

TRACKING AND PREDICTION OF EVOLUTION OF COMMUNITIES IN DYNAMIC NETWORKS

**A Thesis Submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
DOCTOR OF PHILOSOPHY
in Computer Engineering**

**by
Arzum KARATAŞ**

**July 2021
İZMİR**

ACKNOWLEDGMENTS

I would like to thank my thesis supervisor, Asst. Prof. Dr. Serap ŞAHİN, for her supervision, her respect for my individual opinions along the thesis, and for giving me the opportunity to conduct such a joyful study. I appreciate her sincerity, patience and attentive style.

Also, I had the pleasure of working with Assoc. Prof. Dr. Belgin ERGENÇ BOSTANOĞLU. She was the one who uplifted me when I was in trouble with critical decisions. Her valuable support and confidence were the driving force for my courage and enthusiasm.

Next, I should thank Asst. Prof. Dr. Mutlu BEYAZIT for his time and effort spent on my dissertation. He was always kind and gave me constructive feedback. It was a real pleasure to work with him.

I would also like to thank the Scientific and Technological Research Council of Turkey (TÜBİTAK) for supporting me through 2211-C PhD Support Scholarship Program.

In addition, I would also like to thank Turkey Council of Higher Education (YÖK) for supporting me through 100/2000 Doctoral Scholarship Program.

Finally, I would like to thank my family and everyone who contributes to my life by staying with me on good and bad days. It is a great blessing to know them and be close to them.

ABSTRACT

TRACKING AND PREDICTION OF EVOLUTION OF COMMUNITIES IN DYNAMIC NETWORKS

Communities are the most meaningful structures in dynamic networks. Tracking this evolution provides insights into the patterns of community evolution in networks over time and valuable information for decision support systems in many research areas such as marketing, recommender systems, and criminology. Previous work has focused on either high accuracy or time efficiency, but not on low memory consumption. This motivates us to develop a method that combines highly accurate tracking results with low computational resources.

This dissertation first provides a brief overview of research in dynamic network analysis. Then, a novel space-efficient method, called TREC, for tracking the evolution of communities in dynamic networks is presented, where community matching using LSH with minhashing technique is proposed to efficiently track similar communities in terms of memory consumption over time. The accuracy of TREC is evaluated on benchmark datasets, and the execution time performance is measured on real dynamic datasets. In addition, a comparative algorithmic complexity analysis of TREC in terms of space and time is performed. Both theoretical and experimental results show that TREC outperforms competitor methods on both datasets in terms of combination of space, accuracy, and execution time.

Next, it is investigated that whether the TREC method is suitable for predicting the evolution of community areas. In this evaluation, a prediction study is conducted. A common methodology is followed which includes main steps such as feature extraction, feature selection, classifier training and cross validation. Experimental results show that TREC method is suitable for predicting evolution of communities.

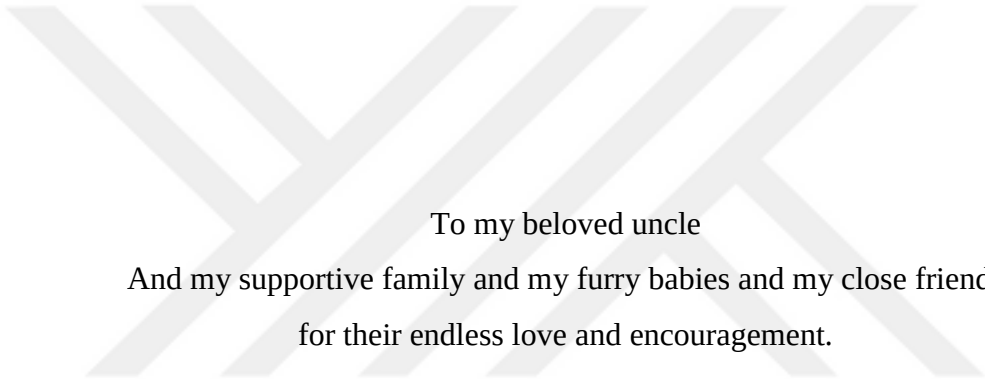
ÖZET

DİNAMİK AĞLARDA TOPLULUKLARIN GELİŞİMİNİN İZLENMESİ VE KESTİRİMİ

Topluluklar dinamik ağlarda karşılaşılan anlamlı yapılardır. Bu ağlardaki toplulukların zaman içerisindeki olası gelişimlerinin takibi farklı alanlardaki araştırma ve karar destek sistemleri için değerli bilgiler sağlar; örneğin bilimsel araştırmalarda, sosyal ağlarda ilgi alanlarındaki değişimin incelenmesi ve suç kestiriminin sağlanmasında, reklam ve pazarlama sistemlerinin yönlendirilmesinde vb. Var olan çalışmalar ya yüksek başarıya ya da zaman verimliliğine odaklanmış, bellek kullanım verimliliği incelenmemiştir. Dolayısıyla, toplulukların gelişimini düşük hesaplama kaynağı kullanarak yüksek başarıyla izleyebilecek bir yöntem geliştirme bu tezin motivasyonunu oluşturur.

Bu doktora tezinde, dinamik bir ağda, benzer toplulukları takip etmek ve benzerlik ilişki tipini belirlemede TREC adında özgün bir yöntem önerilmiştir. Yöntem, bellek kullanım verimliliği için LSH ve minhashing tekniği kullanılarak topluluk eşleştirmesi yapar. Önerilen TREC yönteminin sonuçlarına ait doğrulama ve çalışma zamanı gibi verimlilik analizleri gerçekleştirilmiştir. Yöntemin, benzer ve güncel olan en iyi çalışmalar ile karşılaştırılması ise; hem deneysel uygulamalar ile hem de kullanılan zaman ve bellek alanı açısından algoritmik karmaşıklık analizleri ile sağlanmıştır. Sonuçlar; TREC yönteminin bellek alanı gereksinimi, doğruluk ve çalışma zamanı tüketiminin kombinasyonu ile hem deneysel hem de gerçek veri setlerinde benzer çalışmalara göre üstünlük içerdiğini göstermiştir.

Tezin ana çalışmasına ek olarak; toplulukların gelişiminin kestiriminde TREC yönteminin uygulanabilirliği de değerlendirilmiş, makine öğrenmesine dayalı kestirimci bir çalışma yürütülmüştür. Bu aşamada, yeni bir makine öğrenmesi yöntemi geliştirilmesi ya da yeni bir yöntem-bilim önerilmesi hedeflenmemiştir. Bu nedenle, özneteliklerin belirlenmesi, özneteliklerin seçilmesi, sınıflandırıcıların eğitilmesi ve çapraz geçerlilik ana basamaklarını içeren yaygın bir yöntem-bilim izlenmiştir. Buradaki sonuçlar; TREC yönteminin kestirim alanında benzer çalışmalar ile eş başarımlı düzeyinde olduğunu ve uygulanabilir olduğunu göstermektedir.



To my beloved uncle
And my supportive family and my furry babies and my close friends
for their endless love and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES	viii
LIST OF TABLES	x
 CHAPTER 1. INTRODUCTION	1
1.1. Contributions of The Thesis	4
1.2. Organization of The Manuscript	5
 CHAPTER 2. BASIC CONCEPTS	7
2.1. Graphs	7
2.2. Community Structure	9
2.3. Dynamic Networks	13
 CHAPTER 3. TRACKING EVOLUTION OF COMMUNITIES IN DYNAMIC NETWORKS	15
3.1. Background and Problem Formulation	15
3.1.1. Concepts and Definitions	15
3.1.2. Problem Formulation	21
3.1.3. Evaluation Criteria for Methods for Tracking Evolution of Communities	21
3.1.5. Locality Sensitive Hashing for Minhash Signatures	29
3.2. Related Work	33
3.3. Methodology	42
3.4. Experimental Study	50
3.4.1. Datasets	50
3.4.2. Experimental Configuration	52
3.4.3. Impact of Using Minhashing and LSH	52
3.4.4. Performance and Evaluation of TREC	53
3.4.5. Discussion On the Results	58
3.5. Concluding Thoughts and Future Work	61

CHAPTER 4. A CASE STUDY: PREDICTING COMMUNITY EVOLUTION WITH RESULTS OF TREC METHOD	63
4.1. Predicting The Evolution of Communities	63
4.2. Related Work	64
4.3. Workflow for Prediction Community Evolution	69
4.4. Experimental Results	82
4.4.1. Discussion On the Results	84
4.4.2. Concluding Thoughts	85
CHAPTER 5. CONCLUSION	87
APPENDICES	
APPENDIX A. A TAXONOMY OF METHODS FOR TRACKING EVOLUTION OF COMMUNITIES	101
APPENDIX B. DECIDING GROUND-TRUTH EVENTS	114
APPENDIX C. FEATURE DETERMINATION SUBPROCESS	125
APPENDIX D. SELECTED MACHINE LEARNING CLASSIFIERS	129

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
Figure 1.1. An illustration of a sample social network modelled by a graph in which nodes represent people and edges represent their friendship	1
Figure 1.2. An example of illustrating different types of communities	2
Figure 1.3. A scenario for modifications over time for a single community	3
Figure 2.1. An illustration of a toy (a) friendship graph and (b) follower graph	7
Figure 2.2. Representation of a map with graphs	8
Figure 2.3. An example of illustrating disjoint (nonoverlapping) and overlapping communities	10
Figure 2.4. An abstract illustration two main phases of Louvain algorithm.....	12
Figure 2.5. A sample scenario for possible community events	13
Figure 3.1. A simple illustration of the evolution of communities	16
Figure 3.2. An illustration of community evolution events where i and j represent time steps where $i < j$	17
Figure 3.3. A simple illustration of the consecutively evolving communities	20
Figure 3.4. A simple illustration of the nonconsecutively evolving communities	21
Figure 3.5. Main steps of process of how to compute minhash signatures	23
Figure 3.6. An adjacency matrix, A , representing sample sets (e.g., groups or communities) and their respective members	24
Figure 3.7. The computed hash functions for the matrix of Figure 3.6.	25
Figure 3.8. Initialized signature matrix, $\text{sig}[]$	25
Figure 3.9. Illustration of how LSH works	30
Figure 3.10. Classification chart of the existing tracking community evolution methods in dynamic social networks	35
Figure 3.11. An illustration of replacement of TREC method under Event-based Methods in Dakiche et al.'s taxonomy	37
Figure 3.12. Main steps of tracking evolution of communities	42
Figure 3.13. Main steps of TREC method	43
Figure 3.14. Community vectors representation for time steps	44
Figure 3.15. Vector representation of communities	45

Figure 3.16. An example of signature matrix	46
Figure 3.17. An example of “how LSH with minhashing works”	47
Figure 3.18. A Pseudocode of how to track similar communities over time	48
Figure 3.19. An example of (a) content of tListFile (b) its correspondent conceptual schema	49
Figure 3.20. (a) Accuracy and (b) running time consumptions of TREC, minhashing_effect_TREC and LSH_effect_TREC methods	53
Figure 3.21. Accuracy values of the TREC and its competitors on the datasets	58
Figure 4.1. Fundamental steps of predicting community evolution	63
Figure 4.2. The workflow we follow for this case study	70
Figure 4.3. A sample evolution for community	71
Figure 4.4. K-fold Cross Validation process	79
Figure 4.5. Visualization of an example of 5-fold Cross Validation	80
Figure 4.6. Prediction task inputs and outputs in WEKA	81
Figure 4.7. A sample training set for predicting merge event for WEKA	81
Figure 4.8. A sample output for decision task in WEKA	82
Figure A.1. Independent community detection and matching approach	102
Figure A.2. Dependent community detection approach	106
Figure A.3. Simultaneous Community Detection approach	109
Figure A.4. Dynamic Community Detection on Temporal Networks approach	111
Figure A.5. Input and output of the ground-truth event determination process	115
Figure A.6. ERD diagram of the database application	115
Figure A.7. A pseudocode for the process to determine ground-truth events	116
Figure A.8. An example of “how Ibk (k-NN) works”	129
Figure A.9. Frequency tables and information gains of the features in the Weather dataset	132
Figure A.10. Visualization of the decision tree created by J48 classifier in WEKA ...	132
Figure A.11. An illustration of decision process for an example random forest classifier	133

LIST OF TABLES

<u>Table</u>	<u>Page</u>
Table 3.1. Definition and illustration of community evolution events	18
Table 3.2. Jaccard similarity for original matrix and approximate Jaccard Similarity for signature matrix sig[]	29
Table 3.3. Probabilities of the signature pairs detected by LSH with respect to the λ ...	32
Table 3.4. Overview of the mainstream and latest competitor methods	40
Table 3.5. Specific parameters used in the accuracy experiment	51
Table 3.6. Complexity analysis of TREC and competitor methods	55
Table 3.7. Accuracy scores of the methods per datasets	56
Table 3.8. The highest memory usage in Mega Bytes during their executions and Execution Time of the methods	57
Table 3.9. Complexity orderings of TREC and competitor methods	59
Table 4.1. Overview of the mainstream approaches for predicting community evolutions	67
Table 4.2. Name of the Classifiers and their correspondents in WEKA	77
Table 4.3. Event frequencies according to the chain lengths of the benchmark dataset.	77
Table 4.4. Selected features that gives the highest success score per future event	78
Table 4.5. The highest prediction success values in terms of F-measure per events	83
Table 4.6. Overall performance of the classifiers with respect to chain lengths	84
Table 4.7. The highest success in terms of accuracy in percentage with specific chain length and classifier	84
Table 5.1. Our goals of a method for evolution of communities tracking	88
Table A.1. Structural features of the communities	125
Table A.2. Temporal features of the communities	127
Table A.3. Leadership features of the communities	128
Table A.4. Codes of Values of Event feature	128
Table A.5. Weather.arff file	130

CHAPTER 1

INTRODUCTION

Many real-world systems, such as communication networks, biological networks, and social networks, can be represented as complex networks in the digital world. Complex networks can be described in terms of graph structures consisting a set of nodes (i.e., elements in the network) and edges (i.e., connections between nodes). Recently, Dynamic Network Analysis (DNA) is drawing attention due to the tremendous increase in popularity and importance of dynamic networks such as social networks, scientific collaboration, and biological networks. Social networks are used to represent member relationships or interactions in the networks. Existing social networks provide rich and valuable information about their members. One of their main goals is to understand the relationships and interactions within the network over time. Figure 1.1 represents a sample social network in the form of a graph.



