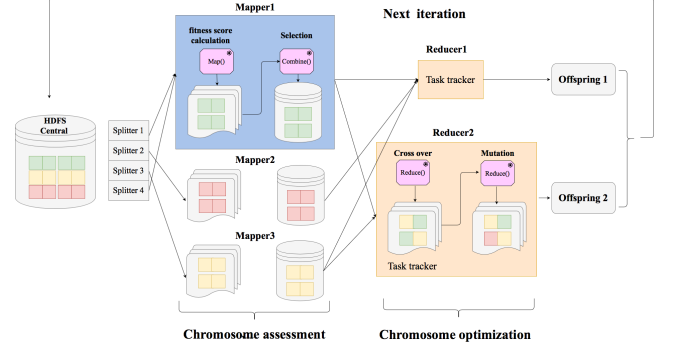


**Figure 3: Online parallel computing process on Hadoop Distributed File System (HDFS)**

calculate new chromosome offsprings for the next generation and store them back to the central repository.

The complexity of the proposed algorithm is $O(2^d KP)$ without parallel computing and $O(\frac{2^d KP}{M})$ with parallel computing, where $d$ denotes the upper bound where the cut links are restricted in the top $d$ depth of the binary community tree $T_b(N^b, L^b)$; $K$ denotes the community number; $P$ denotes the initialized population size of the genetic algorithm and $M$ denotes the number of Mappers/Reducers in parallel environment. As all the parameters are considerably small (compared with the node/edge size in the original graph), the whole process runs very fast to retrieve the final community partition.

## 4 EXPERIMENTS

### 4.1 Datasets

#### 4.1.1 Datasets Description.

Table 2 shows the statistics of the two datasets. The scholarly graph is unweighted and directed, while the music graph is a weighted and undirected.

| Dataset | Node description | | Edge description | |
|---------|------|------|------|------|
| | Type | Size | Type | Size |
| Scholarly | paper | 166,170 | citation | 750,181 |
| Music | song | 145,203 | co-listening | 1,172,525 |

**Table 2: Dataset Description**

**Scholarly Graph:** It contains academic publications with metadata extracted from ACM Digital Library. From the dataset, we build the experimental graph via paper citation relationship. Each vertex in the graph represents a paper, and if a paper cites another paper, there will be an edge linking the two. Our model aims to detect personalized communities on the scholarly graph for authors. For each author in the dataset, we represent his/her need in two ways: their previous publications (text-based query) and cited paper history (vertex-based query).

**Music Graph:** It contains user listening histories and user-generated playlists from an online music streaming service, Xiami. We create

(a) Depth $d$ in Citation model　(b) Depth $d$ in Keyword model　(c) Depth $d$ in Listening model　(d) Iteration $T$ in all models　(e) User searching preference $\lambda$ in all models
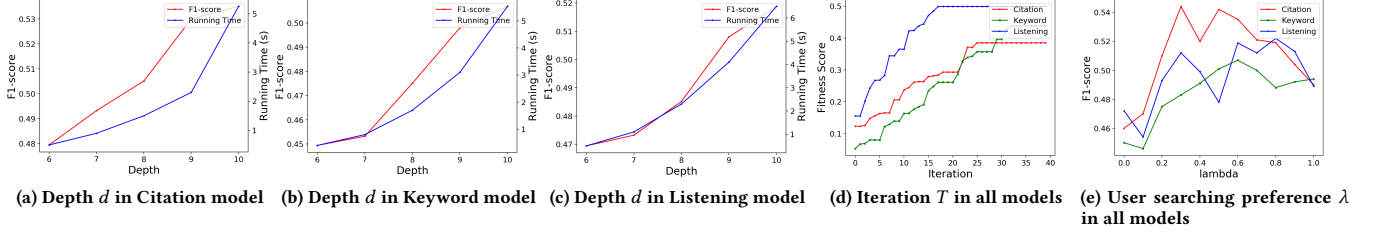
**Figure 4: Parameter effects on model performance**

a music graph with songs as the vertices and co-listening relationship as the edges. If two songs appear in the same user's listening history, there will be an edge linking them two. For users in the music dataset, we represent their music tastes (user need) from the songs in their listening history.

