IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 11, 2022, accepted April 20, 2022, date of publication April 25, 2022, date of current version May 5, 2022.

*Digital Object Identifier 10.1109/ACCESS.2022.3170476*

# A Novel Efficient Method for Tracking Evolution of Communities in Dynamic Networks

**ARZUM KARATAS, (Member, IEEE), AND SERAP SAHIN, (Member, IEEE)**
Department of Computer Engineering, İzmir Institute of Technology, 35430 İzmir, Turkey

Corresponding author: Serap Sahin (serapsahin@iyte.edu.tr)

This work was supported in part by the Turkey's Council of Higher Education (YÖK) through 100/2000 Program, and in part by the Scientific and Technological Research Council of Turkey (TÜBİTAK) through 2211-C Program.

**ABSTRACT** Tracking community evolution can provide insights into significant changes in community interaction patterns, promote the understanding of structural changes, and predict the evolutionary behavior of networks. Therefore, it is a fundamental component of decision-making mechanisms in many fields such as marketing, public health, criminology, etc. However, in this problem domain, it is an open challenge to capture all possible events with high accuracy, memory efficiency, and reasonable execution times under a single solution. To address this gap, we propose a novel method for tracking the evolution of communities (TREC). TREC efficiently detects similar communities through a combination of Locality Sensitive Hashing and Minhashing. We provide experimental evidence on four benchmark datasets and real dynamic datasets such as AS, DBLP, Yelp, and Digg and compare them with the baseline work. The results show that TREC achieves an accuracy of about 98%, has a minimal space requirement, and is very close to the best performing work in terms of time complexity. Moreover, it can track all event types in a single solution.

**INDEX TERMS** Community evolution, community tracking, locality sensitive hashing with minhashing, LSH with minhashing.

## I. INTRODUCTION

Many real-world networks, biological networks, and social networks can be represented by graphs. These networks contain a dynamic community structure that exhibits natural partitioning. Each of these partitions defines a community [1] naturally by groups of vertices with dense connections internally and sparser connections from other groups in the network.Community detection is identifying these partitions in the network. The size of communities in the network can grow or shrink, and new communities can emerge or disappear as relationships or interactions change over time.

Detecting communities and tracking their evolution over time is key to understanding relationships between groups, which is essential for assessing future relationships. These insights provide valuable information for decision support mechanisms in many fields such as criminology [2], targeted marketing and smart advertising [3], recommender systems [4], sociology [5], epidemiology [6], brain networks [7], and community evolution prediction [8]–[12]. Most of these applications require accurate results, reasonable memory consumption and execution time, and work with large dynamic networks. To detect the evolution of communities in dynamic networks, the network data is first decomposed into time steps, and then a community detection method is applied to the data of each time step to reveal the underlying community structure. Later, the evolution of a community is tracked by finding its similar community(s) with community matching between ascending time steps. Then, evolution chains are created to list the communities during their lifespan. Then, evolution events can be labeled with types of evolution events. This solution approach can be classified under the independent community detection and matching approach in the taxonomy of Dakiche *et al.* [13], which classifies methods for tracking community evolution according to their methodological principles.

From examining related work from 2007 to 2021 (in the class of independent community detection and matching approach), some functional criteria for comparison emerge, such as tracking nonconsecutive communities, k-merge and k-splits, and identifying all evolutionary types. The recent works such as the method of Tajeuna *et al.* [14] and ICEM [15] satisfy all the functional criteria, while the recent work WECEM [16] cannot track nonconsecutive

The associate editor coordinating the review of this manuscript and approving it for publication was Dominik Strzalka.

community evolution. In addition, performance criteria such as accuracy, memory consumption, and execution time must also be considered. They all consider accuracy, but only one of them (ICEM [15]) considers execution time efficiency, while none of the existing works consider space efficiency. However, space efficiency is important since the method should be executed on common computing devices such as computers and laptops in general qualifications. In this way, parallelizing the implementation or using cloud solutions is not necessarily required. This has motivated us to conduct this study. In this work, a novel, accurate and space-efficient method with reasonable execution time TREC (Tracking Evolution of Communities) is proposed to track the consecutive and nonconsecutive evolution of communities. This study uses a combination of two probabilistic techniques such as Minhashing [17] and Locality Sensitive Hashing (LSH) [18].

Minhashing is a technique for converting large sets into small fixed signatures while preserving the similarities between the sets. LSH is a technique to find approximate near neighbors with high probability without pairwise matching. In this study, minhashing is used to extract community signatures to represent communities efficiently and LSH is modified to find similar communities with high probability without brute force comparison of community pairs. That is, LSH is used together with minhashing to reduce the execution time and memory cost in community matching. Compared to existing work, extracting signatures of communities by minhashing consumes less memory and searching for similar communities is accelerated by filtering out the meaningless community comparison by LSH. The way LSH with minhashing technique used in our study is explained in detail in section IV.

Since the latest method WECEM [16] cannot track nonconsecutive community evolutions, we do not consider it as one of TREC's competitors and do not include WECEM in our performance analysis. In the performance analysis, we find that TREC is (i) a very accurate method like that of Tajeuna *et al.* [14], (ii) the most space-efficient method in the literature, and (iii) has a reasonable execution time, which achieves the second best value among related works after ICEM [15]. Therefore, it can be said that TREC could be the most suitable method for tracking community developments in low memory environments.

The contributions of this study are summarized below:

*Space-efficient community representation:* The community signatures are extracted using minhashing technique to consume less memory than the real communities need. In this way, communities are stored in main memory as short fixed-length sketches.

*Efficient search for similar communities:* LSH is adapted to efficiently identify similar communities based on community signatures for a queried community signature.

*Tracking algorithm:* A naive tracking algorithm is provided to process LSH with minhashing results to track the evolution of consecutive and nonconsecutive communities.

*TREC method:* A new community evolution tracking method is presented that is accurate, space-efficient, and executable in reasonable time. It can track both consecutive and nonconsecutive evolution and identifies all evolutionary events.

*Complexity analysis:* A time and space complexity analysis for the TREC method is performed to show the computational limitations of the method to potential users and researchers in the field.

*Software profiling:* The memory requirements and execution time of TREC are measured. This shows the real-time profile of the TREC method.

The structure of the paper is as follows. Section II provides introductory information and the problem statement. Section III gives an overview of related work. Section IV presents the TREC method in detail. Section V presents the experimental study and discusses the evaluation results of TREC in terms of algorithmic analysis and experimental analysis. Section VI provides concluding remarks and some suggestions for future work.

## II. BACKGROUND AND PROBLEM FORMULATION

In this section, introductory information and the problem statement are presented. Subsection II.A explains basic concepts and definitions of the problem domain. Then, in subsection II.B, the problem formulation is given. Next, in subsection II.C, the evaluation criteria for tracking methods are presented. Finally, in subsection II.D, the relationship between the similarity threshold of community signatures and LSH is presented.

### A. CONCEPTS AND DEFINITIONS

Depending on their nature, networks can be represented as static or dynamic graphs. While a static graph is just a snapshot of the network for a given time interval, a dynamic graph is an ordered sequence of static graphs over time to capture the temporal properties of the network. In this study, all dynamic networks used are unweighted and undirected. Both the vertices and the edges are not static and we assume that they can change over time. Dynamic networks inherently contain a community structure. The term ''community'' is generally defined as a subgroup whose connections within the network are tight but only loosely connected to the rest of the network, and which has at least three members. Community structures can be of different natures, such as overlapping and nonoverlapping communities. In an overlapping community structure, a member may belong to more than one community. In a nonoverlapping community structure, each member belongs to only one community. Community detection divides a network graph into subgroups based on the densities of the edge connections among vertices in the network. For instance, in social networks, the vertices of a subgroup are members of a community, and edges between two vertices in these subgroups define a 'friendship' relation. The easiest way to track the evolution of communities is first to discover

communities and then match them based on the similarity of their members over time.

The Jaccard similarity (JS) is used to determine the similarity of members between the two communities by (1).

$$JS = \frac{C_{t1}^{i} \cap C_{t2}^{k}}{C_{t1}^{i} \cup C_{t2}^{k}}. \quad (1)$$

where $C_{t1}^{i}$ and $C_{t2}^{k}$ are the compared communities, $(i, k)$ are the numbers identifying the communities, and $(t1, t2)$ are the observed time steps of these communities. JS takes real values between 0 and 1, where 0 means that the communities are completely different and 1 means that they are completely the same with respect to their members. The $\lambda$ is the similarity threshold between a pair of communities. If $JS \geq \lambda$, then two communities are accepted as similar. In other words, $\lambda$ is the member-based similarity threshold of JS for two communities.

The evolution of a community is represented by the sequence of tracked relationships between communities (e.g., matching communities) over time steps. For example, the evolution of community $C_t^1$ from time $t = 1$ to $t = 5$ can be represented as $C_t^1 = C_1^1, C_2^{577}, \cdots, C_5^{2800}$. The problem of tracking communities is thus defined as detecting a set of similar communities at specific time steps and tracking their evolutionary behavior over the lifetime of a dynamic network.

The relationships between communities may change over time, so a community may experience some critical events. In this study, the rate of change in community size is assumed to be 5% to determine the event type such as "continue", "growth" or "shrink". All critical events are defined in Table 1 and visualized in Fig. 1. Note that the evolution events shown in Fig. 1 occur only at two time steps i and j, where $i < j$ represents two sets of matching communities and the matches between communities can be one-to-one, one-to-many, or many-to-one.