Figure 1.1. An illustration of a sample social network modelled by a graph in which nodes represent people and edges represent their friendship (Source: Arredondo 2021)

One of the properties of complex networks is that they inherently contain a community structure. The community structure observed in the network can be of different natures, e.g. disjoint (nonoverlapping), overlapping, hierarchical and local. The disjoint community structure includes communities without overlap, as shown in Figure 1.2. (a). That is, the members of this type of communities can be assigned to only one group. The overlapping community structure represents that a member of any community can have one or more memberships in other communities, as in Figure 1.2. (b). That is, a person can be a member of different interest groups in an online social network. The hierarchical community structure shows hierarchical grouping levels, as in Figure 1.2. (c). As for local communities, they show a different structure from a local point of view, but no structure from a global point of view, as in Figure 1.2. (d). In the context of this paper, the focus will be on disjoint community structures, as this is the most common community structure.

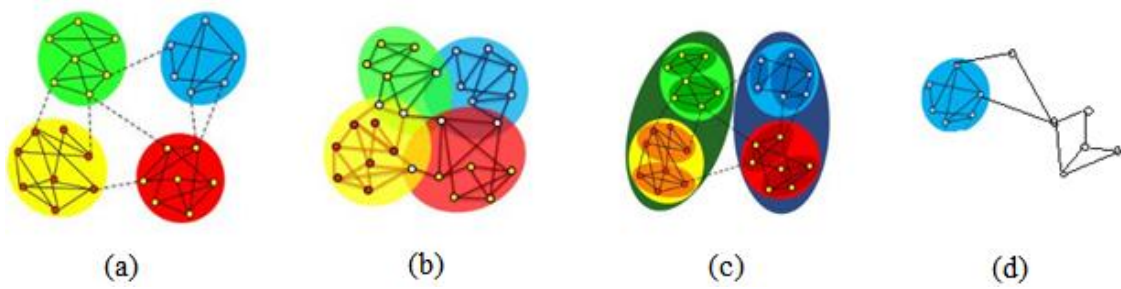


Figure 1.2. An example of illustrating different types of communities: (a) disjoint, (b) overlapping, (c) hierarchical and (d) local communities (Source: Karataş and Şahin 2018a)

Since the diversity of the nature of communities in the given network is not known in advance and depends on the domain, the definition of community is an ill-defined concept (Fortunato 2010). Nevertheless, a commonly accepted definition of community according to the structure of the network is that within the community, members are strongly connected and across the community, members are loosely connected (Girvan and Newman 2002). Communities can be formed not only by structural similarities but also by functional similarities between members of the network (Newman 2004).

Therefore, recognizing community structure provides us with meaningful insights into network structure and its organizing principle.

Community detection is the task of revealing the community structure of a snapshot of a network for a given time interval. It allows us to look at a mesoscopic level (i.e., group level). Therefore, there are many application domains where group-level tasks are performed. For example, community detection is used for market segmentation, community profiling, recommender systems, and more. The detailed application areas of community detection can be found in Reference (Karataş and Şahin 2018a).

In real networks, the members and/or the relationships between members may change over time. It is obvious that the graph in Figure 1.1 cannot represent the evolution of networks over time. That is, one cannot infer the new/leaving members and established/terminated relationships from the static graph in the figure. Therefore, the dynamic network concept was developed to model the temporal evolution of networks over time (Rossetti and Cazabet 2018, Cazabet and Rossetti 2019). Moreover, when the network updates, the community structure of the network changes. That is, communities in the network may grow or shrink, new communities may even emerge, while some of them may disappear. An example scenario of possible community changes can be seen in Figure 1.3. Initially, there are some nodes with no connection in T_1 . Then, they create a community in T_2 . More members join the group; therefore, the community grows in T_3 . In T_4 , the community splits into two new groups; therefore, it is divided. The rest of the scenario shows the possible events that a community can undergo.

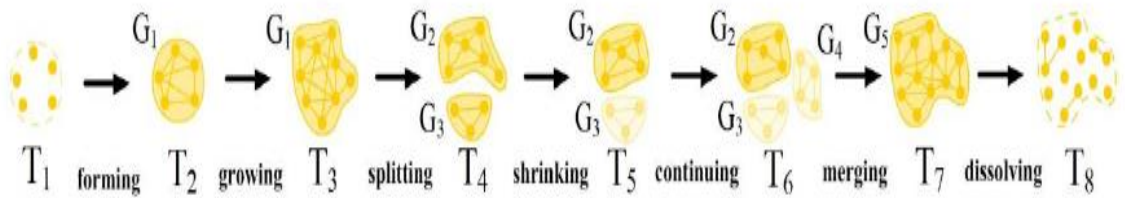


Figure 1.3. A scenario for modifications over time for a single community (Source: Bródka, Saganowski, and Kazienko 2014)

As communities evolve over time, detecting and tracking this evolution provides interesting and valuable information for decision support systems in many research areas such as criminology (Calvó-Armengol and Zenou 2004, Ferrara et al. 2014, Calderoni, Brunetto, and Piccardi 2017), marketing (Kempe, Kleinberg, and Tardos 2003), recommender systems (Zanin et al. 2008), and public health (Zhu et al. 2012, Fan, Yeung, and Wong 2013).

The most common strategy for identifying community evolution is to decompose network data into time steps and identify community structure using a community detection method such as Louvain (Blondel et al. 2008), CPM (Palla et al. 2005), Leiden (Traag, Waltman, and van Eck 2019), and Infomap (Rosvall and Bergstrom 2008). Many methods for characterizing community evolution focus on identifying evolutionary event types (i.e., “form”, “continue”, “grow”, “shrink”, “merge”, “split” and “dissolve”) and then examine the occurrence of these events.

Despite all the work done so far, there are still problems. Previous methods for tracking the evolution of communities in dynamic networks focus on the accuracy of the tracking results and their execution time is generally high. However, in the world of dynamic networks, low resource consumption is as important as accuracy, and none of the existing work touches the problem of developing a novel method with a combination of low space consumption, highest accuracy (currently it is 98%) and reasonable execution time for tracking community evolution.

1.1. Contributions of the Thesis

Before this section, a brief introduction to the problem area and the open problems of tracing the evolution of communities is given. This subsection lists the main contributions of this thesis.

The main objective of this thesis is to efficiently track the evolution of communities in dynamic networks. There is a supplementary goal in this thesis, which is to apply the solution provided for the main objective in predicting the evolution of communities. In this dissertation, a novel efficient tracking method TREC (TRacking Evolution of Communities) is proposed that is both resource efficient and at least as

accurate as competing works. The community detection method used does not matter as long as the detection method is used to identify the disjoint community structure.

The main contributions of this thesis are listed as follows.

- A novel method TREC, short for TRacking Evolution of Communities, for tracking the evolution of communities in dynamic networks is presented. It uses a combination of two probabilistic techniques such as Locality Sensitive Hashing (LSH) (Indyk and Motwani 1998) and minhashing (Broder 1997).
- LSH with minhashing technique is used for the first time for tracking the evolution of communities.
- The high resource consumption of previous work arises from the community matching problem. By using LSH with minhashing, the inefficiency of memory consumption in the community matching task is solved.
- The efficiency and computational limitations of TREC is guaranteed by complexity analysis.
- Creation of a ground truth event dataset to evaluate the prediction success.

1.2. Organization of the Manuscript

This subsection presents the outline of the manuscript of this dissertation. The main work developed in this dissertation is tracking the evolution of communities. The supplementary work to this is the application of the method for the main work to predict the evolution of communities in dynamic networks.

Chapter 2 presents the basics of community analysis starting from the graph data structure.

Chapter 3 focuses entirely on tracking the evolution of communities in dynamic networks. First, the problem domain is introduced, then a literature review of competing works is given. Then, the novel TREC method is presented as a solution. A performance analysis of the TREC method is then performed, both theoretically and experimentally. The theoretical analysis is performed with complexity analysis, while the practical analysis is performed with accuracy analysis and real-time memory and execution time

analysis for both benchmark datasets and real datasets such as AS, DBLP, Yelp and 2009 Digg friendship. Finally, the results of the analysis are evaluated and the chapter is concluded.

Chapter 4 explains our case study using the TREC method to predict community evolution, and does not attempt to introduce any novelty in terms of a machine learning method or a prediction method. That is, Chapter 4 is an evaluation chapter that assesses the feasibility of using tracking results of the TREC method in predicting community evolution. Therefore, the field of community evolution prediction is first introduced. Then, a workflow for predictive analysis is presented. Then, the experimental study and preliminary results are presented. Finally, the results are discussed and brief concluding thoughts are given.

Chapter 5 concludes the thesis by summarizing the main contributions, discussing about the thesis, and pointing to possible research directions that have emerged from this study.

CHAPTER 2

BASIC CONCEPTS

Computers are becoming more ubiquitous in our daily lives, in many forms such as mobile devices and even wearables. In particular, it is hard to imagine our lives without smart applications ranging from tracking and suggestion systems to health apps. When this is the case, our world is becoming more and more connected. Nearly all real-world systems, including social, biological, and technological systems, are represented by complex networks. These complex networks are represented with graphs in the digital world.

2.1. Graphs

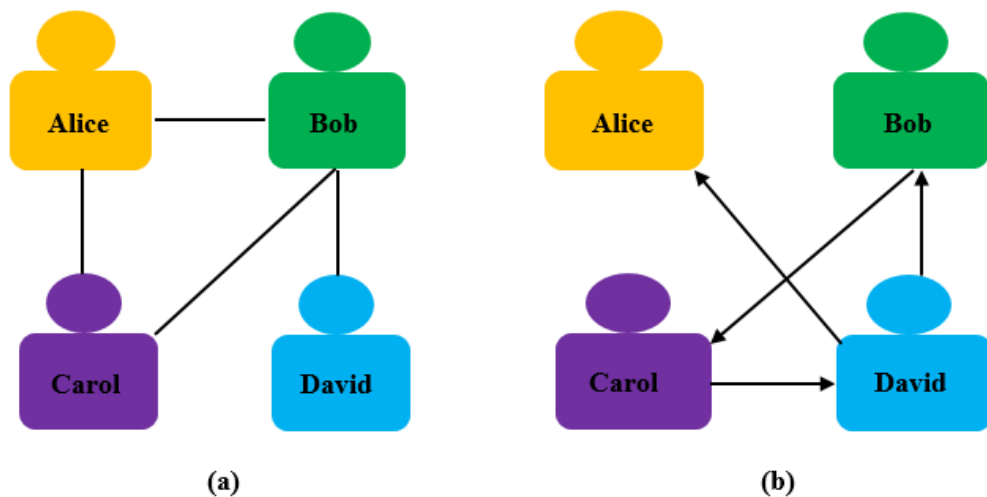


Figure 2.1. An illustration of a toy (a) friendship graph and (b) follower graph

In discrete mathematics, especially in graph theory, a graph is an abstract data structure for a set of objects that represents the relations pairwise in some sense. The objects are called vertices or nodes, and the pairs that belong together in each case are called edges or links. Depending on the real-world system being represented, the vertices and edges can represent different things. For example, if a graph is used to represent a social network, then the vertices and edges correspond to people and friendship/interaction, respectively. If it is a protein-protein interaction network, the vertices correspond to proteins and the edges correspond to interactions between pairs of proteins.