### 4.1.2 Ground Truth Construction.

The ground truth for the two datasets are generated based on each user's publishing/ citing/ listening history:

For the scholarly dataset, the references of 112 random sampled papers are manually annotated from their literature reviews where authors summarize previous works. Different paragraphs (or sub-sections) in the literature review typically focus on separate but coherent topics while the same paragraph talks about the same topic [41]. Based on this assumption, the papers cited in the same paragraph/sub-section naturally form a community with high resolution. To ensure each paper's cited papers form enough communities and each community contains enough papers, only papers with no fewer than three topics and all of whose communities have at least five papers are kept. After applying all these filters, 101 papers are left for evaluation.

For the music dataset, each user has several self-generated playlists. The songs in each playlist should contain a coherent theme. To avoid the playlists sharing mutually exclusive themes with other playlists, a Jaccard similarity check is applied on any pair of playlists created by the same user. If a user has at least two highly correlated playlists (Jaccard coefficient between them is above 0.5), we remove one of the playlists. Each playlist forms a separate community. Furthermore, to ensure the number of playlist and playlist size are both large enough, only users with at least three playlists and each playlist contains at least five songs are kept. In the end, there are 117 users who meet the above criteria.

In this paper, as all communities constructed in the ground truth are relevant communities with high resolution for users, our task is generating personalized communities to reconstruct the ground truth on two different datasets with vertex- and text-based user need. We share our code in Github[1].

## 4.2 Settings

### 4.2.1 Metrics & Parameter Settings.

F1-score (F1), Rand index (Rand), Jaccard index (Jaccard) and running time are reported as the evaluation metrics in this paper. Based on empirical studies, population size $P$ is 100. Crossover rate is 95%. Mutation rate is 1%. The maximum depth of binary

---

[1] https://github.com/RoyZhengGao/gPCD

community tree $d$ is 10. The number of iteration $T$ is 30. User searching preference $\lambda$ is 0.6. Community size $K$ is 50. The number of Mappers/ Reducers for parallelization $M$ is 50. The number of top vertices to construct pseudo ground truth $n$ is 10. Parameters in Infomap and Node2vec are both the default settings in their original papers.

### 4.2.2 Baselines.

Considering both efficacy and efficiency, we select eight widely used user-independent community detection models. Ideally, to achieve personalized community detection, user-independent models should run on each user separately by assigning higher weights on user related edges. Thus, their time complexity should be only compared with our online step time complexity as our offline step is independent with user numbers. In this paper, to run baselines within acceptable time, we report their user-independent community results as the average performance.

- **Spinglass**: Spinglass [10] constructs communities by minimizing the Hamiltonian score on signed graphs.
- **Fast Greedy** (FG): Fast Greedy [7] is a greedy search method to get the maximized modularity for community detection.
- **Louvain**: Louvain [4] is an agglomerative method to construct communities in a bottom-up manner guided by modularity.
- **Walktrap**: Walktrap [33] detects communities based on the fact that a random walker tends to be trapped in dense part of a network.
- **Infomap**: Infomap [35] generates communities by simulating a random walker wandering on the graph and indexing the description length of his random walk path via multilevel codebooks.
- **Bigclam**: Bigclam [38] generates overlapping communities via a non-negative matrix factorization approach.
- **DeepWalk**: DeepWalk [32] generates node embeddings via random walks and utilizes K-means on node embeddings to detect communitis.
- **Node2vec**: Node2vec [19] is an extended version of DeepWalk with a refined random walk strategy.

## 4.3 Results

### 4.3.1 Evaluation Results.

There are two scenarios to construct user need. For the scholarly graph, including a citation (vertex-based query) model and a keyword (text-based query) model. In the citation model, for each user, we first extract all the papers he/she cited before, then use

their centroid embedding as user need vector $\vec{I}$. In the keyword model, for each user (author), we first extract all keywords he/she used in all previous papers to form a text query. Then we retrieve the top 100 relevant papers given the query based on probability language model with Dirichlet smoothing [24]. Finally, we average those retrieved papers' vectors as the user need vector $\vec{I}$.