### B. PROBLEM FORMULATION

Let $G_t = (V_t, E_t)$ be a graph representing a static network, where "$V_t$" is the set of vertices and "$E_t$" is the set of edges at a particular time step $t$. A dynamic network G can be denoted as a sequence of static networks such as $G = G_1, G_2, \cdots, G_{timeStepCount}$ where $t = 1, 2, \cdots, timeStepCount$. A community is a subset of densely connected vertices of each time graph $G_t$ while it is only loosely connected to the rest of $G_t$. There may be a number of $k$ distinct communities belonging to the same $G_t$. A community detection method partitions the time graph $G_t$ into densely connected subgraphs (e.g., communities) such $C_t = C_t1, C_t2, \cdots, C_tk$ where each community $C_t^i \in C_t, i = 1, \cdots, k$ with vertex set and edge set of each community $C_t^i = (V_t^i, E_t^i)$ as $C_t^i \subseteq G_t$.

### C. EVALUATION CRITERIA FOR THE TRACKING METHODS

The methods for tracking evolution of communities in the Related Work section are characterized using some functional criteria as follows. Based on this characterization, the methods are compared and functionally evaluated.

*Criterion #1 Community structure:* Community structure such as overlapping or nonoverlapping; the type of network affects the community detection algorithm and the community tracking method used.

*Criterion #2 Tracking nonconsecutive evolutions:* The ability to track nonconsecutive communities. If a community is observed at each time step, that community evolves

**TABLE 1. Event types for evolution of communities.**

| Definitions of Event Types for Evolution of Communities | Reference at Fig. 1 |
|---|---|
| *Form/birth:* a new community $C_{t_i}$ forming at time $t_i$ is born.<br><br>$birth(C_{tj}^a) = true$;<br>if $JS(C_{ti}^*, C_{tj}^a) < \lambda$ for $\forall C_{t_i}^* \in G$ at time $t_i$ and $t_i < t_j$ and $\lvert V_{C_{ti}^a}\rvert \geq 3$. | (a) |
| *Growth:* new members can join the community or some existing members can move among communities over time in graph $G$; hence, some of the communities can grow.<br><br>$growth(C_{ti}^a) = true$;<br>$if \exists C_{tj}^b \in G$ at time $t_j$ and $t_i < t_j$ and $JS(C_{ti}^a, C_{tj}^b) \geq \lambda$ and $1.05 \times \lvert V_{C_{ti}^a}\rvert \leq \lvert V_{C_{tj}^b}\rvert$. | (b) |
| *Continue:* communities can continue their life either without any change or with changes within tiny upper or lower limits as 0.05 change rate in community size.<br><br>$continue(C_{ti}^a) = true$;<br>$if \exists C_{tj}^b \in G$ at time $t_j$ and $t_i < t_j$ and $JS(C_{ti}^a, C_{tj}^b) \geq \lambda$ and $0.95 \times \lvert V_{C_{ti}^a}\rvert < \lvert V_{C_{tj}^b}\rvert < 1.05 \times \lvert V_{C_{ti}^a}\rvert$. | (c) |
| *Shrink:* some of the members can leave a community and cause it to shrink.<br><br>$shrink(C_{ti}^a) = true$;<br>if $\exists C_{tj}^b \in G$ at time $t_j$ and $t_i < t_j$ and $JS(C_{ti}^a, C_{tj}^b) \geq \lambda$, and $\lvert V_{C_{tj}^b}\rvert \leq 0.95 \times \lvert V_{C_{ti}^a}\rvert$ | (d) |
| *Split:* one community can split into sub communities, if similarity threshold $\lambda$ is satisfied between community and the set of sub-communities at the following time step.<br><br>$split(C_{ti}^a) = true$;<br>$if S_{C_{t_i}^*} = C_{t_j}^1, C_{t_j}^2, \cdots, C_{t_j}^m$ for $C_{ti}^a$ at time $t_i$ and $t_i < t_j$, and $\forall C_{t_j}^* \in S_{C_{t_i}^*}$, and $JS(C_{ti}^a, C_{tj}^*) \geq \lambda$ | (e) |
| *Merge:*communities can form a new and bigger community by merging. A set of communities $S_{C_{t_i}^*} = C_{t_i}^1, C_{t_i}^2, C_{t_i}^3, \cdots, C_{t_i}^n$ merge, and form a community $\exists C_{t_j}^b \in G$ at time $t_j$, and $t_j > t_i$ The similarity threshold $\lambda$ exceeds each community of $S_{C_{t_i}}$ and $C_{tj}^b$.<br><br>$merge(S_{C_{t_i}^a}) = true$;<br>$if \exists S_{C_{t_i}} = C_{t_i}^1, C_{t_i}^2, C_{t_i}^3, \cdots, C_{t_i}^n$ at time $t_i$, $t_i < t_j$,and $\exists C_{tj}^b \in G, \forall C_{t_i}^* in S_{C_{t_i}}$ and $JS(C_{ti}^*, C_{tj}^b) \geq \lambda$. | (f) |
| *Dissolve:* a community $C_{t_i}^a$ dies over time by losing its members and then we cannot observe any community exceeding similarity threshold $\lambda$ at following next steps.<br><br>$dissolve(C_{ti}^a) = true$;<br>$if \nexists C_{t_j}^* in G$ at time $t_j$ and $t_j > t_i$, and $JS(C_{t_i}^a, C_{t_j}^*) \geq \lambda$ and $\lvert V_{C_{ti}^a}\rvert < 3$. | (g) |

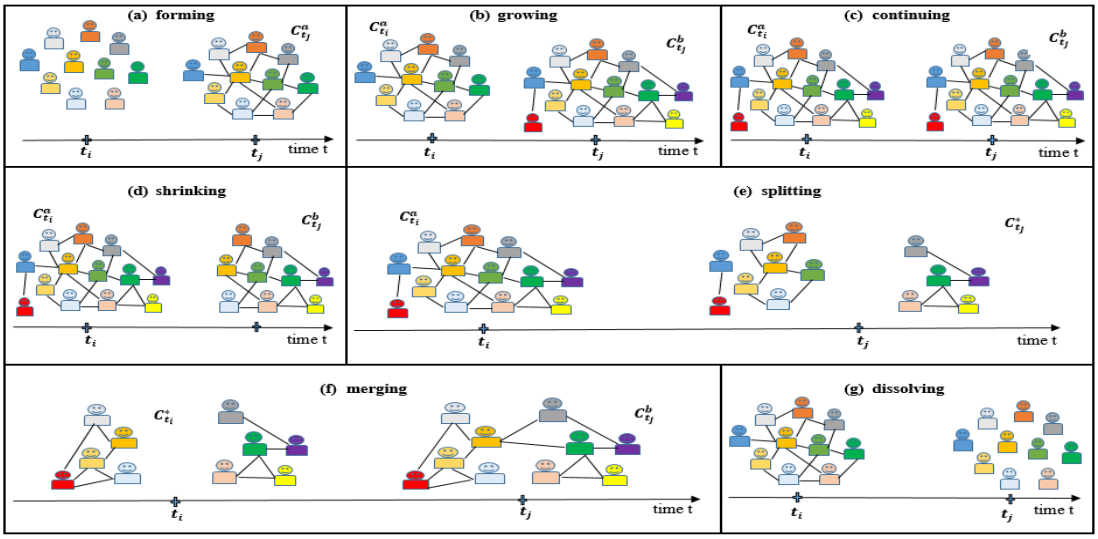


**FIGURE 1.** An illustration of community evolution events where i and j represent time steps where $i < j$.

consecutively and all tracking methods can track it naturally. Nonconsecutiveness is the situation where a community evolves intermittently. That is, the community is observed at time $t$ and then at time $t+2$ or later, rather than at time $t+1$. The expected solution should be able to track both types of communities.

*Criterion #3 Coverage of types of evolution events:* The ability to detect all event types with one solution method. Note that all possible event types are listed in Table 1.

*Criterion #4 Tracking k-Community merge/split:* Support event detection for k-community merge and k-community split, where $k > 2$, since community merge/split can occur between more than two communities in real networks.

### D. SIMILARITY PRESERVING COMMUNITY SIGNATURES

Member sizes are generally large for real networks such as social networks, citation networks, etc. If networks include hundreds of thousands or millions of community members, it may not be possible to store members of communities in the main memory to process. Even if the members fit in the memory, the computation of Jaccard Similarity-JS of each community pair may be infeasible. To overcome these problems, communities should be replaced with more miniature representations where the similarity between the community pairs should be preserved.

Minhashing [17] is one of the solutions to convert each community into a small signature with a group of hash functions. The goal of minhashing is to provide a fast approximation to the Jaccard similarity between any community pair without computing the unions and the intersections of the community pairs. Instead, minhashing once creates a minhash signature for each community and uses it to approximate the similarity.

MinHash has a surprising property, according to which the probability that the MinHash of random permutation produces the same value for the two communities equals the Jaccard Similarity value of those communities.

$$Pr(sig[C^x_{t1}] = sig[C^y_{t2}]) \tag{2}$$

Let X be a set of common minhashes of both communities, and Y be a set of uncommon.Then the probability of the minhash signature of two communities being the same is $\frac{\#ofX}{\#of(X+Y)}$. Jaccard similarity of two communities is the common number of members in both communities (# of X) divided by the total number of members in both communities (# of (X+Y)). Therefore, (2) holds true, which shows the similarity of the community pairs is preserved. The proof of (2) can be found in reference [19]. Note that the creation of minhash signatures is explained in Section IV.B.

### E. THE RELATION BETWEEN SIMILARITY THRESHOLD AND LOCALITY SENSITIVE HASHING

Searching for similar communities by pairwise comparison of all vertices of communities is a brute force procedure and requires a $(timeStepCount-1)\times n^2$ comparison. Comparing minhash signatures of communities instead of checking all vertices of these community pairs $(timeStepCount-1)\times c^2$ obviously takes less time, but it is a brute force technique. There is a need for an effective solution to find similar community signatures for a given community to get rid of the brute force approach. Since LSH is an efficient technique to find near neighbors, we adapt LSH to find likely similar communities without pairwise comparison of community signatures. This adaptation and the theoretical analysis of LSH are described below.