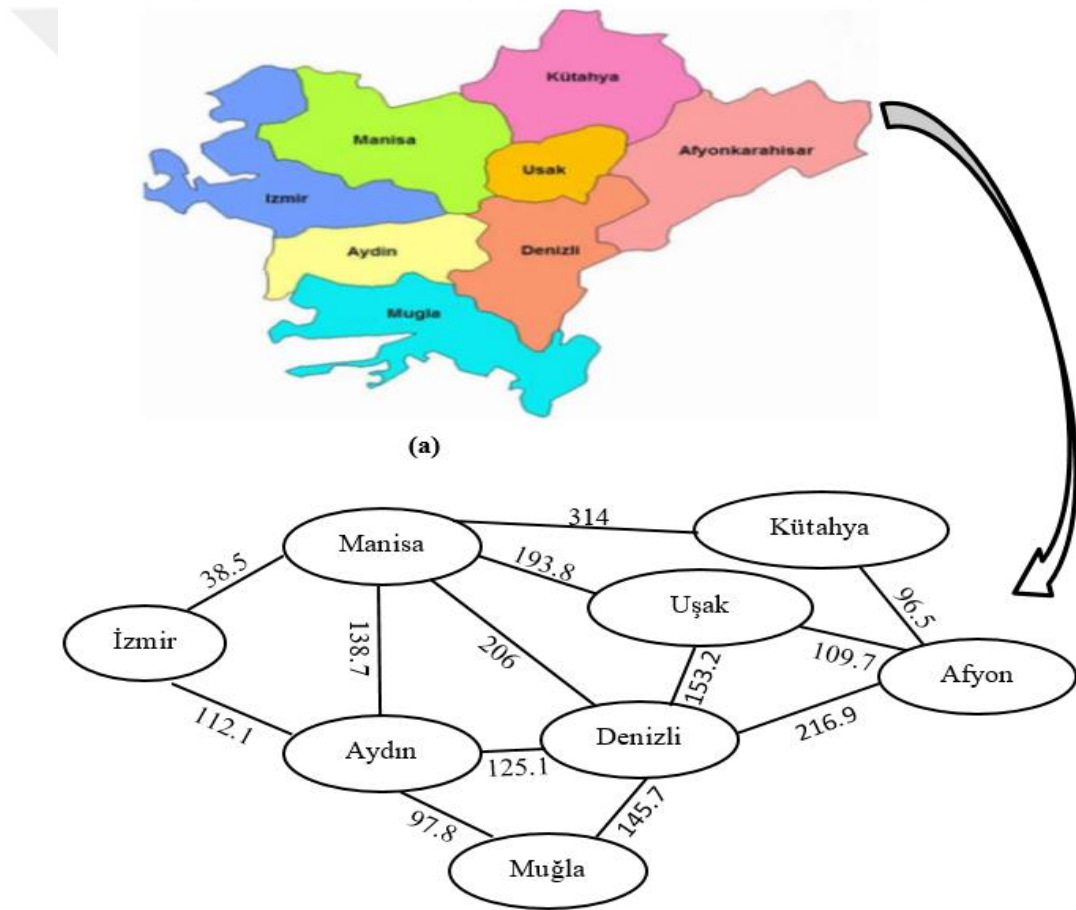


Figure 2.2. Representation of a map with graphs (a) Geographical Map showing the Aegean region of Turkey (b) A weighted graph representation of the region where weights are the shortest distances in kilometers

Edges can be **directed** or **undirected**. Figure 2.1(a) and Figure 2.2(b) show two toy graphs as examples of undirected and directed graphs, respectively. If there is symmetry between the pairs, the edges are undirected. For example, friendship is a symmetric relationship. As shown in Figure 2.1(a), Alice is friends with Bob and Bob is friends with Alice. However, not all relationship types have symmetric/shared relationships, such as following-followed, sending short messages, and emails. Therefore, the interaction between vertices must be directed. As shown in Figure 2.1(b), Carol follows David, but this does not mean that David follows Carol, and the direction of the arrow on the edge shows who follows whom. Therefore, the interaction between pairs need not be mutual/symmetric.

Graphs can be **weighted** or **unweighted**. A weighted graph can be defined as a graph in which each edge is assigned a weight (can be an integer or a real number). If there are no weights on the edges or the weights of all edges are equal, then the graph is considered an unweighted graph. It depends on the problem being modeled whether a graph is weighted or unweighted.

For example, if the problem is modeling a friendship relation, which can be seen in the network in Figure 2.1(a), then all relations have the same importance (weight) and there is no need to assign weights to the edges. Another example: If the problem is to find the most efficient route in terms of distance between destinations that a person should visit given a list of certain destinations, then the distances between destinations need not be equal. Therefore, weighted graphs are suitable for solving this problem, where the weights represent the distances between pairs of destinations. To illustrate, suppose there is a salesman who needs to visit each city in the Aegean Region of Turkey shown in Figure 2.2(a). He is supposed to find the most efficient route for himself in terms of distances between cities (this problem is well known in theoretical computer science and computational mathematics and is called Traveling Sales Man). Therefore, one of the solutions is to model the problem as a weighted graph, where the nodes are the cities of the region, the edges are the links between the cities, and the edge weights are the labels indicating the distances between pairs of cities, as shown in Figure 2.2(b).

2.2. Community Structure

Complex networks such as communication networks, biological networks, and social networks inherently exhibit community structure. There is no universal definition for the term "community" (Fortunato, 2010). On the other hand, it is informally accepted as a subset of nodes that are closely connected to the rest of the network, and a community should have at least three members. Depending on the nature of the network, the community structure may consist of disjoint or overlapping communities. Figure 2.3(a) shows a simple illustration of nonoverlapping (disjoint) communities, where each node is a member of only one community. As can be seen in the figure, there are three communities, Community A, Community B, and Community C, and each member can belong to only one of them. Overlapping communities are the communities whose nodes are allowed to be members of more than one community. Figure 2.3(b) shows a simple illustration of overlapping communities, where Community A and Community B overlap, and Community B and Community C overlap. The overlapping nodes are shown with red circles.

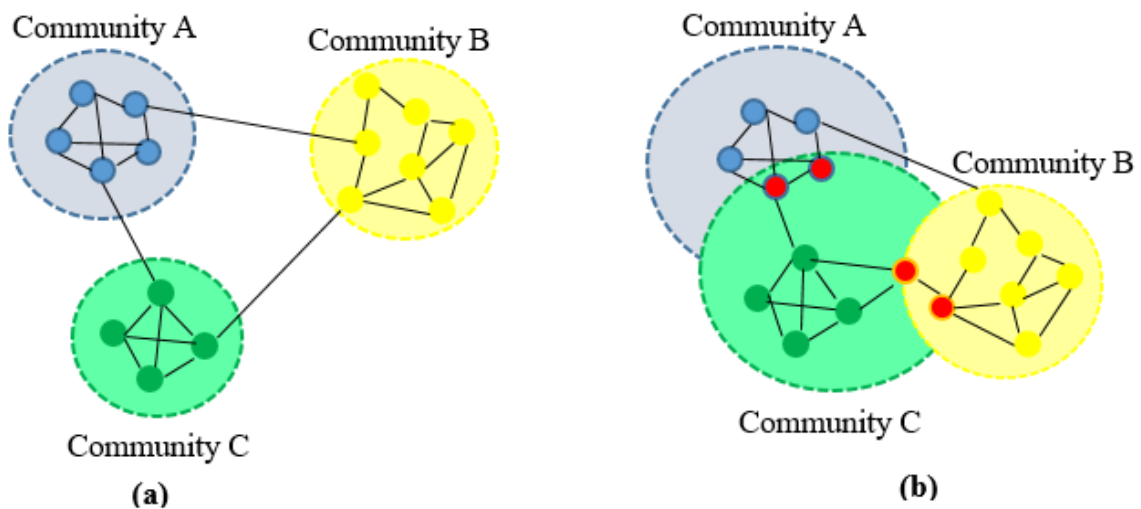


Figure 2.3. An example of illustrating disjoint (nonoverlapping) and overlapping communities

Community detection is the task of uncovering the community structure of a snapshot of a network. It is a powerful tool to view a network from a mesoscale (group-level) perspective. Therefore, it draws attention of many researchers from many different fields. For example, it is used for market segmentation, community profiling, recommender systems, and cybercrime detection, etc. Since the optimal selection of community members is a combinatorial problem, the exact solution of community detection can be NP-hard (Fortunato 2010). Therefore, heuristic solutions and approximation-based solutions are suitable for the problem. Both application areas of community detection are discussed in the paper (Karataş and Şahin 2018a) and existing methods for community detection are summarized.

The most common community detection methods are based on modularity optimization because of its ease of implementation and low consumption of execution time. Modularity (Q) is a metric that measures the modularity of a community structure. It is determined by the ratio of the high number of intra-community connections to the expected inter-community connections. It is formulated as in equation (2.1) (Chakraborty et al. 2017).

$$Q = \frac{1}{2e} \sum_{i,j} \left[A_{ij} - \frac{d(i)d(j)}{2e} \right] \sigma_{i,j} \quad (2.1)$$

where A_{ij} is the adjacency matrix of the network snapshot, e is the number of edges in the graph, and $d(i)$ and $d(j)$ are the degrees of node i and j respectively. σ is the function that returns 1 if both node i and j in the same community, else returns 0. Modularity value lies between -1 and 1 where higher modularity values implies strongly connected community structure.

Although there are many methods based on modularity optimization, Louvain is one of the best methods in terms of its accuracy and low execution time consumption (Yang, Algesheimer, and Tessone 2016, and Karataş and Şahin 2018b). Moreover, its open-source code and application is easily accessible and can be found at the reference link (Waltman and Jan van Eck 2015). Blondel et al. (Blondel et al. 2008) present the Louvain algorithm. It uses a greedy local approach and performs a local moving heuristic

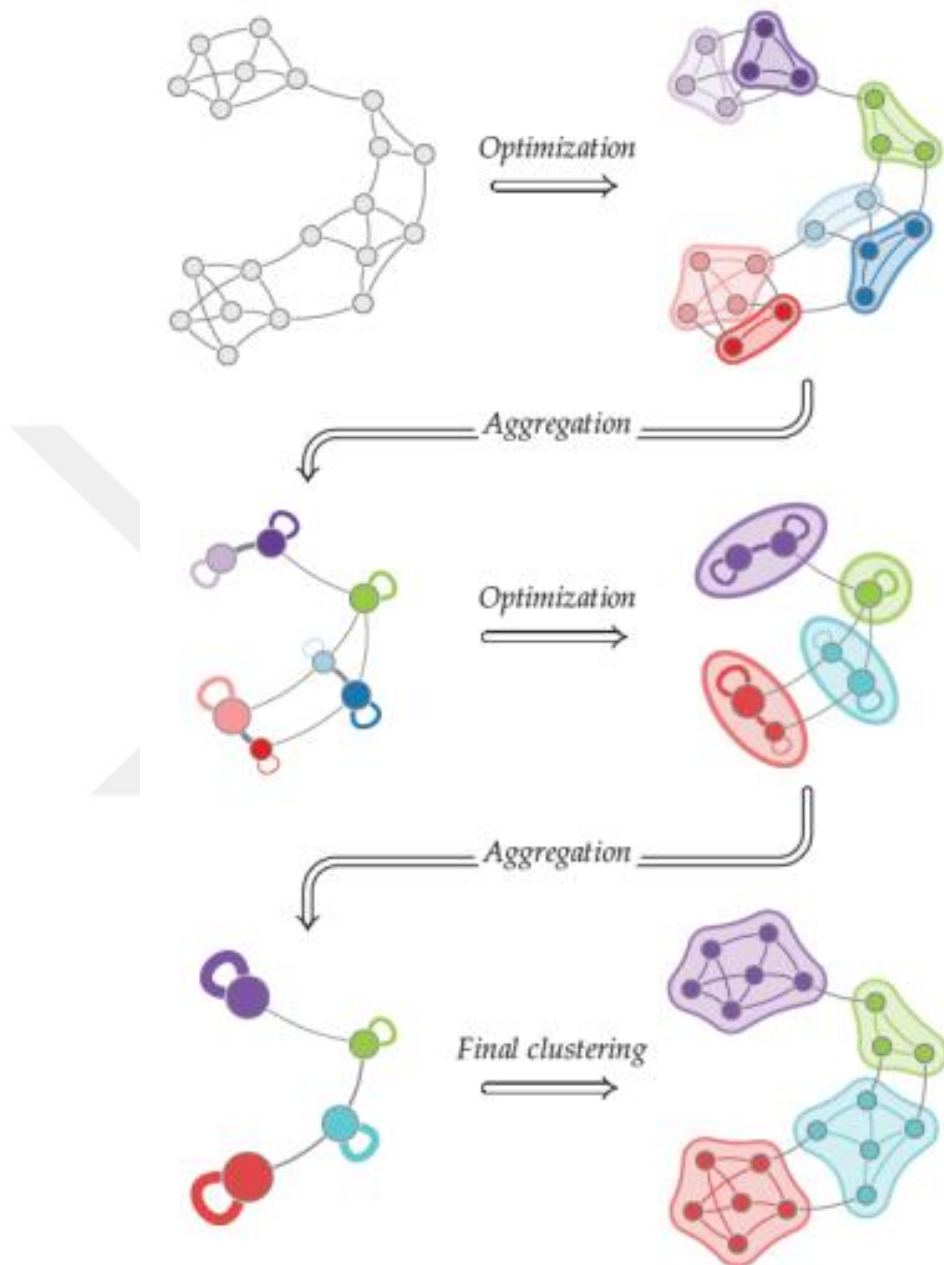


Figure 2.4. An abstract illustration two main phases of Louvain algorithm (Source: Browet 2014)

to obtain an improved community structure. The local moving heuristic depends on repeatedly moving individual nodes from one community to another neighboring community, so that each node move leads to an increase in modularity.