| Model | Scholarly Graph | | | Music Graph | | |
|---|---|---|---|---|---|---|
| | F1 | Rand | Jaccard | F1 | Rand | Jaccard |
| Spinglass | 0.4294 | 0.4149 | 0.3593 | 0.4282 | 0.5317 | 0.2823 |
| FG | 0.4290 | 0.3852 | 0.3735 | 0.4645 | 0.4100 | 0.3070 |
| Louvain | 0.4417 | 0.4546 | 0.3627 | 0.1832 | 0.4201 | 0.1174 |
| Walktrap | 0.4304 | 0.3777 | 0.3777 | 0.3999 | 0.3507 | 0.3490 |
| Infomap | 0.4436 | 0.4165 | 0.3606 | 0.2147 | **0.6074** | 0.1344 |
| Bigclam | 0.2314 | 0.2572 | 0.1348 | 0.1499 | 0.2078 | 0.1227 |
| DeepWalk | 0.3904 | 0.3237 | 0.3234 | 0.3535 | 0.3253 | 0.3001 |
| Node2vec | 0.4001 | 0.3472 | 0.3433 | 0.4122 | 0.4101 | 0.3087 |
| gPCD-Citation | **0.5351*** | **0.4551*** | **0.4086*** | - | - | - |
| gPCD-Keyword | 0.5069 | 0.4114 | 0.3708 | - | - | - |
| gPCD-Listening | - | - | - | **0.5188*** | 0.5865 | **0.3550*** |

Note: "*" means the p-value through a pairwise t-test is smaller than 0.001.
**Table 3: Personalized Community Evaluation on gPCD and Baselines.**

For the music task, text information is not available. The centroid embedding of the songs listened to by a target user are taken as user need.

**Community Accuracy**: We compare the average performance of our model on all testing users with baselines. Table 3 shows the detailed metrics. When running on the Scholarly graph, both Citation model and Keyword model can achieve around 10% increase on F1-score compared with all baselines. Citation model also performs the best in Rand Index and Jaccard Index. For Music graph, our Listening model also has a significant improvement on F1-score and Jaccard Index. Although it has similar performance on Rand Index compared with Infomap, we believe our model in fact works much better due to the Infomap's poor performance on the rest two metrics. Moreover, we apply pairwise t-tests [9] for all metrics on all testing users. All metrics' p-values in Citation model and the p-values of F1-score and Jaccard Index in Listening model are all smaller than 0.001, which means the improvements of our model performance are significant compared with baselines.

**Running Time Comparison**: Table 4 shows both the theoretical time complexity and real running time. To represent baseline algorithms' time complexity, "$V$" refers to the vertex number and "$E$" refers to the edge number in the graph $G(V, E)$. For some models (FG, Walktrap, and Infomap.), their specific time complexities are officially mentioned in the original papers. The time complexity of Louvain and Bigclam are roughly estimated in the original papers as well but those papers don't mention specific numbers. For Spinglass, DeepWalk and Node2vec, we can't find the exact time complexity in existing studies. Hence in this paper, we arbitrarily assign labels based on the their real running speed. Considering the running time, all the baseline algorithms run relatively fast except for the Spinglass algorithm. However, compared with all other models, the distributed gPCD always performs the fastest. Its

| Model | Time Complexity | Scholarly Graph (s) | Music Graph (s) |
|---|---|---|---|
| Spinglass | very slow | 12548.68 | 10372.17 |
| FG | $O(|V|log^2|V|)$ | 280.60 | 272.34 |
| Louvain | linear | 80.01 | 63.02 |
| Walktrap | $O(|V|^2log|V|)$ | 638.44 | 503.24 |
| Infomap | $O(|V|(|V| + |E|))$ | 501.79 | 425.63 |
| Bigclam | linear | 57.01 | 112.43 |
| DeepWalk | fast | 720.56 | 688.32 |
| Node2vec | slow | 3508.44 | 3100.12 |
| gPCD | $O(\frac{2^d KP}{M})$ | **5.25** | **6.50** |

**Table 4: Running time analysis on gPCD and all baselines in seconds (s).**

real running time of is less than one-tenth of the fastest baseline's running time.

### 4.3.2 Parameter Analysis.
We show how three parameters can affect our gPCD model performance in this section. They are the depth of the binary community tree $d$, genetic iteration number $T$ and user searching preference $\lambda$. Figure 4 show the overall impacts of all tuned parameters.

**Depth on the Tree**: Figure 4(a) to Figure 4(c) show how the depth of the binary community tree affects the model performance in accuracy and efficiency. From the figures, larger depth leads to a better personalized community detection result, while causes an exponential running time increase at the same time. Based on empirical studies, the upper bound of the depth is set to be 10 in this paper. While the depth selection may varies based on different graph sizes.

**Convergence Analysis**: We observe the best chromosome updates in 40 iterations. From Figure 4(d), we can see the fitness score start to be stable after the 30 iterations, which means the best chromosome is no longer changed after around 30 iterations. Thus, we set $T = 30$ as the default iteration number in our approach.