LSH table is a signature matrix ($sig[]$) that contains the minhash signatures of all communities in the network. That is, each column of sig[] is a signature of a particular community. Thus, there is a need to match these signatures. A general approach of the LSH technique is to hash minhash signatures

such that similar signatures are more likely to be hashed into the same bucket than dissimilar signatures. For this reason, *sig*[] is first divided into $b$ bands, where each band contains the same number of $r$ rows, where $h = b \times r$ and $h$ is the number of universal hash functions used. A part of a minhash signature whose length is r within a band is called a ''pickle''. Then, LSH takes pickles and hashes them into a large number of buckets. In this way, the same pickles are hashed into the same buckets for each band. If there is at least one pickle with minhash signatures of a pair of different communities in the same bucket in the same band, they are considered as candidates for similarity. When LSH is queried for similar communities of a given community, only candidate pairs are returned. However, there is a possibility of false positives since the dissimilar pairs are in the same bucket. On the other hand, false negatives may occur if some similar signature pairs cannot be hashed into the same bucket. Note that the same hash function can be used for all bands, but it is necessary to use separate bucket collections for each band. The application of minhashing and LSH techniques in TREC is explained in detail with examples (Table 5 and Fig. 3) in Section IV.

$$LSH_{ST} = (\frac{1}{b})^{(\frac{1}{r})} \tag{3}$$

The pickles in the same buckets (candidate communities to be similar) must exceed a threshold $LSH_{ST}$ for similarity. That is, the candidate communities for a given community are at least $LSH_{ST}$ similar with certain probabilities. The relationship between $LSH_{ST}$, $b$ and $r$ is given in (3), and the values of $b$ and $r$ are chosen experimentally to obtain the desired $LSH_{ST}$.

Table 2 shows the JS relationship between the communities corresponding to the different chosen λ-values and the number of matching communities, and consequently the number of communities to be tracked, in the *BirthDeath benchmark dataset*. Note that the dataset is explained in Section V. As can be seen from the table, the number of matching communities and consequently the number of communities to be tracked decreases with increasing λ-value, as expected, because the higher the threshold, the fewer matches there are. Therefore, the accuracy of community tracking decreases. However, the execution time efficiency increases as fewer communities need to be tracked. Therefore, the λ-value is important to determine the accuracy bounds or the target time efficiency of the proposed solution.

In this study, $LSH_{ST}$ is chosen as 0.1(e.g.,10%) because we want to identify all community signature pairs whose similarity is at least $LSH_{ST}$, and include them in the community matching process to ensure maximum accuracy in community tracking. In the tracking phase, JS of possibly similar communities is calculated by processing all members (vertices) of the community pairs. If the value is at least 0.1 (e.g., $\lambda = 10\% = 0.1$), then they are matched assuming that they are similar. The analysis results show that λ-values of up to 0.3 can be chosen for the benchmark dataset. However, we choose

**TABLE 2. The effect on λ on tracking evolution of communities.**

| λ | # of matched communities | # of communities to be tracked |
|---|---|---|
| 0.1 | 1730 | 683 |
| 0.2 | 1717 | 683 |
| 0.3 | 1660 | 672 |
| 0.4 | 1549 | 640 |
| 0.5 | 1417 | 626 |
| 0.6 | 1104 | 498 |
| 0.7 | 710 | 338 |
| 0.8 | 238 | 125 |
| 0.9 | 13 | 5 |

0.1 as the threshold for JS between community pairs for all our datasets for two reasons. First, we want to detect even small changes that lead to community evolution. Second, we believe that $\lambda = 0.1$ is a good threshold for similarity since it is used by our competing methods in Section V and the paper provides the benchmark datasets [20].

$$1 - (1 - s^r)^b \tag{4}$$

LSH guarantees that communities that are either equal to or greater than $LSH_{ST}$ are returned with a certain probability value. The probability of a community signature pair becoming a candidate that has a percent similarity of $s$ is calculated by (4), where $s \geq LSH_{ST}$. The banding analysis of LSH is described in detail in [19]. The probabilities of signature pairs detected by LSH with similarity of s are calculated and shown in Table 3, where $LSH_{ST} = 0.1$.

As can be seen in Table 3, LSH recognizes community-signature pairs that have a similarity of 10% with a probability of 36%. LSH also recognizes the pairs with 20% similarity with a probability of 84%. Moreover, it recognizes the pairs that have a similarity of 40% or more with a probability of 100%. From the table, it can be seen that as the similarity of the community signature pairs increases, the recognition performance of LSH also increases.

**TABLE 3. Probabilities of the signature pairs detected by LSH.**

| Similarity between a community signature pair, $s$ | Probability of the pair that has $s$ similarity detected by LSH, $1 - (1 - s^r)^b$ |
|---|---|
| 0.1 | 0.364 |
| 0.2 | 0.841 |
| 0.3 | 0.986 |
| 0.4 | 1.000 |
| 0.5 | 1.000 |
| 0.6 | 1.000 |
| 0.7 | 1.000 |
| 0.8 | 1.000 |
| 0.9 | 1.000 |

## III. RELATED WORK

A recent taxonomy introduced by Dakiche *et al.* [13] divides methods for tracking community evolution into four groups according to their methodological principles, which are described below:

(i)The methods that use the independent community detection and matching approach [14]–[16], [20]–[24] divide the

network data into time steps and then run a community detection method on the data of each time step to reveal the underlying community structures. Later, the evolution of a community in these time steps is tracked in ascending order by finding its similar community(s) in the next time steps, which is called community matching.

(ii) The methods that use the dependent community discovery approach [25]–[27] use the network data from the current time step and the known community structure from the previous time step to discover the related communities.

(iii) The methods that use the simultaneous community detection approach [28]–[30] first construct a single graph for all time steps of the network and then run a static community detection method on this single graph.

(iv) The methods that use a dynamic community detection approach [31]–[33] work with temporal networks. They run a community detection method on the data of the first time step, and then evaluate added/deleted members and connections/relationships of subsequent time steps as updates to the first detected community structures.

Our proposed method belongs to group (i) in the taxonomy of Dakiche *et al.* Therefore, related methods for tracking community evolution using the independent community detection and matching approach are presented and compared in the rest of this section.

Table 4 summarizes the characteristics of the most commonly used and the most recent methods from 2007 to 2021. The columns of the table contain the names of the methods presented in related works. The attribute Year in the second row indicates the publication date of the method. Attribute criterion #1 indicates whether the communities overlap or not (are disjoint), and they are represented in the table by ''O.'' and ''NO'', respectively. Attribute criterion #2 indicates whether the method tracks nonconsecutively evolving communities, since all methods track consecutively evolving communities. Attribute criterion #3 indicates whether or not the method can detect all types of evolutionary events. Attribute criterion #4 indicates whether the method is able to detect k-community merge and splits. Attribute Used Similarity Metric represents the similarity measure used by the tracking method to determine the similarity of a pair of communities.

As can be seen from Table 4, some methods work with nonoverlapping communities [14]–[16], [20], [21], and [22], and the others with overlapping communities [23], [24]. While all methods track consecutively emerging communities, only some methods ( [14], [15] and [20]) are able to track nonconsecutively emerging communities. However, only the method of Greene *et al.* [20] is limited to detecting the nonconsecutive evolution of a community for two consecutive snapshots. For criterion #3, the ability to detect events, [14]–[16], [23], and [24] can detect all evolution events, but [20]–[22] do not detect ''shrink''/''growth'' and ''continue'' events, respectively. For criterion #4: The methods [20], [21] cannot detect whether k-community merge and split occurs, and the others support k-community merge and splits. However, in real networks, k-communities can merge and a community can split into k-communities. As for the similarity metric used, Jaccard similarity is the most common metric among the methods. [22] accepts a pair of communities as similar if their common members have a proportion of k (i.e., a predefined similarity threshold) or more in the largest community. The inclusion measure of GED [23] combines both the quantity (i.e., how many members of one community belong to another) and the quality of community members (i.e., the social importance of community members such as degree of centrality, degree of density, page rank). ICEM [15] and WECEM [16] simply calculate similarity using the ratio of intersection to size of a community.

In summary, some of the newer methods, such as Tajeuna *et al.* [14] and ICEM [15], meet all of the functional criteria listed in Section II.C and Table 4. The most recent work whereas WECEM [16] cannot track nonconsecutive evolutions and due to that reason it is not considered as one of the competing works and is not considered in our performance analysis for TREC. In terms of performance, the method of Tajeuna *et al.* [14] focuses on accuracy, while the ICEM method [15] focuses on execution time efficiency and accuracy. But none of the related works consider the accuracy of tracking results, space efficiency, and execution in reasonable time while performing all desired functions simultaneously. This is a shortcoming in a low memory environment, and we focus on this shortcoming in this paper.

## IV. TRACKING EVOLUTION OF COMMUNITIES

The task of tracking community evolution can be divided into three main parts, namely network representation, community detection and evolution analysis. In network representation, a dynamic network is represented as a sequence of static networks by decomposing the network into time steps. In the second step, an existing static community detection method is applied to all time steps in the network and the underlying community structure of the network is revealed. In this study, we use Louvain method [34] for community detection in all time steps because it is one of the best methods among community detection methods in terms of execution time and accuracy [35], [36]. Since Louvain is executed once in each time step of a dynamic network, there is no need to worry about the nondeterminism of Louvain. However, there is no obligation to use the Louvain algorithm. Instead, any method can be used to detect disjoint communities. The evolution analysis step consists of two parts, namely uncovering the evolution chain of communities and labeling evolutionary events. In this chapter, we focus on the evolution analysis part and how we perform it, since the network representation step and the community detection step are common to tracking methods.