Louvain is an iterative algorithm and contains two main phases in each iteration: Modularity Optimization with Local Moving Heuristics and Community Aggregation. In the first phase, it starts by considering each node in the network as a community. Then, the local shift heuristic is used to obtain an improved community structure by moving individual nodes from one community to another neighboring community until no further increase in modularity can be achieved. In the second phase, Louvain merges the all nodes that belong to the same community. Then, a network is built in which the nodes are the communities from the previous phase. The iterations continue until there is only one community. This process is illustrated in Figure 2.4.

2.3. Dynamic Networks

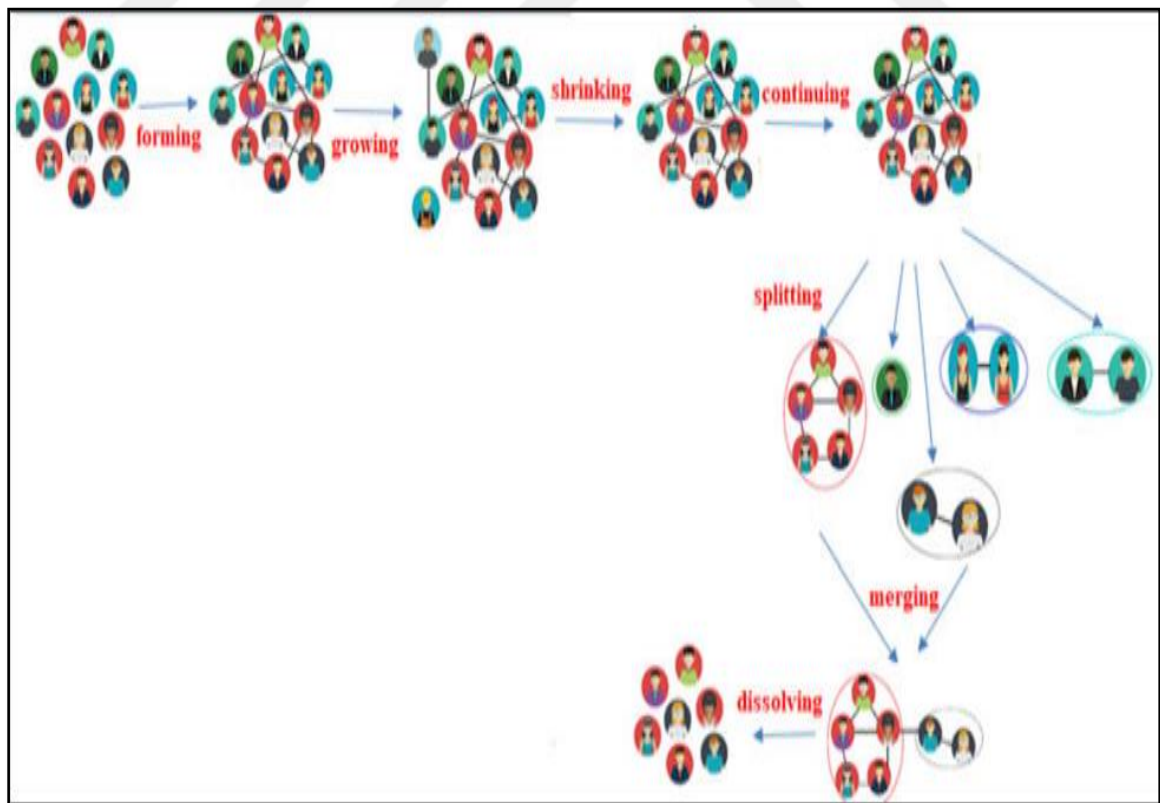


Figure 2.5. A sample scenario for possible community events

Traditional methods for analyzing networks model the network as a static graph that represents a frozen snapshot of the network at a particular point in time. However, this modelling does not capture the evolution of the network. Therefore, dynamic networks are modelled with dynamic graphs. A dynamic graph can be represented as a series of static graphs ordered over time, with each graph corresponding to a particular point in time.

Because relationships or interactions change over time, real-world networks are dynamic. Communities in the network may grow or shrink, new communities may even emerge, while some of them may disappear. An example scenario of possible changes in communities can be seen in Figure 2.5. At the beginning, there are some people without any connection. Then they **form** a community. Over time, more members join the community; thus, the community **grows**. Maybe some of the members lose their interest in the community and they leave it; thus, the community **shrinks**. Furthermore, communities can **continue** their lives either unchanged or unchanged within an upper limit. As the time passes, the community may **split** into some subcommunities or some communities can form a new and larger community by **merging**. Finally, a community may **dissolve** over time by losing its members.

Dynamic networks are important for many scenarios. First, when studying the spread of information, rumors, diseases, or changes in the network, dynamic networks can provide accurate estimates of changes/spreads. Second, when it comes to predicting patterns of change, dynamic networks can help in detecting the patterns, e.g., seasonal changes, etc. These patterns can also be used to predict future changes. Finally, dynamic networks are better suited to represent dense interactions between members/communities.

CHAPTER 3

TRACKING EVOLUTION OF COMMUNITIES IN DYNAMIC NETWORKS

In this chapter, a novel space-efficient method TREC (Tracking Evolution of Communities) is introduced for tracking the evolution of communities in dynamic networks. The main idea behind TREC method for space reduction is simply focus on community matching phase. Similarity preserving community signatures are created by minhashing technique instead of using real communities and LSH (Locality Sensitive Hashing) is used to identify possibly similar communities.

3.1. Background and Problem Formulation

First, key terms and definitions are given in Section 3.1.1 and then the problem is defined in Section 3.1.2 to clarify the methodology of TREC (Tracking Evolution of Communities) for the readers.

3.1.1. Concepts and Definitions

This study is concerned with dynamic networks represented by graphs. Note that the terms networks and graphs are used interchangeably in this manuscript. These dynamic networks/graphs contain community structure unless they are random networks. The community structure can be overlapping or disjoint (nonoverlapping) for both dynamic and static networks. Community detection can be viewed as a solution to the graph partitioning problem. In the simplest case, the evolution of communities is a time-

ordered sequence of the same community. To track them, communities are first detected at each time step and then they are matched according to the similarity of their members to create the time-ordered sequence.

Jaccard Similarity (JS) is one of the most common measures for calculating the member similarity of a pair of communities. It takes real values in the range [0,1], where 0 means that the communities are completely different, and 1 means that they are completely equal with respect to their members. It is suitable for many applications, such as textual similarity of documents and similarity of customers' buying habits. It is calculated by equation (3.1):

$$JS(C_{t_1}^i, C_{t_2}^j) = \frac{C_{t_1}^i \cap C_{t_2}^j}{C_{t_1}^i \cup C_{t_2}^j} \quad (3.1)$$

where $C_{t_1}^i$ and $C_{t_2}^j$ are the compared communities, (i, j) are the numbers to identify the communities, and (t_1, t_2) are the current time steps of these communities. For a pair of communities to be considered similar, they must meet or exceed a similarity threshold (λ). In our study, if $JS(C_{t_1}^i, C_{t_2}^j) \geq \lambda$, then this pair of communities ($C_{t_1}^i$ and $C_{t_2}^j$) is accepted as similar.

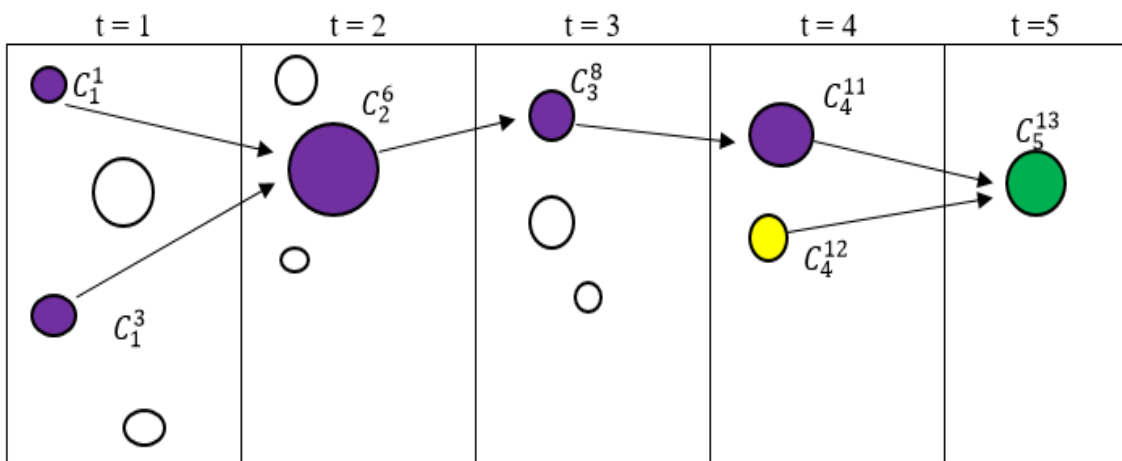


Figure 3.1. A simple illustration of the evolution of communities

Figure 3.1. shows a simple illustration of the evolution of a community, with the purple circles representing the communities in the evolution chain and the white circles representing the other communities and the arrows showing the evolved version of a community that just preceded it. The evolution of a community is represented by the sequence of matching communities over time steps. For example, the evolution of a community C_1^1 from time $t = 1$ to $t = 5$ can be represented as $C_1^1 = \{C_1^1, C_2^6, C_3^8, C_4^{11}, C_5^{13}\}$, with superscript and subscript labels indicating community identification numbers and time steps, respectively. Thus, the community tracking problem is defined as recognizing a set of similar communities between different specific time steps and tracking their evolutionary behavior over the lifetime of a dynamic network.

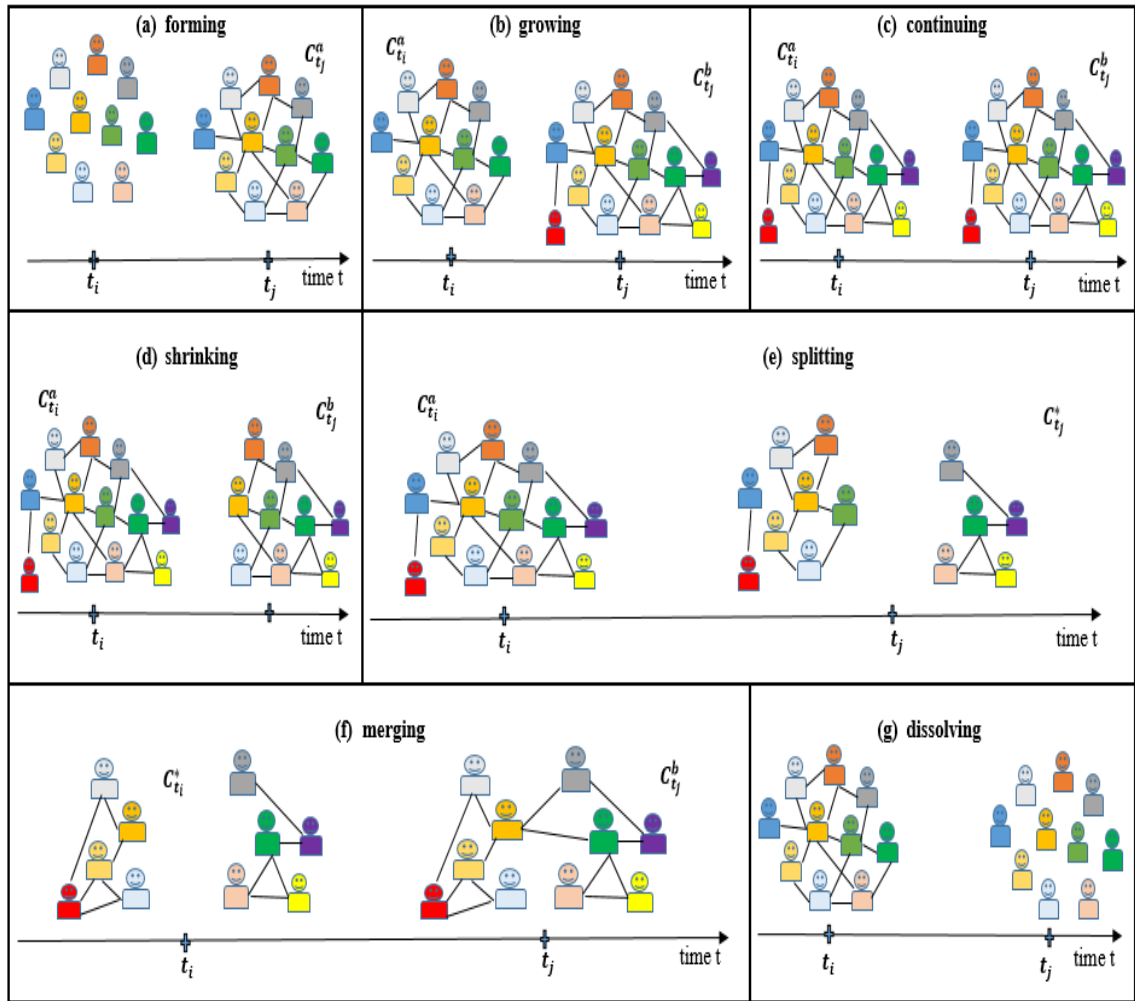


Figure 3.2. An illustration of community evolution events where i and j represent time steps where $i < j$

Table 3.1. Definition and illustration of community evolution events

Definitions of community evolution event types	Reference To Figure 3.2.
Form: A new community C_{t_i} forms at time t_i. $form(C_{t_i}^a) = true$; if $JS(C_{t_i}^*, C_j^a) < \lambda$ for $\forall C_{t_i}^* \in G$ at time t_i and $t_i < t_j$ and $V_{C_{t_i}^a} \geq 3$.	(a)
Grow: New members may join the community or some existing members may move between communities in graph G over time; hence, some of the communities may grow. $growth(C_{t_i}^a) = true$; if $\exists C_{t_j}^* \in G$ at time t_j and $t_i < t_j$ and $JS(C_{t_i}^a, C_{t_j}^b) \geq \lambda$ and $1.05 V_{C_{t_i}^a} \leq V_{C_{t_j}^b}$	(b)
Continue: Communities can continue their lives either without any change or with changes within tiny upper or lower limits as 0.05 rate of change in community size. $continue(C_{t_i}^a) = true$; if $\exists C_{t_j}^b \in G$ at time t_j and $t_i < t_j$, and $JS(C_{t_i}^a, C_{t_j}^b) \geq \lambda$ and $0.95 V_{C_{t_i}^a} < V_{C_{t_j}^b} < 1.05 \times V_{C_{t_i}^a}$.	(c)
Shrink: A portion of the members may leave a community and cause it to shrink. $shrink(C_{t_i}^a) = true$;	(d)

(cont. on next page)

Table 3.1 (cont.)