**Searching Preference**: In Figure 4(e), $\lambda$ reflects the user searching preference whether he/she wants to explore new information or avoid redundant information. By selecting $\lambda$ from 0 to 1, we find the F1-score are not very stable or have a clear correlation with $\lambda$. Based on the empirical experiments, we achieve the best performance on three models when $\lambda = 0.6$.

## 5 CONCLUSION
To our best knowledge, the personalized community detection task proposed in this paper is the first attempt to address on detecting communities with different resolutions to meet with user need. To solve this task, we propose a model with an offline binary community tree construction step and an online genetic pruning step. A distributed version of our model is also deployed to accelerate running efficiency. Extensive experiments on two different datasets shows our model outperforms all baselines in terms of accuracy and efficiency. However, the current approach still partially relies on existing models such as Infomap and Node2vec. In the next step, we will design our own user-independent community detection and vertex & user need representation models so that we can achieve a more integrated and unified model.

# REFERENCES

[1] Emmanuel Abbe. 2018. Community Detection and Stochastic Block Models: Recent Developments. *Journal of Machine Learning Research* 18, 177 (2018), 1–86.

[2] Hassan Abbas Abdelbary, Abeer Mohamed ElKorany, and Reem Bahgat. 2014. Utilizing deep learning for content-based community detection. In *Science and Information Conference (SAI), 2014*. IEEE, 777–784.

[3] Austin R Benson, David F Gleich, and Jure Leskovec. 2016. Higher-order organization of complex networks. *Science* 353, 6295 (2016), 163–166.

[4] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.

[5] Tanmoy Chakraborty, Ayushi Dalmia, Animesh Mukherjee, and Niloy Ganguly. 2017. Metrics for community analysis: A survey. *ACM Computing Surveys (CSUR)* 50, 4 (2017), 54.

[6] Benjamin Paul Chamberlain, Josh Levy-Kramer, Clive Humby, and Marc Peter Deisenroth. 2018. Real-time community detection in full social networks on a laptop. *PloS one* 13, 1 (2018), e0188702.

[7] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. 2004. Finding community structure in very large networks. *Physical review E* 70, 6 (2004), 066111.

[8] Pasquale De Meo, Emilio Ferrara, Giacomo Fiumara, and Angela Ricciardello. 2012. A novel measure of edge centrality in social networks. *Knowledge-based systems* 30 (2012), 136–150.

[9] Ben Derrick, Antonia Broad, Deirdre Toher, and Paul White. 2017. The impact of an extreme observation in a paired samples design. *Metodološki Zvezki-Advances in Methodology and Statistics* 14, 2 (2017), 1–17.

[10] Eric Eaton and Rachael Mansbach. 2012. A spin-glass model for semi-supervised community detection.. In *AAAI*. 900–906.

[11] Filomena Ferrucci, M Kechadi, Pasquale Salza, Federica Sarro, et al. 2013. A framework for genetic algorithms based on hadoop. *arXiv preprint arXiv:1312.0086* (2013).

[12] David B Fogel. 1997. Evolutionary algorithms in theory and practice.

[13] Santo Fortunato. 2010. Community detection in graphs. *Physics reports* 486, 3 (2010), 75–174.

[14] Santo Fortunato and Darko Hric. 2016. Community detection in networks: A user guide. *Physics Reports* 659 (2016), 1–44.

[15] Mohadeseh Ganji, James Bailey, and Peter J Stuckey. 2018. Lagrangian constrained community detection. In *proc. of AAAI Conf. on Artificial Intelligence. AAAI*.

[16] Zheng Gao, Gang Fu, Chunping Ouyang, Satoshi Tsutsui, Xiaozhong Liu, and Ying Ding. 2018. edge2vec: Learning Node Representation Using Edge Semantics. *arXiv preprint arXiv:1809.02269* (2018).

[17] Zheng Gao and Xiaozhong Liu. 2017. Personalized community detection in scholarly network. *iConference 2017 Proceedings Vol. 2* (2017).

[18] Michelle Girvan and Mark EJ Newman. 2002. Community structure in social and biological networks. *Proceedings of the national academy of sciences* 99, 12 (2002), 7821–7826.

[19] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 855–864.

[20] Ekta Gujral and Evangelos E Papalexakis. 2018. SMACD: Semi-supervised Multi-Aspect Community Detection. In *Proceedings of the 2018 SIAM International Conference on Data Mining*. SIAM, 702–710.