Note that the community detection method used can be applied to both nodes and edges and does not affect the evolution analysis. However, in the evolution analysis, we are

**TABLE 4.** Overview of the functional aspects of the mainstream and latest competitor methods.

| Community Tracking Method | Asur et al. [21] | Greene et al. [20] | Takaffoli et al. [22] | GED [23] | SCGI [24] | Tajeuna et al. [14] | ICEM [15] | WECEM [16] | Our Method TREC |
|---|---|---|---|---|---|---|---|---|---|
| Year | 2007 | 2010 | 2011 | 2013 | 2013 | 2016 | 2020 | 2021 | 2021 |
| Criterion #1 Community Structure | NO. | NO. | NO. | O. | O. | NO. | NO. | NO. | NO. |
| Criterion #2 Tracking Nonconsecutive Evolutions | No | Limitedly yes | Yes | No | No | Yes | Yes | No | Yes |
| Criterion #3 Coverage of Types of Evolution Events | No (Shrink, Growth) | No (Continue) | No (Shrink, Growth) | Yes | Yes | Yes | Yes | Yes | Yes |
| Criterion #4 Tracking k-Community Merge/split | No | No | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Similarity Metric Used | Modified Jaccard | Jaccard Similarity | A Specific Measure | Inclusion measure | Modified Jaccard | Mutual Transition | A Specific Measure | A Specific Measure | Jaccard Similarity |

| | |
|---|---|
| $C_1^1$ | 2 5614 ... 4735 5556 |
| $C_1^2$ | 3 14179 ... 9060 8972 |
| ... | ... |
| $C_1^k$ | 10440 13916 ... 9800 11275 |
| $C_2^{k+1}$ | 1 12522 ... 6548 7837 |
| $C_2^{k+2}$ | 2 10652 ... 9106 6854 |
| ... | ... |
| $C_2^l$ | 955 12427 ... 9680 10578 |
| ... | ... |
| $C_{tsc}^{z+1}$ | 1 13572 ... 6547 8086 |
| $C_{tsc}^{z+2}$ | 1083 5719 ... 7433 6306 |
| ... | ... |
| $C_{tsc}^{tcc}$ | 12140 393 ... 10246 10873 |

**FIGURE 2.** Illustration of community vector *v*.

only interested in the vertices of the communities, since we detect the relationships of the revealed communities over time based on the vertices.

In the following sections, we explain how we track the evolution of communities and detect evolution events.

### A. REPRESENTING COMMUNITIES

The result of the community detection is entered into the evolution analysis step. That is, the text files containing the vertex IDs (members of a community listed in a row, as shown in Fig. 2) of all detected communities must be loaded into main memory as a single community vector $v$. The vector $v$ indicates which community ($C_t^i$) belongs to which time step, as shown in Fig. 2.

Each line of v, the members (vertices) of the identified communities are listed in order. Note that the members are not ordered. In the notation $C_t^i$, the subscripts $t$ and $i$ represent the time step and ID of the community, respectively. The members of the communities detected for each time step $t$ are stored in $v$. Therefore, the space required to store all the detected communities for the network is $S(n) = t \times n = O(t \times n)$ where $n$ is the total number of unique vertices in the network. The size of $v$ is $c \times t$. The execution time cost of creating community vectors requires each vertex to be visited at most t times, $T(n) = O(t \times n) = O(n)$.

### B. BUILDING MINHASH SIGNATURES

Minhashing is a technique for compressing large datasets into fixed-length sketches called "signatures". It was introduced by Broder [17], whose implementation initially dealt with binary vectors but was extended to integer vectors and continuous variables [37]. The general implementation using bit vectors is described in reference [19]. In this study, we implement minhashing with integer community vectors, where each vector contains the members of a community (a row of $v$), since all members ID's are positive integers.

Minhashing uses several universal hash functions (H) such as $H_i(v) = av + b(mod\ p), i = 1, \cdots, h$ where $h$ is number of hash functions used, $a$, $b$ are random integers, and $p$ is a prime number where greater than or equal to the number of unique vertices in the dataset. Using minimum hash values for $v$ satisfies the random sampling requirement for the community representation. Therefore, each community integer vector (containing all the member IDs of the community) is passed through these h-functions and the minimum hash values for each of the hash functions used are selected. At the end of this process, a two-dimensional signature matrix *sig[]* is obtained as seen in Table 5.

Table 5 shows an example of a signature matrix *sig[]*. The columns of the matrix represent community signatures, the rows represent the hash functions used, and each cell contains the associated minimum hash result for the hash function over these community members. *tcc*(total number of communities) is the number of communities and *tsc* is the total number of time steps in the dataset. The cost of creating these minhash signatures is manageable and their complexity is represented as $T(n) = h \times n \times t = \mathcal{O}(n)$ since $h$ and $t$ are constant, and where $h$ is the number of hash functions used in a signature, $n$ is the number of unique vertices in the dataset, and $t$ is the number of time steps into which the dataset has been divided.The storage cost of the signature matrix is $S(n) = O(h \times c \times t)$ because the length of each signature is $h$, and in total, there are at most $c \times t$ communities.

**TABLE 5.** An example of the signature matrix, is sig[].

| | $C_1^0$ | $C_1^1$ | $C_1^2$ | ... | $C_3^{537}$ | $C_3^{538}$ | $C_3^{539}$ | $C_3^{540}$ | ... | $C_5^{2880}$ | $C_5^{2881}$ | $C_5^{2882}$ | ... | $C_{tsc}^{tcc}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **h1** | 3 | 5 | 2 | | 4 | 17 | 3 | 4 | | 5 | 5 | 3 | | .. |
| **h2** | 7 | 13 | 8 | | 10 | 24 | 7 | 10 | | 13 | 18 | 7 | | .. |
| **h3** | 17 | 24 | 1 | | 24 | 43 | 37 | 11 | | 24 | 15 | 10 | | .. |
| **h4** | 24 | 34 | 44 | | 34 | 22 | 44 | 55 | | 34 | 22 | 15 | | .. |
| **h5** | 31 | 60 | 23 | | 57 | 58 | 44 | 44 | | 31 | 16 | 44 | | .. |
| **h6** | 38 | 54 | 12 | | 70 | 66 | 15 | 54 | | 38 | 42 | 54 | | .. |
| **h7** | 45 | 83 | 64 | | 64 | 1 | 5 | 15 | | 83 | 34 | 40 | | .. |
| **h8** | 52 | 96 | 72 | | 72 | 3 | 8 | 20 | | 96 | 96 | 50 | | .. |

### C. BUILDING LSH TABLE AND BUCKETS

LSH is a technique originally introduced by Indyk and Motwani [18] for nearest neighbor search. The combination of LSH and minhashing technique is an elegant method for identifying near neighbors in Jaccard space. Therefore, LSH with minhashing technique is adapted for the first time in this study to identify similar communities. The *sig[]* after minhashing is divided into bands and pickles are identified as shown in Fig. 3(a). Then, same pickles in the same band are assigned to the same buckets, as shown in Fig. 3(b). If two communities are present in at least one bucket, they are considered as a candidate pair for similarity.

This step includes the generation of pickles, adding these pickles into a hash map, and putting the community identification numbers into the same row of a bucket. Note that a bucket is represented by a cell of buckets vector. Therefore, the time complexity of the generation of pickles dominates, and the execution time complexity of this step is $T(n) = O(h \times n \times t) = O(n)$. Since this step needs to store buckets vector, its space cost is S(n) = O(c × t) = O(v), where $v$ is the overall communities.

### D. TRACKING SIMILAR COMMUNITIES OVER TIME

The pseudocode for tracking the evolution of communities is described in Algorithm 1. For each community to be tracked, a set of communities in its evolution chain is recorded (Line 3), and all time steps are traversed to account for nonconsecutive evolution by traversing successive time steps (Line 4, Line 5-Line 20). LSH buckets are queried for the actively tracked community in the set (Line 8). The LSH buckets are used to obtain the list of candidate communities (Line 8). The JS between the actively tracked community and its candidates is checked (Line 9 - Line 11). If they are similar, the candidate community is added to the evolution set (Line 12) and the candidate community is removed from the tracking list (Line 13). After all time steps coming after its time step have been searched for a community, the evolution events are determined (Line 21-Line 34) according to the rules formulated in Table 1. After all time steps have been processed, S is appended to the tracking list (Line 35) and S is released for the next community evolution (Line 36).

In summary, the main idea of pseudocode is to filter dissimilar communities via LSH buckets and then check the actual similarity between community pairs by processing all time steps starting from the actively processed time step, even for nonconsecutive evolutions. Although LSH is a probabilistic technique, it guarantees to identify similar community signatures of a given community with a certain probability, which is explained in the subsection II.D.

Fig.4(a) and Fig.4(b) show an example of *tList* and the corresponding conceptual schema. In each line, there is an evolution chain of a community. For example, in row 1, $C_1^{56}$ has no other subsequent community in its evolution chain. This means that $C_1^{56}$ was born and has not been observed for a single time step during the observation; thus, it dissolves. Communities $C_1^{61}$ and $C_1^{277}$ merge and $C_2^{1134}$ forms. The community $C_4^{2026}$ is split into two subcommunities $C_5^{2587}$ and $C_5^{2851}$.