$if \exists C_{t_j}^b \in G \text{ at time } t_j \text{ and } t_i < t_j, \text{ and } JS(C_{t_i}^a, C_{t_j}^b) \geq \lambda, \text{ and } V_{C_{t_j}^b} \leq 0.95 V_{C_{t_i}^a} .$	
Split: A community can split into subcommunities if the similarity threshold λ between the community and the set of subcommunities is satisfied at the following time step. $split(C_{t_i}^a) = true;$ $if \exists S_{C_{t_j}^*} = \{C_{t_j}^1, C_{t_j}^2, \dots, C_{t_j}^m\} \text{ for } C_{t_i}^a \text{ at time } t_i \text{ and } t_i < t_j, \text{ and } \forall C_{t_j}^* \in S_{C_{t_j}^*}, JS(C_{t_i}^a, C_{t_j}^*) \geq \lambda.$	(e)
Merge: Communities can form a new and larger community by merging. A set of communities $S_{C_{t_i}} = \{C_{t_i}^1, C_{t_i}^2, C_{t_i}^3, \dots, C_{t_i}^n\}$ merge, and form a community $\exists C_{t_j}^b \in G \text{ at time } t_j > t_i$. The similarity threshold λ is exceeded by each community of $S_{C_{t_i}}$ and $C_{t_j}^b$. $merge(S_{C_{t_i}^a}) = true;$ $if \exists S_{C_{t_i}} = \{C_{t_i}^1, C_{t_i}^2, C_{t_i}^3, \dots, C_{t_i}^n\} \text{ at time } t_i < t_j \text{ and } \exists C_{t_j}^b \in G, \forall C_{t_i}^* \in S_{C_{t_i}}, JS(C_{t_i}^*, C_{t_j}^b) \geq \lambda.$	(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 λ in the following steps. $dissolve(C_{t_i}^a) = true; \text{ if } \nexists C_{t_j}^* \in G \text{ at time } t_j > t_i \text{ with } JS(C_{t_i}^a, C_{t_j}^*) \geq \lambda \text{ and } V_{C_{t_i}^a} < 3$	(g)

Relationships/interactions between communities may change over time, so a community may experience some critical events. They are briefly mentioned in section 2.3. All critical events are defined in Table 3.1 and the corresponding visualizations are shown in Figure 3.2.

The evolution of a community may be observed either in consecutive or in nonconsecutive phases. If the evolution of communities is observed at each time step as in Figure 3.3, community C_1^4 evolves continuously. The evolved version of the community from the previous time step is shown at each consecutive time step.

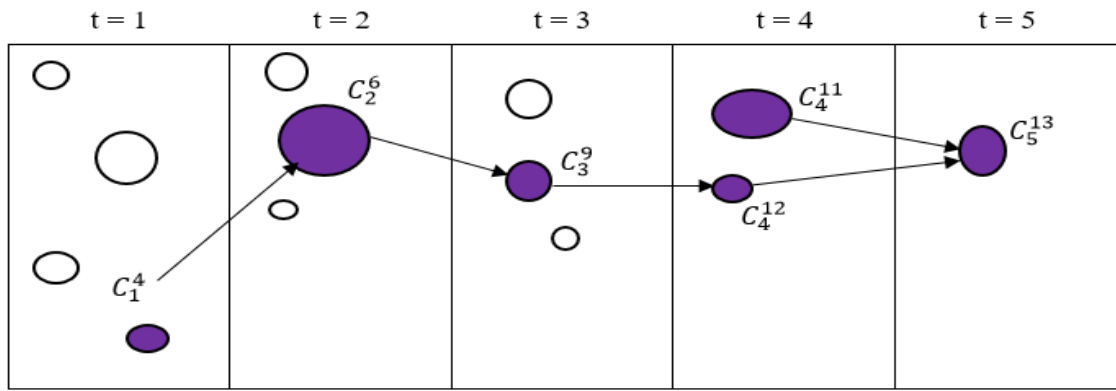


Figure 3.3. A simple illustration of the consecutively evolving communities

However, some communities may not be observed immediately after a particular time step, that is, a community cannot dissolve only after a step. This type of evolution of communities is called nonconsecutive evolution. A simple illustration is shown in Figure 3.4. for nonconsecutive evolving communities C_1^1 and C_1^3 . As can be seen from the figure, the evolution of these communities is not observed at time steps $t = 3$ and $t = 5$.

The number of communities in an evolution chain indicates the chain length, and the chain length is indicated by the character “L”. For example, the evolution chain of the consecutively evolving community C_1^4 shown in Figure 3.3 is $C_1^4 = \{C_1^4, C_2^6, C_3^9, C_4^{11}, C_5^{13}\}$ and $L(C_1^4) = 5$. As another example, the evolution chain of nonconsecutively evolving community C_1^4 is $C_1^3 = \{C_1^3, C_2^6, C_4^{11}\}$ and $L(C_1^4) = 3$.

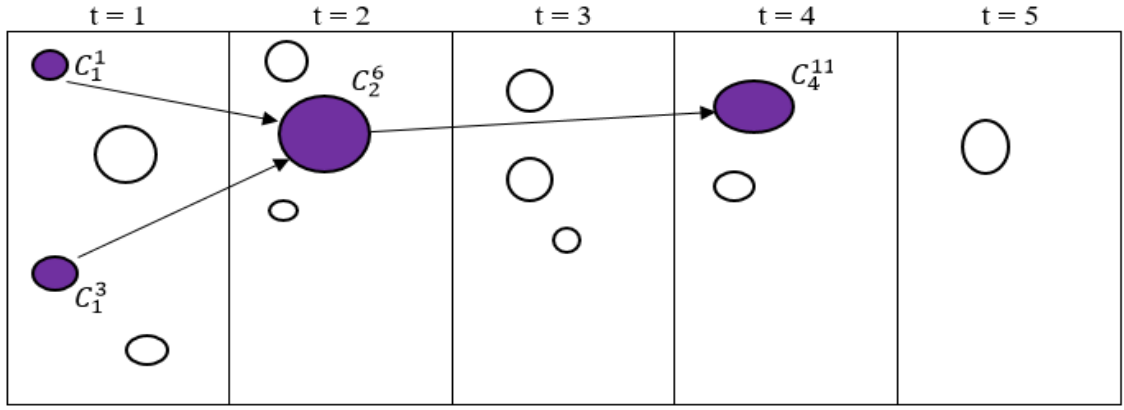


Figure 3.4. A simple illustration of the nonconsecutively evolving communities

3.1.2. Problem Formulation

Let $G_t = (V_t, E_t)$ be a graph representing a static network, where V_t is the set of vertices (i.e., nodes) and E_t is the set of edges at a given time step t . A dynamic network G can be denoted as a sequence of static networks such as $G = \{G_1, G_2, \dots, G_{timeStepCount}\}$, where $t = 1, 2, \dots, timeStepCount$. A community is a subset of densely connected vertices of each time graph G_t , while it is loosely connected to the rest of G_t . There can 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 (i.e., communities) such that $C_t = \{C_t^1, C_t^2, \dots, C_t^k\}$, where each community $C_t^i \in C_t$, $i = 1, \dots, k$ with vertex set and edge set of each community $C_t^i = (V_t^i, E_t^i)$ as $C_t^i \subseteq G_t$.

3.1.3. Evaluation Criteria for Methods for Tracking Evolution of Communities

There are many different ways to track the evolution of communities. But there are some key criteria to characterize them and with this characterization it is possible to compare and evaluate the methods. These criteria are explained below:

Criterion #1 (Community Structure): As mentioned in Chapter 2, the community structure is either disjoint (nonoverlapping) or overlapping, depending on the type of network. For example, members of an OSN (Online Social Networks; such as Facebook, Twitter) may be members of many communities. In protein-protein interaction networks, on the other hand, groups of proteins that have the same specific function within the cell belong to a particular community. Therefore, the underlying community structure in the network influences the community detection algorithm used and the method used to track the evolution of communities.

Criterion #2 (Ability of the community type to evolve): As mentioned in Section 3.1.1, some communities can evolve consecutively, while others evolve nonconsecutively in the same network. All existing methods for tracking community evolution already track consecutively evolving communities. However, some communities in real dynamic networks do not evolve consecutively. Therefore, to obtain more realistic results, an appropriate community evolution tracking method must be able to track both types of evolution.

Criterion #3 (Coverability): Evolutionary events such as formation, growth, etc. are presented in Section 3.1.1. An appropriate method for tracking the evolution of communities must cover all possible evolutionary events.

Criterion #4 (Recognise the ability to merge/split k communities): In real-world networks, a community can be split into k -subcommunities and/or k -subcommunities can be merged into one community, where $k \geq 2$. An appropriate method for tracking community evolution must support the detection of k -community merging and k -community splits.

3.1.4. Minhashing

Communities of members in real-world networks, such as social networks, are generally large. If the network includes hundreds of thousands or millions of community members, it may not be possible to store all the members that make up the communities in main memory. Even if all members fit in the main memory, the number of pairs may

be too large to evaluate JS (Jaccard Similarity) of each community pair. Therefore, there is a need to replace large communities with much smaller representations.

One of the solutions is hashing to convert each community into a small signature, using a hashing function h . The function h must have the following properties, where $t1$ and $t2$ are time steps, x and y are the identification numbers of the communities.

- $h(C_{t1}^x)$ is the signature of community C_{t1}^x and it occupies less space in main memory than the members of the whole community. In this way, the signatures of all communities can be accommodated in main memory. If $\text{similarity}(C_{t1}^x, C_{t2}^y)$ is high, then $\text{Probability}(h(C_{t1}^x) == h(C_{t2}^y))$ is high.
- If $\text{similarity}(C_{t1}^x, C_{t2}^y)$ is low, then $\text{Probability}(h(C_{t1}^x) == h(C_{t2}^y))$ is low.

The choice of hashing function is closely related to the similarity metric used. Minhashing is the appropriate hashing function for computing JS (Leskovec, Rajaraman, and Ullman 2015). Minhashing is a technique introduced by Andrei Broder (Broder 1997) to find out duplicate pages on Alta Vista. It compresses large data sets into fixed-length sketches called "signatures". A minhashing function depends on the permutation of sets/groups/communities. The basic idea of minhashing is to hash each set into a small fixed length *signature* $\text{sig}(\text{set})$ such that:

- $\text{sig}(\text{set})$ is small enough to fit the signature in main memory.
- $\text{sim}(\text{set1}, \text{set2})$ is same as $\text{sim}(\text{sig}(\text{set1}), \text{sig}(\text{set2}))$

1. Compute $h_1(\text{row\#}), h_2(\text{row\#}), \dots, h_n(\text{row\#})$.
2. Initialize all entries of $\text{sig}[]$ with infinity
3. For each column col
 - 3.1 If col has 1 in the current row ,
 - 3.2 For each $i = 1, 2, \dots, n$
 - 3.3 $\text{sig}[i, col] = \min\{\text{sig}[i, col], h_i(r)\}$

Figure 3.5. Main steps of process of how to compute minhash signatures

It is not computationally feasible to permute randomly a large adjacency matrix explicitly. Thus, there is need to mimic random permutation. Therefore, instead of picking n random permutations, n random hash functions from h_1 to h_n are picked. Then a signature matrix is constructed and its construction steps in Figure 3.5 are described below. (Leskovec, Rajamaran and Ullman 2015).