[21] Manish Gupta, Jing Gao, Xifeng Yan, Hasan Cam, and Jiawei Han. 2014. Top-k interesting subgraph discovery in information networks. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*. IEEE, 820–831.

[22] Brian Karrer and Mark EJ Newman. 2011. Stochastic blockmodels and community structure in networks. *Physical Review E* 83, 1 (2011), 016107.

[23] Theodoros Lappas, Kun Liu, and Evimaria Terzi. 2009. Finding a team of experts in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 467–476.

[24] Ray R Larson. 2010. Introduction to information retrieval.

[25] Marc T Law, Raquel Urtasun, and Richard S Zemel. 2017. Deep spectral clustering learning. In *International Conference on Machine Learning*. 1985–1994.

[26] Yafang Li, Caiyan Jia, Jianqiang Li, Xiaoyang Wang, and Jian Yu. 2018. Enhanced semi-supervised community detection with active node and link selection. *Physica A: Statistical Mechanics and its Applications* (2018).

[27] Xiaozhong Liu, Xing Yu, Zheng Gao, Tian Xia, and Johan Bollen. 2016. Comparing community-based information adoption and diffusion across different microblogging sites. In *Proceedings of the 27th ACM Conference on Hypertext and Social Media*. ACM, 103–112.

[28] Xiaoke Ma, Di Dong, and Quan Wang. 2019. Community detection in multi-layer networks using joint nonnegative matrix factorization. *IEEE Transactions on Knowledge and Data Engineering* 31, 2 (2019), 273–286.

[29] Xiaoke Ma, Lin Gao, Xuerong Yong, and Lidong Fu. 2010. Semi-supervised clustering algorithm for community structure detection in complex networks. *Physica A: Statistical Mechanics and its Applications* 389, 1 (2010), 187–197.

[30] Xiaoke Ma, Bingbo Wang, and Liang Yu. 2018. Semi-supervised spectral algorithms for community detection in complex networks based on equivalence of clustering methods. *Physica A: Statistical Mechanics and its Applications* 490 (2018), 786–802.

[31] Mark EJ Newman. 2006. Modularity and community structure in networks. *Proceedings of the national academy of sciences* 103, 23 (2006), 8577–8582.

[32] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[33] Pascal Pons and Matthieu Latapy. 2005. Computing communities in large networks using random walks. In *International Symposium on Computer and Information Sciences*. Springer, 284–293.

[34] Martin Rosvall and Carl T Bergstrom. 2008. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences* 105, 4 (2008), 1118–1123.

[35] Martin Rosvall and Carl T Bergstrom. 2011. Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems. *PloS one* 6, 4 (2011), e18209.

[36] Wenjun Wang, Xiao Liu, Pengfei Jiao, Xue Chen, and Di Jin. 2018. A Unified Weakly Supervised Framework for Community Detection and Semantic Matching. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 218–230.

[37] Tian Xia, Xing Yu, Zheng Gao, Yijun Gu, and Xiaozhong Liu. 2017. Internal/External information access and information diffusion in social media. *iConference 2017 Proceedings Vol. 2* (2017).

[38] Jaewon Yang and Jure Leskovec. 2013. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proceedings of the sixth ACM international conference on Web search and data mining*. ACM, 587–596.

[39] Liang Yang, Xiaochun Cao, Dongxiao He, Chuan Wang, Xiao Wang, and Weixiong Zhang. 2016. Modularity Based Community Detection with Deep Learning.. In *IJCAI*. 2252–2258.

[40] Chenwei Zhang, Zheng Gao, and Xiaozhong Liu. 2016. How others affect your Twitter# hashtag adoption? Examination of community-based and context-based information diffusion in Twitter. *IConference 2016 Proceedings* (2016).

[41] Lin Zhang and Xiaozhong Liu. 2006. Synopsizing "literature review" for scientific publications. *IConference 2016 Proceedings* (2006).

[42] Yuanyuan Zhu, Lu Qin, Jeffrey Xu Yu, and Hong Cheng. 2012. Finding top-k similar graphs in graph databases. In *Proceedings of the 15th International Conference on Extending Database Technology*. ACM, 456–467.

[43] Lei Zou, Lei Chen, and Yansheng Lu. 2007. Top-k subgraph matching query in a large graph. In *Proceedings of the ACM first Ph. D. workshop in CIKM*. ACM, 139–146.