Note that the tracking task is repeated for all communities to be tracked ($tcc = t \times c$) and most of the complexity of the process comes from computing $JS(n \times \log n)$. Querying LSH buckets takes $\mathcal{O}(1)$ time since it only returns the communities that are in the index of the queried community ID. Therefore, the time complexity of the tracking algorithm is $T(n) = t \times c \times n \times \log n = \mathcal{O}(cn \log n)$, since $t$ is constant and thus negligible. Since $v$ and LSH buckets are kept in main memory during the execution of TREC, the total memory required is $S(n) = n + v = \mathcal{O}(n + v)$, where $n$ is the number of unique vertices in the dataset and $v$ is the vector consisting of all communities at all time steps.

## V. EXPERIMENTAL STUDY

In this section, we first describe the datasets used and the experimental setup. Then, we evaluate the impact of the LSH and minhashing techniques on the TREC method. Our TREC method aims to accurately track all types of evolution events of the dynamic communities, use an efficient memory space, and run in a reasonable time compared to related works. Therefore, in this section, we evaluate the performance of our method both theoretically through complexity analysis and practically through accuracy analysis and profiling of the TREC method in terms of memory space and execution time.

### A. DATASETS

In this study, both benchmark datasets and real datasets are used. The details of the datasets are as follows:

The *benchmark datasets* of Greene *et al.* [20] are used for accuracy analysis. They contain ground truth information about communities at all time steps and are accessible online. The datasets are constructed from four different synthetic

**Algorithm 1** Tracking Evolution of Communities

**Input :** $L$(List of communities to be tracked), $\lambda$, $LSH$, $tcs$ (the number of time steps in the network).
**Output:** $tList$ (List of tracked communities).

**foreach** *community* $C_t^i$ *in* $L$ **do**
  $C_t^i$ "form"s
  $S = \{C_t^i\}$ // $S$ is the set holds the evolution of each $C_t^i$
  $t_j = t + 1$ // $t_j$ shows the time step to be processed
  **while** $t_(j) \leq tcs$ **do** // for processing both consecutive and nonconsecutive time steps
    let $rCommunities$ be the list of the recent time step communities in $S$
    **foreach** *community* $C_{tx}^k$ *in* $rCommunities$ **do** // in the case of splitting communities
      $cCommunities = LSH(C_{tx}^k)$ // $cCommunities$ is the list of the candidate communities returned by $LSH$ for $C_{tx}^k$
      **foreach** *community* $C_{tc}^c$ *in* $cCommunities$ **do**
        **if** $tc == t_j$ **then**
          **if** $JS(C_t x^k, C_{tc}^c) \geq \lambda$ **then** // similarity is matched the tracked community and candidate community
            $S = S \cup C_{tc}^c$ // append the matched community to the evolution set
            $L = L - C_{tc}^c$ // remove matched communities from the tracking list
            let $mMap$ hold (a community, its matched communities) as (key, value) pairs respectively.
            search for $C_{tx}^k$ in $mMap$
            **if** *it is a key in* $mMap$ **then**
              append $C_{tc}^c$ to the respective value of $mMap$
            **else**
              insert ($C_{tx}^c$,$C_{tc}^c$) pair into $mMap$
    $t_j = t_j + 1$
  $mCommunities = mMap(C_{tx}^k)$ // $mCommunities$ is the list of communities matching with $C_{tx}^k$
  **if** $mCommunities == \emptyset$ **then** // no matching
    $C_{tx}^k$ "dissolves"
  **if** $|mCommunities| \geq 2$ **then** // based on $mMap$, more than one matching
    mc = $mCommunities.count(t_x < t_m)$
    **if** $mc \geq 2$ **then** // more than one match with communities in next time steps
      $C_{tx}^k$ "splits"
    mc = $mCommunities.count(t_x > t_m)$
    **if** $mc \geq 2$ **then** // more than one match with communities in previous time steps
      $C_{tx}^k$ "merges"
  **if** $|mCommunities| == 1$ *and* $t_x < t_m$ **then** // one to one matching
    **if** $C_{tx}^k < C_{tm}^m$ **then**
      "shrinks"
    **if** $C_{tx}^k > C_{tm}^m$ **then** $C_{tx}^k$ "grows" **else** $C_{tx}^k$ "continues"
  $tList = tList \cup S$
  S = {}

graphs, each containing five static networks, meaning that there are five time steps (t) with 15000 vertices (n) to simulate nonconsecutively evolving communities, and containing all types of community evolution events. In the BirthDeath dataset, the authors randomly create 40 additional communities and randomly remove 40 communities. In the Expand dataset (Grow-Shrink), they create graphs in which 40 randomly selected communities grow or shrink by 25%. In the MergeSplit dataset, the authors simulate 40 communities that split and 40 communities that merge. In the Nonconsecutiveness dataset, they randomly hide 10% of the members at each time step.

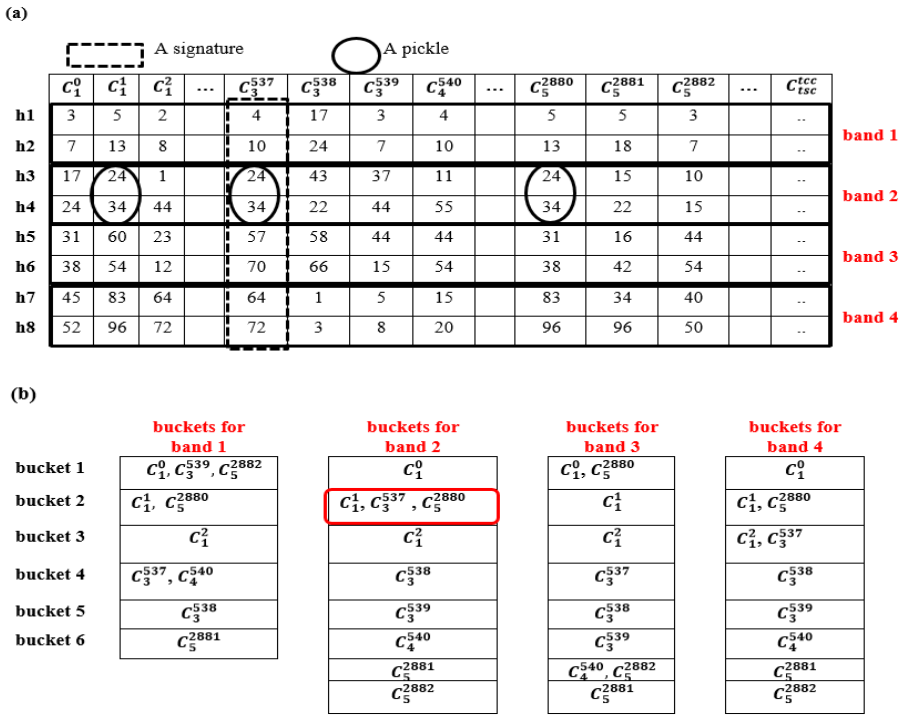


**FIGURE 3.** An example of "how logically LSH with minhashing works". (a) LSH table and (b) LSH Buckets.

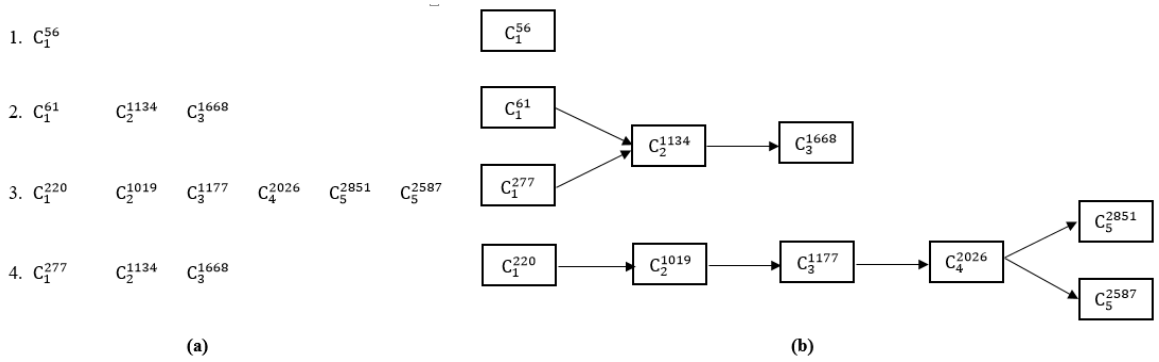


**FIGURE 4.** An example of (a) conceptual content of tList (b) its correspondent conceptual schema.

The memory and execution time performance of TREC for is evaluated on *real datasets*. These datasets are:

- *Autonomous Systems (AS) Dataset* [38] contains a daily communication network of routers from logs. Daily unweighted, undirected graphs are created from December 1999 to January 2000, where each vertex has a router identification number and each edge indicates the relationship between each vertex.
- *The DBLP Dataset* [39] contains the authors' co-publications. Using data mining and artificial intelligence domains, unweighted and undirected graphs between the years 2001 and 2013, where each vertex represents the authors and each edge represents the co-authorships and citations of the publications.
- *The Yelp Dataset* [40] contains user reviews about businesses, but our study, user friendships are important. Therefore, monthly unweighted friendship graphs are constructed for each user who has at least one friend from October 2013 to July 2014, where each vertex represents users and each edge represents a friendship.
- *The 2009 Digg friendship Dataset* [41] contains information about friendships on the Digg platform. Bimonthly, undirected, and unweighted friendship graphs are constructed from July 2007 to July 2009. Each vertex represents a user and each edge represents a friendship. The dataset is available in [42].

### B. EXPERIMENTAL CONFIGURATION

In these experiments, the rate of change of community size is assumed to be 5% to determine "continue", "growth", and "shrink" events. The parameters $\lambda$, $h$, $r$, $b$ and $LSH_{ST}$

for TREC are respectively assigned as 0.10, 90, 2, 45, 0.10. The theoretical relations for the assigned values of these parameters are explained in Section II in subsection D.