Example: Computing minhash signatures

Member ID	Set ₁	Set ₂	Set ₃	Set ₄
1	1	0	0	1
2	0	0	1	0
3	0	1	0	1
4	1	0	1	1
5	0	0	1	0

Figure 3.6. An adjacency matrix, A , representing sample sets (e.g., groups or communities) and their respective members

Just for completeness, let us compute a signature matrix for the adjacency matrix in Figure 3.6. as an example, where the sets are the groups (or communities) and the member IDs are the identification numbers of the members of those groups. Suppose there are two hash functions such as $h_1(x) = x+1 \pmod{5}$ and $h_2(x) = 3x+1 \pmod{5}$. x in the hash functions refers to the row numbers, so the permutations are effectively simulated. The entries of the adjacency matrix, A , are binary. The entry "1" represents the presence of a member, while the entry "0" represents the absence of a member. In Figure 3.6, $\text{Set}_1 = \{1,4\}$, $\text{Set}_2 = \{3\}$, $\text{Set}_3 = \{2,4,5\}$ and $\text{Set}_4 = \{1,3,4\}$. Note that these simple functions generate real permutations because the number of rows in A , 5, is a prime number. That is, the minhash functions, $h_1, \dots, h_n$ implicitly rearrange the rows of the matrix of the figure.

In the first step, the values $h_1(\text{row\#})$, $h_2(\text{row\#})$, $\dots$, $h_n(\text{row\#})$ are calculated. The values obtained can be seen in Figure 3.7.

Row #	Member ID	Set ₁	Set ₂	Set ₃	Set ₄	h_1	h_2
						$x+1 \pmod{5}$ $(\text{row\#})+1 \pmod{5}$	$3x+1 \pmod{5}$ $3(\text{row\#})+1 \pmod{5}$
1	1	1	0	0	1	1	1
2	2	0	0	1	0	2	4
3	3	0	1	0	1	3	2
4	4	1	0	1	1	4	0
5	5	0	0	1	0	0	3

Figure 3.7. The computed hash functions for the matrix of Figure 3.6.

In the second step, all entries of $sig[]$ matrix are initialized to infinity. Figure 3.8. shows the initialized signature matrix, $sig[]$.

	Set ₁	Set ₂	Set ₃	Set ₄
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞

Figure 3.8. Initialized signature matrix, $sig[]$

In the third step, entry '1's are scanned by $A []$. The minimum values generated by the hash functions are compared with the corresponding cell of $sig[]$ and the minimum of them is selected and assigned to the cell. For Set₁, the minimum value of h_1 is 1 and $sig[1, 1]$ is ∞ . Therefore, $sig[1,1]$ is assigned as 1. For the same set, the minimum value of h_2 is 0 and $sig[1,2]$ is ∞ . Since 0 is less than ∞ , $sig[1,2]$ is assigned 0.

Row #	Member ID	Set ₁	Set ₂	Set ₃	Set ₄	h_1	h_2
						$x+1 \pmod{5}$ $(\text{row\#})+1 \pmod{5}$	$3x+1 \pmod{5}$ $3(\text{row\#})+1 \pmod{5}$
1	1	1	0	0	1	1	1
2	2	0	0	1	0	2	4
3	3	0	1	0	1	3	2
4	4	1	0	1	1	4	0
5	5	0	0	1	0	0	3

Altered version of sig [] is below.

	Set ₁	Set ₂	Set ₃	Set ₄
h_1	1	∞	∞	∞
h_2	0	∞	∞	∞

After the computation of the minhash values for Set₁ is complete, the minimum hash values for Set₂ are determined.

As can be seen, the minimum values are 3 and 2 generated by h_1 and h_2 , respectively. Since the corresponding cells of the signature matrix span ∞ and are all less than ∞ , the minimum hash values are 3 and 2.

Row #	Member ID	Set ₁	Set ₂	Set ₃	Set ₄	h_1	h_2
						$x+1 \pmod{5}$ $(\text{row\#})+1 \pmod{5}$	$3x+1 \pmod{5}$ $3(\text{row\#})+1 \pmod{5}$
1	1	1	0	0	1	1	1
2	2	0	0	1	0	2	4
3	3	0	1	0	1	3	2
4	4	1	0	1	1	4	0
5	5	0	0	1	0	0	3

The updated version of the signature matrix is below.

	Set ₁	Set ₂	Set ₃	Set ₄
h_1	1	3	∞	∞
h_2	0	2	∞	∞

As for Set3, the minimum hash values are 0 generated by both h_1 and h_2 . Since the corresponding cells of the signature matrix contain ∞ and 0 is less than ∞ , the minimal hash values are 0s.

Row #	Member ID	Set ₁	Set ₂	Set ₃	Set ₄	h_1	h_2
						$x+1 \pmod{5}$ $(\text{row\#})+1 \pmod{5}$	$3x+1 \pmod{5}$ $3(\text{row\#})+1 \pmod{5}$
1	1	1	0	0	1	1	1
2	2	0	0	1	0	2	4
3	3	0	1	0	1	3	2
4	4	1	0	1	1	4	0
5	5	0	0	1	0	0	3

The updated version of the signature matrix is below.

	Set ₁	Set ₂	Set ₃	Set ₄
h_1	1	3	0	∞
h_2	0	2	0	∞

Lastly, minimum hash values are determined for Set₄.

Row #	Member ID	Set ₁	Set ₂	Set ₃	Set ₄	h_1	h_2
						$x+1 \pmod{5}$ $(\text{row\#})+1 \pmod{5}$	$3x+1 \pmod{5}$ $3(\text{row\#})+1 \pmod{5}$
1	1	1	0	0	1	1	1
2	2	0	0	1	0	2	4
3	3	0	1	0	1	3	2
4	4	1	0	1	1	4	0
5	5	0	0	1	0	0	3

As can be seen from the above figure, the minimum values are 1 and 0 generated by h_1 and h_2 , respectively. Since the corresponding cells of the signature matrix contain ∞ and are all less than ∞ , the minimum hash values are 1 and 0. The latest version of the signature matrix can be found below.

	Set ₁	Set ₂	Set ₃	Set ₄
h ₁	1	3	0	1
h ₂	0	2	0	0

The Relation Between Minhash Signatures and Jaccard Similarity:

There is a relationship between the minhashes of a pair of sets (e.g. groups, communities) and the Jaccard similarities (JS) between them. This relationship states that the probability of similarity between the minhash signatures of a pair of sets is equal to the JS of the pair of sets. To find out why this is so, we need to examine the row types of a pair of sets in terms of the members they contain. Then, the rows are divided into three classes, e.g., X (both contain the member), Y (one of them contains the member), and Z (none contains the member), as shown in the following table. Since only the number of rows of type X and type Y determine both JS and the probability of minhash signatures of a pair of sets is equal, the rows whose type is Z are negligible.

Row Type	Set ₁	Set ₂
X	1	1
Y	1	0
Y	0	1
Z	0	0

Suppose there are x rows of type X and y rows of type Y in the minhash signatures of the set pair. Then, $JS(Set_1, Set_2) = \frac{x}{(x+y)}$.

Let us focus on the probability that the minhash signatures of a pair of sets are equal. Since it is necessary to randomly permute the rows of the membership list/matrix of sets, the probability that we hit a row of type X before we hit a row of type Y is $\frac{x}{(x+y)}$. As can be seen, $JS(Set_1, Set_2)$ has the same relationship with $P(h(Set_1), h(Set_2))$.

In Table 3.2, we compute the Jaccard similarity values of the adjacency matrix A from Figure 3.6 and the Jaccard similarity values for the signature matrix ($sig[]$). As can be seen from the table, the Jaccard similarities are preserved. That is, there is no similarity between Set₁ and Set₂, and Set₂ and Set₃ in the A matrix, the case is valid for the signature

matrix. Other values are only an approximation. The more hash functions there are, the better approximations are obtained.

Table 3.2. Jaccard similarity for original matrix and approximate Jaccard Similarity for signature matrix sig[]

	Set ₁ -Set ₂	Set ₁ -Set ₃	Set ₁ -Set ₄	Set ₂ -Set ₃	Set ₂ -Set ₄	Set ₃ -Set ₄
Similarity over A[]	0	0.2	0.4	0	0.2	0.2
Similarity over sig[]	0	0.5	1	0	0	0.5

One may be concerned with the difference between exact similarities and approximate similarities calculated via minhash signatures. For example, the approximate similarity over their signatures is 50%, while the actual similarity between Set3 and Set4 is 20%. On the one hand, the minhash procedure guarantees that if there is no similarity between two sets, it provides a similarity value of zero for them. On the other hand, as the number of hash functions increases, the approximation to exact similarity values gets better and better. Incidentally, this problem is completely solved by Line 14 of the pseudocode in Figure 3.18. In this line, the exact similarities of the communities are checked externally by $JS(C_{tx}^k, C_{tc}^c)$. Therefore, there is no need to worry about using approximate similarities. They are only used to quickly find similar communities by LSH.

3.1.5. Locality Sensitive Hashing for Minhash Signatures

To determine the evolution of communities, one must know the similarity of pairs of communities. Suppose that one wants to calculate the exact similarity of pairs of communities in order to match them. This calculation needs $(t - 1) \times c^2$ comparison operations, where c is the number of average communities for each time step and t is the number of time steps in the dynamic network. If someone wants to know the exact similarity of each pair, there is nothing to reduce the work, but parallelism can help reduce

the time spent. Minhashing dramatically improves the comparison times between two communities. Note that the members of the community pairs must be compared if minhashing is not used. However, finding similar communities is still expensive. In fact, it is enough to know that the similarity between a community pair exceeds a lower bound (LSH_{ST}) to decide whether a community pair is similar. Therefore, focusing only on community pairs that are likely to be similar in the community matching phase is sufficient to track the evolution of communities without examining the similarity of individual pairs. LSH, short for Locality Sensitive Hashing, is an efficient technique to find out approximate near neighbors. Combining LSH with minhash signatures is so elegant for queries in Jaccard space.

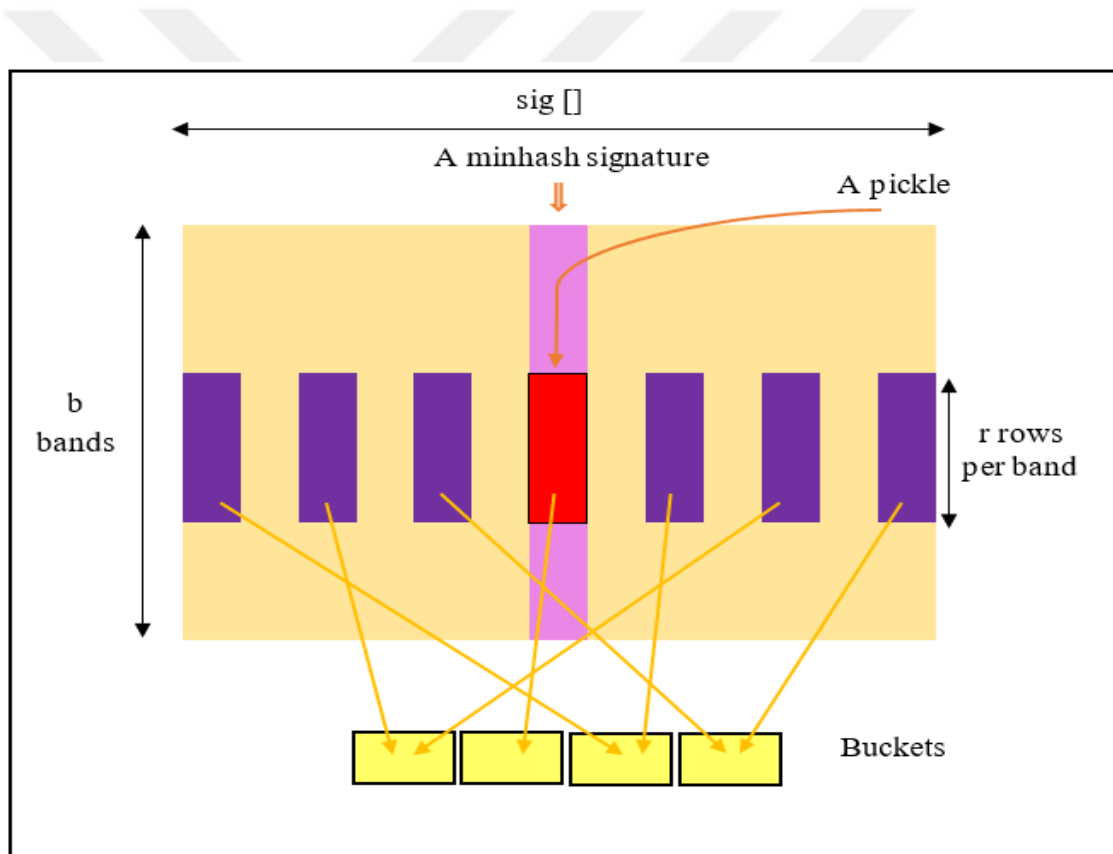


Figure 3.9. Illustration of how LSH works

Figure 3.9. shows the representation of the LSH table and buckets, and how the LSH technique works. A signature matrix is formed from the minhash signatures of each

community. Therefore, each column in the signature matrix $\text{sig}[]$ is a **signature** of a particular community. The matrix $\text{sig}[]$ is divided into b bands, where each band contains the same number of r rows. We denote the number of rows in the signature matrix as n , where $n = b \times r$ and n is determined by the number of hash functions used. A part of a minhash signature whose length is r within a band is called a "**pickle**". A general approach to the LSH technique is to hash minhash signatures such that similar signatures are more likely to be hashed into the same bucket than dissimilar ones. Note that the same hash function can be used for all bands, but it is necessary to use separate bucket collections for each band.

To hash likely similar signatures into the same bucket than dissimilar ones, 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. When LSH is queried for near neighbors of a 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.

LSH_{ST} (similarity threshold for LSH) is determined according to equation (3.2) as seen from the figure above. LSH guarantees that communities that are either equal to or greater than LSH_{ST} are returned with a certain probability value. Its calculation is explained below.

$$LSH_{ST} = \left(\frac{1}{b}\right)^{(1/r)} \quad (3.2)$$

The probability that a community pair will become candidates that have a percent λ similarity is calculated using equation (3.3).

$$1 - (1 - \lambda^r)^b \quad (3.3)$$

The following steps show how to calculate equation (3.3). Assume that Community 1, C_1 , and Community, C_2 , and λ is the similarity threshold for LSH.

Step 1: The Probability that C_1 and C_2 are identical in a particular band is λ^r .

Step 2: The Probability that C_1 and C_2 are not similar in a particular band is $1 - (\lambda^r)$.

Step 3: The Probability that C_1 and C_2 are not similar in any of the bands is $(1 - (\lambda^r))^b$.

Step 4: The Probability that C_1 and C_2 are identical in at least one band is $1 - (1 - (\lambda^r))^b$. That is, LSH consider C_1 and C_2 as a similar pair with this probability.

In this thesis, λ first set to 0.1 between community signatures. Then, LSH_{ST} is determined as close as possible to λ . Later, the values of b and r are chosen experimentally to obtain the desired LSH_{ST} . By substituting these values into equation (3.3), the probabilities of the signature pairs discovered by LSH are calculated and presented in Table 3.3.

Table 3.3. Probabilities of the signature pairs detected by LSH with respect to the λ

Similarity between a minhash signature pair, λ	Probability of the pair have λ similarity detected by LSH, $1 - (1 - (\lambda^r))^b$
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

The results in Table 3.3 show us that LSH recognizes the signature pairs that have 10% similarity with 36% probability. Also, LSH recognizes the pairs with 20% similarity

with a probability of 84%. Moreover, it recognizes the pairs that have 40% similarity or more with 100% probability. As can be seen, as the similarity of the pairs increases, the recognition performance of LSH also increases. These results are quite satisfactory, especially with respect to the trade-off between time and space requirements in exact searching for similar communities of a given community.

The reason for choosing λ as 0.1 is to determine the candidate community pairs that maximize the community relationships, including the community pairs that have a similarity of 10% with 36% probability in the analyzed dataset, over the LSH data structure. Moreover, our tracking process verifies the exact JS between these candidates with respect to their members. If the value of JS of the candidate pairs exceeds the λ -value, then they are matched. The analysis of the event types over the time steps is done after the completion of our tracking process.

The application of “LSH for minhash signatures” in this study is explained in Section 3.3 Methodology.

3.2. Related Work

Since understanding and explaining group-level dynamic networks are important for many research areas, tracking community evolution has emerged and continues to grow. In recent years, several studies have been published on this topic. To provide an up-to-date overview of the topic, we review the studies published in recent years. They are summarized below.

Cazabet et al. (Cazabet, Rosetti, and Amblard 2017) present a distilled information on dynamic community detection, operations on communities (i.e., community events-growth, contraction, merge, splitting, birth, death, and resurgence), a classification of approaches, and give references of famous works in each class with their respective advantages and disadvantages. They classified the existing approaches into three groups according to the definition of what good dynamic communities are: *Instant Optimal* (independent detection of communities in each snapshot and matching communities), *Temporal Trade-off* (initial detection of communities in the first snapshot and sequential detection of communities in the next snapshot using the current network

and previous communities), and *Cross-time* detection of communities (a coupling graph is created from all snapshots and community detection is done in this graph).

Saganowski et al. (Saganowski, Bródka, and Kazienko 2017) describe and review methods for tracking community evolution. These methods are designed for different types of social networks, such as disjoint, overlapping, or both. They provide basic ideas about all the methods they addressed in the paper. They also describe in detail the method of Asur et al. (Asur, Parthasarathy, and Ucar 2009), the method of Palla et al. (Palla, Derényi, Farkas, and Vicsek 2005), and Group Evolution Discovery (GED) (Brodka, Saganowski, Kaizenko 2012).

Rossetti and Cazabet (Rozetti and Cazabet 2018) introduce network models (i.e., temporal networks and network snapshots) and extend their classification in (Cazabet, Rosetti, and Amblard 2017) as a two-level one. The higher level categorization approaches are the same. At the second level (i.e., subcategorization), they further classify according to their community detection techniques, such as iterative similarity-based approaches, updating by global optimization, updating by set of rules, etc. Next, they discuss issues related to community evaluation, such as benchmarks, synthetic generators, and in the case of the existence of ground truth communities. They summarize the application areas of dynamic community detection for real-world problems and visualization.

A novel taxonomy introduced by Dakiche et al. (Dakiche, Tayeb, Slimani, and Benatchba 2019a) is a three-level classification of existing methods for tracking community evolution in dynamic social networks in terms of their network models (first level of tree structure), operating principles (second level), and algorithmic techniques (third level). This taxonomy is graphically represented in Figure 3.10. The figure shows four basic approaches according to their operating principles:

- (i) The independent community detection and matching approach includes methods that first detect the structure of communities at each time step separately and match these communities across consecutive or nonconsecutive time steps. All methods in this category aim to track the evolution of communities by identifying key community events (e.g., form, dissolve, growth, etc.) across community life cycles. Core-based methods identify one or more specific nodes for each community, called core nodes. For example, the nodes with the highest centrality value in the

network may be core nodes. Then, the methods determine community events that correspond to the core nodes. In the event-based methods, all nodes are considered to determine the community events.

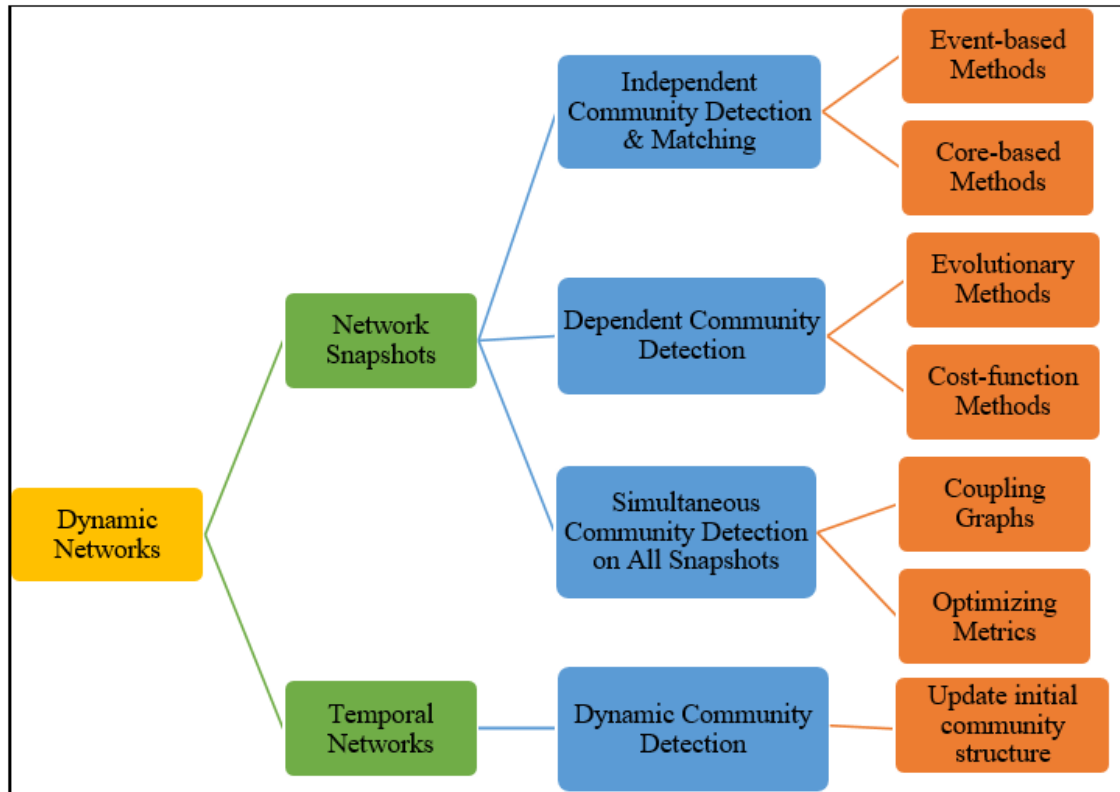


Figure 3.10. Classification chart of the existing tracking community evolution methods in dynamic social networks

- (ii) The dependent community detection approach includes methods that detect the community structure using the snapshot at time t and past community information (e.g., using the previous snapshot or some recent snapshots). In this approach, there are two main types of methods: evolutionary methods and cost function methods. Evolutionary methods build on or modify basic community detection algorithms, such as Louvain (Blondel, Jean-Loup, Renaud Lambiotte, and Lefebvre 2008). They initialize the community structure with the communities detected by these algorithms and re-run the modified basic algorithm. For cost function methods, they

use a cost function (e.g., any function to minimize community changes between successive snapshots, such as maximizing modularity between two successive snapshots).

- (iii) The Simultaneous Community Detection on All Snapshots approach includes methods that first construct a single from all snapshots and detect the community structure on the coupling (joint graph). The methods using this approach are either based on coupling graphs or on optimizing metrics. In methods based on coupling graphs, the basic idea is to construct a coupling graph and detect communities on this graph. For methods based on optimizing metrics, the basic idea is to design a metric that can be optimized directly on all given snapshots.
- (iv) Dynamic community detection on temporal networks involves methods that discover communities only on the first snapshot network and then change this community structure for each incoming update.

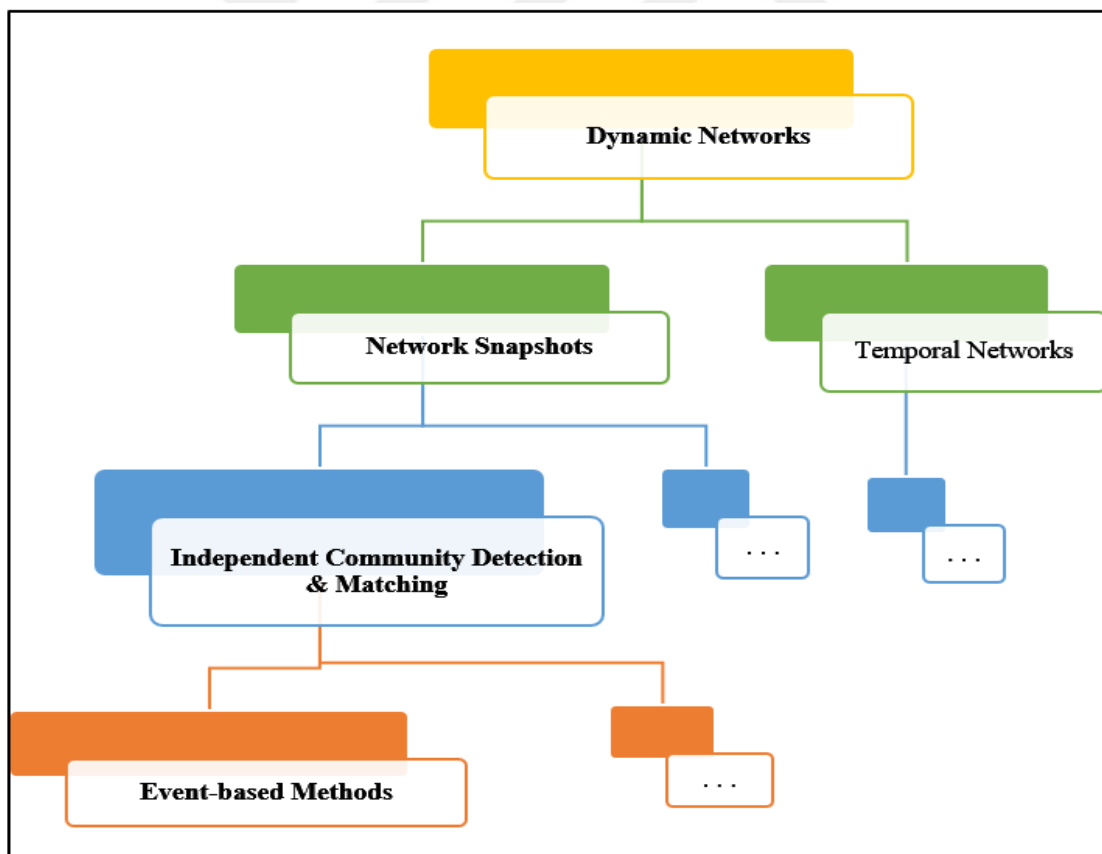


Figure 3.11. An illustration of replacement of TREC method under Event-based Methods in Dakiche et al.'s taxonomy

With the exception of the taxonomy of Dakiche et al. (Dakiche, Tayeb, Slimani, and Benatchba 2019a), the other taxonomies summarized above are very similar and generally complementary. The Dakiche et al.'s taxonomy is the most recent and comprehensive taxonomy specifically for dynamic social networks.