For the experimental analysis, a laptop with the following configuration is used: Intel (R) Core(TM) i7 CPU @ 2.30 GHz. Processor, 64-bit Win10 operating system and 16 GB of main memory. Both TREC and its competitors are implemented in C++ programming language.

### C. IMPACT OF USING LSH AND/OR MINHASHING TECHNIQUES

The following two experiments are conducted to separately demonstrate and compare the effects of minhashing and/or LSH techniques on the proposed method TREC in terms of their accuracy and required execution times. The matrix of minhash signatures of all detected communities (sig[]) is used in the following experiments, as shown in Table 5.

*Experiment 1:* The goal was to demonstrate the effect of using JS of minhash signatures of community pairs instead of JS of community pairs with respect to a member-based comparison. For this purpose, the method *"minhashing_effect_TREC"* was developed. In this method, the event types of minhash signatures of candidate communities filtered out from LSH for a tracked community (relative columns of sig[] in Table 5) are identified after checking JS of the signatures. This shows us how accurate community matching is when community signatures (in Table 5) are used instead of the community itself. Note that it is sufficient to update the computation of JS in Line 11 in Algorithm 1 to compute the JS signatures of the communities.

*Experiment 2:* It was used to demonstrate the effect of LSH technique. For this purpose, the method *"WithoutLSH_effect_TREC"* was developed. In this method, there is no LSH table and its buckets. Instead, JS of all pairs of minhash signatures (see Table 5) are checked and the community IDs of similar signatures are stored as pairs in a hash map, where a key is a community ID and a value is the IDs of their similar communities. As in the case of Experiment 1, JS of a community and its similar communities is computed using the minhash signatures of the communities (see Table 5) and similar communities are filtered out from this hash map. This shows us the contribution of LSH technique in terms of accuracy and execution time. Note that it is sufficient to change the obtaining of the candidate communities in Line 8 in Algorithm 1 from LSH to the hash map.

The TREC, minhashing_effect_TREC and WithoutLSH_effect_TREC methods are run separately on the benchmark datasets. Fig. 5(a) and Fig. 5(b) show the accuracy and execution time of the methods, respectively.

The execution time of TREC and minhashing_effect_TREC are very close to each other. minhashing_effect_TREC consumes slightly less time since it only checks the similarity of signatures. As can be seen in Fig. 5(b), the time consumption of WithoutLSH_ effect_TREC is ten times higher than the others, since it has to check the signatures of each community pair. Therefore, the hybrid uses of minhashing and LSH in TREC provides better accuracy with reasonable execution times.

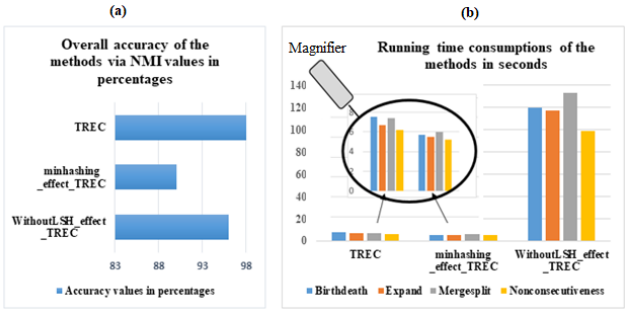


**FIGURE 5.** (a) Accuracy and (b) execution time consumptions of TREC, *minhashing_effect_TREC* and *WithoutLSH_effect_TREC methods*.

**TABLE 6.** Complexity analysis of the TREC method, and its competitors.

| Method | Time Complexity | Space Complexity |
|---|---|---|
| Tajeuna et al.'s method | $\mathcal{O}(c^2 \log c)$ | $\mathcal{O}(c^2)$ |
| ICEM with Evolution Chain | $\mathcal{O}(n \log n)$ | $\mathcal{O}(2n)$ |
| TREC (our method) | $\mathcal{O}(cn \log n)$ | $\mathcal{O}(n+v)$ |

### D. PERFORMANCE ANALYSIS

The method of Tajeuna *et al.* [14] and the ICEM method [15] are considered to be competitors of TREC since they share the same functional properties, such as tracking type (e.g., both consecutive and nonconsecutive evolutions), k-community merging/splitting, detecting all event types, and working with disjoint community structures. Moreover, both the method of Tajeuna *et al.* and ICEM are the most recent methods in related works. The performance of our TREC method is studied using the three main components of complexity analysis, accuracy analysis, and execution time analysis by comparing the competitors' methods.

#### 1) COMPLEXITY ANALYSIS

Tracking community evolution requires three basic components such as network representation, community detection, and evolution analysis. Since TREC and its competitors use the same networks and community detection method (Louvain [34]), the computational cost of network representation and community detection is neglected. Therefore, only the community evolution analysis component is considered in the complexity analysis. Although Louvain was used in our experiment, any community detection method for nonoverlapping communities such as Markov Cluster Algorithm (MCL) [43], Local Community Mining [44], Infomap [45] and Leiden [46] can be used.

TREC's time and space complexity and competing methods reside in Table 6. Since the ICEM method [15] does not generate evolution chains, a balance must be found between ICEM and TREC in terms of generating evolution chains of communities. Therefore, a subroutine is added to the ICEM method whose title is concatenated with "with Evolution

**TABLE 7. Accuracy of the TREC method and its competitors in terms of NMI values.**

| Benchmark Datasets | Tajeuna et al.'s method | ICEM with Evolution Chain | TREC |
|---|---|---|---|
| BirthDeath | 0.994 | 0.967 | 0.995 |
| Expand | 0.983 | 0.984 | 0.983 |
| MergeSplit | 0.939 | 0.957 | 0.947 |
| Nonconsecutiveness | 0.989 | 0.991 | 0.991 |
| **Average accuracy scores** | **0.976** | **0.974** | **0.979** |

Chain''. In TREC, most of the execution complexity is caused by the tracking algorithm $\mathcal{O}(cn \log n)$, where $c$ is the average number of communities per time step and $n$ is the number of unique vertices in the dataset. The community representations and buckets reside in main memory during the execution of TREC. Therefore, the total space requirement is $\mathcal{O}(n + v)$, where $|v|$ is the total number of communities in community vector $v$.

### 2) ACCURACY ANALYSIS

Since accuracy is one of the most important performance measures, this subsection examines the accuracy of the TREC method and compares it to the work of its competitors. For an objective and consistent comparison, the same benchmark datasets and community detection method (Louvain [34]) are used for all competing methods. The NMI [48] values are used to measure the accuracy of the detected community structures. NMI [48] is one of the conventional cluster validation techniques for the case where the ground truth cluster structure is known. It measures the accuracy of the detected community structure by comparing it with the ground truth community structure. It takes real values between 0 and 1. Higher NMI values mean that the tracking is more successful. The NMI values between the ground truth data and the data obtained after tracking are calculated using a special software [49].

Table 7 shows the accuracy in terms of NMI values per dataset achieved by the TREC method and its competitors. Looking at the average accuracy values of the methods, we find that the method of Tajeuna *et al.* [14], ICEM with Evolution Chain and the TREC method have percentage accuracy values of 97.6, 97.4 and 97.9 respectively. Hence, it can be concluded that their performances are almost equal.

### 3) EXECUTION TIME AND SPACE ANALYSIS

The last performance measure considered is the execution time and memory performance of TREC and its competitors. Profiling of all methods is performed in the Visual Studio 2019 Community Edition [49] environment.

Table 8 shows some properties of the dataset as well as the execution time and memory requirements of the TREC method and its competitors. Both benchmark datasets and real datasets are considered in the table. The '' Dataset Characteristics column contains the subcolumns ''t'', ''n'' and ''c'' for each dataset. The subcolumn ''t'' indicates the number of time steps that make up the network. The subcolumn ''n'' indicates the number of unique vertices each dataset has. The subcolumn ''c'' indicates the number of average communities per time step. The column ''Methods'' contains the performance results of TREC and its competitors such as the method of Tajeuna *et al.* method [14] and the ICEM with Evolution Chain method. The subcolumns ''(1) in MB'' show the highest memory consumption in megabytes and the subcolumns ''(2)'' show the CPU consumption in seconds/minutes and hours during the execution time of each method.

Profiling the Tajeuna *et al.*'s method is performed on the Yelp dataset until computer's memory is exhausted. About six hours (e.g., exactly 356 minutes) pass until this point, and the highest memory used is 636.2 MB. Therefore, the actual performance of the method for this dataset cannot be measured, and the values are marked with ''x'' sign in the corresponding cells of Table 8. According to the table, the Tajeuna *et al.*'s method has the highest memory and execution time consumption. The TREC method has the lowest memory consumption while ICEM with Evolution Chain method has the lowest execution time.

### 4) EVALUATION OF RESULTS

In this subsection, we evaluate the algorithmic (time and space analysis) and analytical (accuracy and real-time analysis including memory and execution time requirements) results of the TREC method and its competitors.

Table 9 shows the order of complexity of the TREC method and the competing methods. As can be seen from the table, the TREC method and the ICEM with Evolution Chain method have linear complexity in terms of space and time. On the other hand, the Tajeuna *et al.*'s method [14] has a quadratic complexity in terms of time and space. When we compare the order of complexity of the two methods, we find that the time complexity of the ICEM with Evolution Chain method is lower than that of the TREC method, while the space complexity of the TREC method is lower than that of the ICEM with Evolution Chain.

Table 8 is a summary table of some of the properties of the datasets, the highest memory usage in megabytes, and the execution time for the methods. From the table, it can be seen that the method of ICEM with Evolution Chain requires the least execution time and the runtime consumption of TREC is close to it. However, the method of Tajeuna *et al.* is the most time consuming among them and requires more than 750 times execution time for the same datasets. Moreover, it cannot work with the ''Yelp'' dataset (although it takes about 6 hours to reach this point), while the other two methods finish their execution in seconds.

Regarding the space requirements of the methods, the Tajeuna *et al.* method is the most space-consuming among them, while the TREC method is the most space-efficient, as shown in Table 8 and Table 9. The ICEM with Evolution Chain method requires a memory space that is a constant multiple of the memory space required by the TREC method.

**TABLE 8.** The highest memory usage in Mega Bytes during their executions and execution time of the methods.

| | | Datasets Characteristics | | | Methods: Tajeuna et al.'s method | | Methods: ICEM with Evolution Chain | | Methods: TREC | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | t | n | c | (1) in MB | (2) | (1) in MB | (2) | (1) in MB | (2) |
| Benchmark Datasets | BirthDeath | 5 | 15000 | 577 | 5.8 | 16 min. 28 s. | 3.6 | 1.3 s. | 2.9 | 9.2 s. |
| | Expand | | | 584 | 5.9 | 19 min. 52 s. | 4 | 1.6 s. | 2.9 | 9.9 s. |
| | MergeSplit | | | 611 | 6 | 21 min. 24 s. | 3.8 | 1.7 s. | 2.9 | 10.2 s. |
| | Nonconsecutiveness | | | 538 | 5.9 | 12 min. 43 s. | 3.7 | 1.8 s. | 2.8 | 9.3 s. |
| Real Datasets | AS | 15 | 6521 | 23 | 2.2 | 19.9 s. | 1.9 | 0.92 s. | 1.3 | 1.5 s. |
| | DBLP | 13 | 7672 | 357 | 10.5 | 6 h. 6 min. | 3.7 | 1.56 s. | 3.6 | 21 s. |
| | Yelp | 10 | 2344970 | 6847 | x | x | 352.5 | 2.47 s. | 60.9 | 11.2 s. |
| | 2009 Digg friends | 12 | 64183 | 329 | 7.9 | 44 min. 46 s. | 10.3 | 5.1 s. | 3.8 | 12.6 s. |

**TABLE 9.** Complexity orderings of TREC and competitor methods.

| | |
|---|---|
| Time Complexity Ordering | $\mathcal{O}(n\log(n) < \mathcal{O}(cn\log(n)) < \mathcal{O}(c^2\log(c))$<br>ICEM with Evolution Chain <TREC <Tajeuna et al.'s method |
| Space Complexity Ordering | $\mathcal{O}(n+v) < \mathcal{O}(2n) < \mathcal{O}(c^2)$<br>TREC <ICEM with Evolution Chain <Tajeuna et al.'s method |

As shown in Table 8, the number of vertices (n) and the number of time steps (t) are the same in the benchmark datasets. For the benchmark datasets, the memory requirement of the TREC method is almost the same even if the number of communities (c) is increased, since the communities are represented by minhashing with small fixed-length signatures. The most notable difference in the execution time of the Tajeuna *et al.*'s method can be seen in the table. That is, The Tajeuna *et al.*'s method tends to consume more execution time and memory as the number of communities increases.

Large real-world networks, such as social networks or citation networks, can have hundreds of millions of members. Given the storage capacity of ordinary computers and laptops, it may not be possible to store and/or process community members. In such a situation, the ICEM method is not applicable because it must have a global hash map to track the actual locations of community members and build the similarity lists of communities during execution. TREC, on the other hand, is applicable in such a situation because the TREC method only needs to load the community members filtered by LSH buckets and identified as similar into main memory to determine the evolution of the communities. As shown in Table 8 ICEM can provide efficient results for datasets with a small number of communities, even when the dataset is large. Looking at the Yelp data in Table 8, ICEM requires 352.5 MB, while TREC requires 60.9 MB as memory space. This is a confirmation of TREC's efficient memory usage. However, for dynamic networks with a large number of communities with a large number of members, TREC may be more advantageous than ICEM, even for communities like Yelp.

## VI. CONCLUSION

This study aims to provide a memory-efficient community evolution tracking method that can identify all types of evolution events and is still highly accurate. The core idea behind the method is to reduce the memory space and execution time in the community matching phase by using LSH and Minhashing. This use enables a memory-efficient representation of communities and time-efficient filtering of potentially similar communities. To the best of our knowledge, this is the first work that uses LSH and minhashing in combination to track the evolution of communities.

We perform an experimental evaluation on benchmark and real datasets and compare the baseline work. The results show that TREC is helpful in tracking community evolution in low-memory environments without a loss in accuracy and time.

TREC can also adapt to different domain requirements or constraints by tuning the threshold for LSH. A higher threshold decreases the number of matching communities, and consequently, the number of tracked communities is reduced. Users can easily adjust their implementations according to the desired accuracy level, acceptable execution time, and available memory.

Jaccard Similarity computations become cumbersome on real datasets where the average number of community members is large. Therefore, TREC with alternative similarity measures is a valuable research direction. Moreover, TREC can be extended to work with overlapping community structures and machine learning models to predict the future evolution of communities.

## ACKNOWLEDGMENT

The authors thank an Associate Professor Belgin Ergenç Bostanoglu for constructive criticism of the study and the manuscript. In addition, they also thank an Assistant Professor Dr. Selma Tekir for proofreading the manuscript.

## REFERENCES

[1] A.-L. Barabási and M. Pósfai, "Communities," in *Network Science*, 1st ed. Cambridge, U.K.: Cambridge Univ. Press, 2016, ch. 9, sec. 9.1, p. 320. [Online]. Available: http://networksciencebook.com/chapter/9#introduction9

[2] A. Calvo-Armengol and Y. Zenou, "Social networks and crime decisions: The role of social structure in facilitating delinquent behavior," *Int. Econ. Rev.*, vol. 45, no. 3, pp. 939–958, Aug. 2004.

[3] D. Kempe, J. Kleinberg, and E. Tardos, "Maximizing the spread of influence through a social network," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, Washington DC, USA, 2003, pp. 137–146.

[4] S. B. Abdrabbah, R. Ayachi, and N. B. Amor, "Collaborative filtering based on dynamic community detection," in *Proc. Int. Workshop Dyn. Netw. Knowl. Discov.*, Nancy, France, 2014, p. 85—106.

[5] S. Wasserman and K. Faust, "Cohesive subgroups," in *Social Network Analysis: Methods and Applications*, 1st ed. New York, NY, USA: Univ. Cambridge Press, 1994, ch. 7, sec. 7.1, p. 251. [Online]. Available: http://www.asecib.ase.ro/mps/Social%20Network%20Analysis%20%5B1994%5D.pdf

[6] D. A. Luke and J. K. Harris, "Network analysis in public health: History, methods, and applications," *Annu. Rev. Public Health*, vol. 28, no. 1, pp. 69–93, Apr. 2007, doi: 10.1146/annurev.publhealth.28.021406.144132.

[7] J. O. Garcia, A. Ashourvan, S. Muldoon, J. M. Vettel, and D. S. Bassett, "Applications of community detection techniques to brain graphs: Algorithmic considerations and implications for neural function," *Proc. IEEE*, vol. 106, no. 5, pp. 846–867, May 2018, doi: 10.1109/JPROC.2017.2786710.

[8] P. Bródka, P. Kazienko, and B. Kołoszczyk, "Predicting group evolution in the social network," in *Proc. SocInfo*, Lausanne, Switzerland, 2012, pp. 54–67.

[9] B. Gliwa, P. Bródka, A. Zygmunt, S. Saganowski, P. Kazienko, and J. Koźlak, "Different approaches to community evolution prediction in blogosphere," in *Proc. IEEE/ACM Int. Conf. Adv. Social Netw. Anal. Mining*, Aug. 2013, pp. 1291–1298.

[10] M. Takaffoli, R. Rabbany, and O. R. Zaiane, "Community evolution prediction in dynamic social networks," in *Proc. IEEE/ACM Int. Conf. Adv. Social Netw. Anal. Mining (ASONAM)*, Aug. 2014, pp. 9–16.

[11] G. Diakidis, D. Karna, D. Fasarakis-Hilliard, D. Vogiatzis, and G. Paliouras, "Predicting the evolution of communities in social networks," in *Proc. 5th Int. Conf. Web Intell., Mining Semantics*, Jul. 2015, pp. 1–6.

[12] N. Dakiche, F. B.-S. Tayeb, Y. Slimani, and K. Benatchba, "Community evolution prediction in dynamic social networks using community features' change rates," in *Proc. 34th ACM/SIGAPP Symp. Appl. Comput.*, Apr. 2019, pp. 2078–2085.

[13] N. Dakiche, F. B.-S. Tayeb, Y. Slimani, and K. Benatchba, "Tracking community evolution in social networks: A survey," *Inf. Process. Manage.*, vol. 56, no. 3, pp. 1084–1102, May 2019, doi: 10.1016/j.ipm.2018.03.005.

[14] E. G. Tajeuna, M. Bouguessa, and S. Wang, "Tracking communities over time in dynamic social network," in *Proc. Int. Conf. Mach. Learn. Data Min. Pattern Recognit.*, New York, NY, USA, 2016, pp. 341–345.

[15] K. K. Mohammadmosaferi and H. Naderi, "Evolution of communities in dynamic social networks: An efficient map-based approach," *Expert Syst. Appl.*, vol. 147, Jun. 2020, Art. no. 113221, doi: 10.1016/j.eswa.2020.113221.

[16] S. Qiao, N. Han, Y. Gao, R.-H. Li, J. Huang, H. Sun, and X. Wu, "Dynamic community evolution analysis framework for large-scale complex networks based on strong and weak events," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 51, no. 10, pp. 6229–6243, Oct. 2021, doi: 10.1109/TSMC.2019.2960085.

[17] A. Z. Broder, "On the resemblance and containment of documents," in *Proc. Compress. Complex. Sequences*, 1997, pp. 21–29.

[18] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. 13th Annu. ACM Symp. Theory Comput. (STOC)*, 1998, pp. 604–613.

[19] J. Leskovec, A. Rajaraman, and J. D. Ullman, "Finding similar items," *Mining Massive Datasets*, 3rd ed.ed. Cambridge, U.K.: Univ. Cambridge Press, 2014, ch. 3, sec. 3.3, pp. 81–91. [Online]. Available: http://infolab.stanford.edu/~ullman/mmds/ch3n.pdf

[20] D. Greene, D. Doyle, and P. Cunningham, "Tracking the evolution of communities in dynamic social networks," in *Proc. Int. Conf. Adv. Social Netw. Anal. Mining*, Aug. 2010, pp. 176–183.

[21] S. Asur, S. Parthasarathy, and D. Ucar, "An event-based framework for characterizing the evolutionary behavior of interaction graphs," *ACM Trans. Knowl. Discovery From Data*, vol. 3, no. 4, pp. 1–36, Nov. 2009, doi: 10.1145/1631162.1631164.

[22] M. Takaffoli, J. Fagnan, F. Sangi, and O. R. Zaïane, "Tracking changes in dynamic information networks," in *Proc. Int. Conf. Comput. Asp. Soc. Netw.*, Salamanca, Spain, 2011, pp. 94–101.

[23] P. Bródka, S. Saganowski, and P. Kazienko, "GED: The method for group evolution discovery in social networks," *Social Netw. Anal. Mining*, vol. 3, no. 1, pp. 1–14, Mar. 2013, doi: 10.1007/s13278-012-0058-8.

[24] B. Gliwa, S. Saganowski, A. Zygmunt, P. Brodka, P. Kazienko, and J. Kozak, "Identification of group changes in blogosphere," in *Proc. IEEE/ACM Int. Conf. Adv. Social Netw. Anal. Mining*, Aug. 2012, pp. 1201–1206.

[25] F. Folino and C. Pizzuti, "An evolutionary multiobjective approach for community discovery in dynamic networks," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 8, pp. 1838–1852, Aug. 2014, doi: 10.1109/TKDE.2013.131.

[26] C. Guo, J. Wang, and Z. Zhang, "Evolutionary community structure discovery in dynamic weighted networks," *Phys. A, Stat. Mech. Appl.*, vol. 413, pp. 565–576, Nov. 2014, doi: 10.1016/j.physa.2014.07.004.

[27] J. He and D. Chen, "A fast algorithm for community detection in temporal network," *Phys. A, Stat. Mech. Appl.*, vol. 429, pp. 87–94, Jul. 2015, doi: 10.1016/j.physa.2015.02.069.

[28] C. Tantipathananandh, T. Berger-Wolf, and D. Kempe, "A framework for community identification in dynamic social networks," in *Proc. 13th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2007, pp. 717–726, doi: 10.1145/1281192.1281269.

[29] P. J. Mucha, T. Richardson, K. Macon, M. A. Porter, and J.-P. Onnela, "Community structure in time-dependent, multiscale, and multiplex networks," *Science*, vol. 328, no. 5980, pp. 876–878, May 2010, doi: 10.1126/science.1184819.

[30] B. Mitra, L. Tabourier, and C. Roth, "Intrinsically dynamic network communities," *Comput. Netw.*, vol. 56, no. 3, pp. 1041–1053, Feb. 2012, doi: 10.1016/j.comnet.2011.10.024.

[31] R. Cazabet, F. Amblard, and C. Hanachi, "Detection of overlapping communities in dynamical social networks," in *Proc. IEEE 2nd Int. Conf. Social Comput.*, Aug. 2010, pp. 309–314, doi: 10.1109/SocialCom.2010.51.

[32] N. P. Nguyen, T. N. Dinh, Y. Xuan, and M. T. Thai, "Adaptive algorithms for detecting community structure in dynamic social networks," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 2282–2290, doi: 10.1109/INFCOM.2011.5935045.

[33] G. Rossetti, L. Pappalardo, D. Pedreschi, and F. Giannotti, "Tiles: An online algorithm for community discovery in dynamic social networks," *Mach. Learn.*, vol. 106, no. 8, pp. 1213–1241, 2017, doi: 10.1007/s10994-016-5582-8.

[34] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *J. Stat. Mech., Theory Exp.*, vol. 2008, no. 10, Oct. 2008, Art. no. P10008, doi: 10.1088/1742-5468/2008/10/P10008.

[35] Z. Yang, R. Algesheimer, and C. J. Tessone, "A comparative analysis of community detection algorithms on artificial networks," *Sci. Rep.*, vol. 6, no. 1, p. 68, Aug. 2016, doi: 10.1038/srep30750.

[36] A. Karataş and S. Sahin, "A comparative study of modularity-based community detection methods for online social networks," in *Proc. Turk. Nat. Softw. Eng. Symp.*, Istanbul, Turkey, 2018, p. 68.

[37] B. P. Chamberlain, J. Levy-Kramer, C. Humby, and M. P. Deisenroth, "Real-time community detection in full social networks on a laptop," *PLoS ONE*, vol. 13, no. 1, Jan. 2018, Art. no. e0188702, doi: 10.1371/journal.pone.0188702.

[38] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: Densification laws, shrinking diameters and possible explanations," in *Proc. 11th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2005, pp. 177–187.

[39] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "ArnetMiner: Extraction and mining of academic social networks," in *Proc. 14th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2008, pp. 990–998.

[40] (May 12, 2019). *Yelp Open Dataset an All-Purpose Dataset for Learning*. Dataset. [Online]. Available: https://www.yelp.com/dataset/

[41] T. Hogg and K. Lerman, "Social dynamics of Digg," *EPJ Data Sci.*, vol. 1, no. 1, pp. 1–26, Dec. 2012, doi: 10.1140/epjds5.

[42] K. Lerman. (Oct. 8, 2020). *Digg 2009 Dataset*. Dataset. [Online]. Available: https://www.isi.edu/ lerman/downloads/digg2009.html

[43] S. M. van Dongen, "Graph clustering by flow simulation," Ph.D. dissertation, Cent. Wiskd. Inform., Utrecht Univ., Utrecht, NL, USA, 2000. Accessed: Apr. 10, 2022. [Online]. Available: http://dspace.library.uu.nl/bitstream/handle/1874/848/full.pdf?sequence=1&isAllowed=y

[44] J. Chen, O. Zaïane, and R. Goebel, "Local community identification in social networks," in *Proc. Int. Conf. Adv. Social Netw. Anal. Mining*, Jul. 2009, pp. 237–242.

[45] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *Proc. Nat. Acad. Sci. USA*, vol. 105, no. 2, pp. 1118–1123, 2008, doi: 10.1073/pnas.0706851105.

[46] V. A. Traag, L. Waltman, and N. J. van Eck, "From Louvain to leiden: Guaranteeing well-connected communities," *Sci. Rep.*, vol. 9, no. 1, pp. 1–12, Mar. 2019, doi: 10.1038/s41598-019-41695-z.
[47] L. Danon, A. Díaz-Guilera, J. Duch, and A. Arenas, "Comparing community structure identification," *J. Stat. Mech., Theory Exp.*, vol. 2005, no. 9, Sep. 2005, Art. no. P09008, doi: 10.1088/1742-5468/2005/09/P09008.
[48] A. Lancichinetti, S. Fortunato, and J. Kertész, "Detecting the overlapping and hierarchical community structure in complex networks," *New J. Phys.*, vol. 11, no. 3, Mar. 2009, Art. no. 033015, doi: 10.1088/1367-2630/11/3/033015.
[49] (2019). *Visual Studio 2019 Community Edition*. Microsoft Corporation. Accessed: Apr. 10, 2022. [Online]. Available: https://visualstudio.microsoft.com/tr/vs/community/

**ARZUM KARATAS** (Member, IEEE) received the A.A.S. degree in computer technology and programming from Dokuz Eylul University, İzmir, Turkey, in 2004, as ranked first degree, and the B.S., M.S., and Ph.D. degrees in computer engineering from the İzmir Institute of Technology, İzmir, in 2012, 2015, and 2021, respectively, as ranked in the Rector's List.

She was a Teaching Assistant at the Department of Computer Engineering, Gediz University, from 2013 to 2016. Her research interests include community tracking and its applications, complex networks, and machine learning applications.

Dr. Karatas has been a member of ACM, since 2014; and a member of UCTEA Chamber of Computer Engineers, since 2016.

**SERAP SAHIN** (Member, IEEE) was born in 1966. She received the B.S. degree in computer engineering from Ege University, Turkey, in 1987, with a thesis on "Expert Systems," the Master of Engineering degree from Ege University, in 1989, with a thesis on "System Analysis and Design of Arkas Holding Information System," and the Ph.D. degree in information systems security and cryptology from the Information Systems Strategy and Security Laboratory, İzmir Institute of Technology, Turkey, on the "Problem of Computational Speed of Elliptic Curve Cryptosystems in Software Implementation."

She started her academic studies, in 2002, and completed her Ph.D. thesis. From 1987 to 2002, she worked in industry and carried out numerous projects in the fields of maritime, air and rail transport, port management, and logistics. After her Ph.D. degree, she completed her postdoctoral research on the project 'Security of Next Generation Networks' at the Joint Research Center, Institute for the Protection and Security of the Citizen, Italy, with the support of the Postdoctoral Research Fellowship Program of TUBITAK, Turkey, between September 2008 and 2009. Since 2010, she has been working as an Assistant Professor at the Department of Computer Engineering, İzmir Institute of Technology.