Our novel method TREC can be placed in the category of event-based methods in the taxonomy of Dakiche et al. as shown in Figure 3.11. TREC works with snapshot networks and uses an independent community detection and matching approach. It is an event-based method. Since TREC can be classified into this taxonomy, we explain the taxonomy in detail by describing the work on each category and discussing the advantages and disadvantages of each category in Appendix A.

As competing works, the methods are examined in the literature under the same place in the taxonomy. These works are summarized below.

Asur et al. (Asur, Parthasarathy, and Ucar 2009) proposed a simple method for identifying community events via a matching approach on time-step community membership matrices. They use MCL (Van Dongen 2000), a modularity-based algorithm, to detect community structures.

Greene et al. (Greene, Doyle, and Cunningham 2010) propose a model for community evolution over time. They obtain community structures at each time step as step communities. They represent each dynamic community by a timeline of its constituent step communities. The most recent observation in a timeline is called the front of the dynamic community. After matching communities via Jaccard similarity, the authors update the front elements and timelines for each dynamic community.

Brodka et al (Bródka, Saganowski, and Kazienko 2012) develop the Group Evolution Discovery(GED) method. The core of this method is the inclusion measure, which allows to evaluate the inclusion of one community in another by combining both the group quantity (what number of members of the first group in the second group) and the quality (importance of all members in the group) of community members. Based on the inclusion measure, some rules for determining community events are established.

Gliwa et al. (Gliwa, Saganowski, Zygmunt, Bródka, Kazienko, and Kozak 2012) propose a different method, which they call Stable Group Changes Identification(SCGI). They use the method CPM (Palla, Derényi, Farkas, and Vicsek 2005) for community detection and propose a modified Jaccard similarity instead of an inclusion metric. They assume that two communities are similar if the similarity is greater than 50%.

Takaffoli et al. (Takaffoli, Fagnan, Sangi, and Zaïane 2011) develop a method to improve community evolution tracking by defining rules. They perform matching operations communities not only between consecutive time steps but also between different time steps by using a simple greedy matching algorithm.

Tajeuna et al. (Tajeuna, Bouguessa, and Wang 2016) propose a new method for modeling and tracking community evolution. They first independently identify the community structure at each time step. For each identified community, they find the number of members it has in common with all other communities. Then, for each identified community, they estimate a representative vector that contains members in common with the other communities. Next, they compare the representative vectors with a new similarity measure called mutual transition. Using this measure, they create rules to capture community events.

Mohammadmosaferi and Naderi (Mohammadmosaferi and Naderi 2020) propose a new method ICEM (Identification of Community Evolution by Mapping) for tracking the evolution of communities. ICEM can identify both consecutive and nonconsecutive evolutions of communities. Basically, it determines evolution by tracking the members of communities within a global hash map. It assigns each member to a pair, where time represents the last observed time step and community represents the last observed community. In addition, it records the similarity of each community at a particular time step. Similarity lists and the hash map are used to determine the evolutions.

Table 3.4. summarizes the characteristics of the most popular and recent methods that use the independent community detection and matching approach. The columns of the table indicate the names of the methods presented in related works. The attribute *Year* indicates the publication date of the work. The attribute *CD algorithm* indicates the community detection method used. The attribute *Similarity Metric* represents the similarity measure used by the tracking method to decide the similarity of a pair of communities. Jaccard similarity and modified Jaccard similarity are represented as "J." and "M.J." in Table 3.4. The attributes of the criteria are evaluation criteria, which are explained in subsection 3.1.3. Criterion #1 indicates whether the communities are overlapping or nonoverlapping (disjoint), and they are represented in the table as "O." and "NO.", respectively. Criterion #2 indicates the tracking ability of the methods for either/both consecutive and/or nonconsecutive evolutions. Criterion #3 lists the

undetected evolutionary events. Criterion #4 indicates whether the tracking method is able to detect k-community merge and split events.

As can be seen in Table 3.4, some methods work with nonoverlapping communities (Asur, Parthasarathy, and Ucar 2009; Greene, Doyle, and Cunningham 2010; Takaffoli, Fagnan, Sangi, and Zaïane 2011; Tajeuna, Bouguessa, and Wang 2016; Mohammadmosaferi and Naderi 2020) and the others work on overlapping communities (Bródka, Saganowski, and Kazienko 2013; Gliwa, Saganowski, Zygmunt, Bródka, Kazienko, and Kozak 2012). Markov Cluster Algorithm (MCL) (Van Dongen 2000), Louvain (Blondel, Guillaume, Lambiotte, and Lefebvre 2008), Local Community Mining (Chen, Zaïane, Goebel 2009), Infomap (Rosvall and Bergstrom 2008), and Leiden (Traag, Waltman, and Van Eck 2019) are the most common methods for detecting nonoverlapping community structures, while Clique Percolation Method (CPM) (Palla, Derényi, Farkas, and Vicsek 2005) is the most common method for detecting overlapping community structures.

Table 3.4. Overview of the mainstream and latest competitor methods

	Methods						
Attributes	Asur et al. (Asur, Pathasary, and Ucar 2009)	Greene et al. (Greene, Doyle, and Cunningham, 2010)	Takaffoli et al. (Takaffoli, Fagnan, Sangi, <u>Zaiane</u> , 2011)	GED (Bródka, Saganowski, and Kazienko 2012)	SCGI (Gliwa, et al. 2012)	Tajeuna et al. (Tajeuna, Bouguessa, and Wang 2016)	ICEM (Moham-madmosafer i and Naderi 2020)
Year	2009	2010	2011	2012	2012	2016	2020
CD algorithm	MCL	Louvain	LCM	CPM	CPM	Infomap	Louvain Leiden
Similarity Metric	M.J.	J.	A Specific Measure	Inclusion measure	M.J.	Mutual Transition	A Specific Measure
Criterion #1	NO.	NO.	NO.	O.	O.	NO.	NO.
Criterion #2	C.	C. & limited NC.	C. & NC.	C.	C.	C. & NC.	C. & NC.
Criterion #3	Shrink, expand	Continue	Shrink, expand	None	None	None	None
Criterion #4	No	No	Yes	Yes	Yes	Yes	Yes

It is obvious that all tracking methods detect consecutively evolving communities; however only some methods are capable of tracking nonconsecutively evolving communities (Takaffoli, Fagnan, Sangi and Zaïane 2011; Tajeuna, Bouguessa and Wang 2016; Mohammadmosaferi and Naderi 2020). However, only Greene et al.'s method (Greene, Doyle and Cunningham 2010) is limited to detecting the nonconsecutive evolution of a community for three consecutive snapshots.

For criterion #3 (ability to detect events), the methods GED, SCGI, (Tajeuna, Bouguessa, and Wang 2016), and ICEM can all detect evolutionary events, but (Asur, Parthasarathy, and Ucar 2009; Takaffoli, Fagnan, Sangi, and Zaïane 2011) and (Greene, Doyle, and Cunningham 2010) do not detect shrink/expand and continue events, respectively.

For criterion #4; the methods (Asur, Parthasarathy, and Ucar 2009) and (Greene, Doyle, and Cunningham 2010) cannot track 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 by the methods, the most common metric is Jaccard similarity. Takaffoli et al. (Takaffoli, Fagnan, Sangi, and Zaïane 2011) accept a pair of communities as similar if the proportion of their common members is equal to or greater than k (i.e., predefined similarity threshold) of the largest community. The inclusion measure of GED combines both quantity (i.e., one community includes how many members of another) and quality of community members (i.e., social importance of community members, e.g., degree of centrality, degree betweenness degree, page rank). ICEM calculates similarity based only on the ratio of intersection to size of a community. An appropriate method for tracking the evolution of communities must have some desired functional and performance characteristics below.

Desired Functional Properties:

- (i) It should be able to track both consecutively and nonconsecutively evolving communities.
- (ii) It should be able to detect all evolutionary events.
- (iii) It should be able to track merge and split events for more than two communities.

Desired Performance Characteristics:

- (i) It should produce highly accurate tracking results.
- (ii) It should have low memory consumption.
- (iii) It should have reasonable execution time.

Current studies focus on either accuracy or time efficiency, but not both. And also many of them do not consider algorithmic complexity analysis to represent and measure their scalability. Moreover, none of the methods in the literature consider space efficiency. However, in the world of dynamic networks, low resource consumption is definitely a key element to improve the usability of solutions. This is the motivation to propose the TREC method, which has all the desired properties.

3.3. Methodology

In this study, the task of tracking community evolution can be divided into three basic steps: Network Representation, Community Detection, and Evolution Analysis, as shown in Figure 3.12. In the first step, the network data is represented as a sequence of static networks. To do this, the network data is decomposed 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 the last step, the life chain of communities across the time steps is revealed by matching communities and creating their evolution chains.

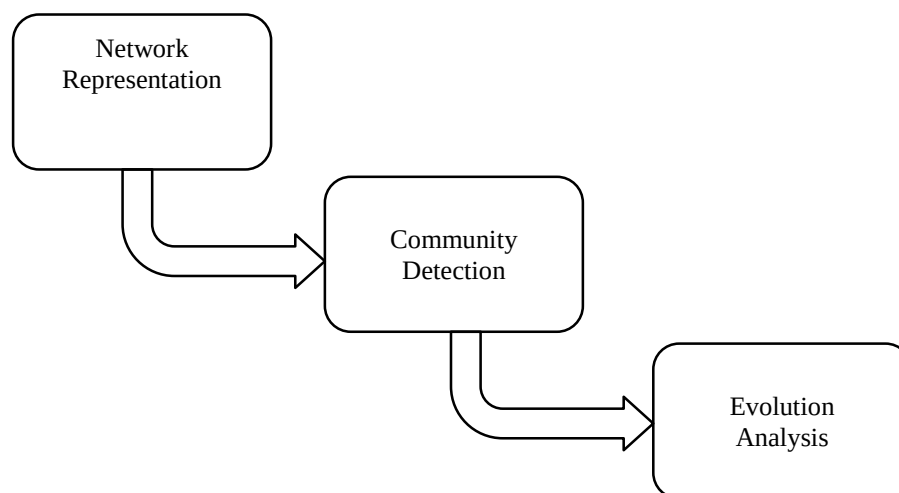


Figure 3.12. Main steps of tracking evolution of communities

After reviewing related work, it appears that there is a need for an event-based method for tracking community evolution that detects all types of evolutionary events for both consecutively and nonconsecutively evolving communities with highly accurate tracking results while consuming low resources, especially in terms of memory space. This need motivates us to develop TREC.

Since the network representation and community detection steps are default for all related methods for tracking community evolution, they are not explicitly included in the steps of the TREC method, as can be seen in Figure 3.13. Additionally, the Louvain method (Blondel, Guillaume, Lambiotte, and Lefebvre 2008) is used to detect communities. Since the Louvain method is one of the best methods among community detection methods (Yang, Algesheimer, and Tessone 2016; Karataş and Şahin 2018b) in terms of both accuracy and execution time, it is preferred. The Louvain method detects disjoint communities. However, there is no obligation to use the Louvain algorithm, instead any community detection method can be used for disjoint communities.

The main steps of the evolution analysis of TREC can be seen in Figure 3.13. The following subsections explain the main steps in detail.

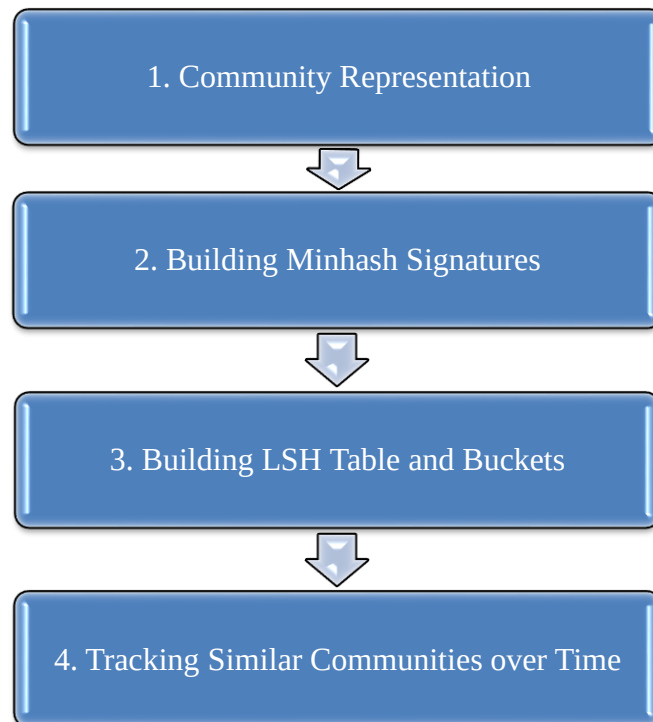


Figure 3.13. Main steps of TREC method

Step 1. Community Representations

For each time step, there is a text file containing the community members on each line (vertex identifiers only, with each vertex identifier represented by positive integers). These text files are first created and kept on disk. Then they are loaded into main memory as a single community vector map v . This vector map v contains community identifiers and current time step information as corresponding key-value pairs. The map is conceptually illustrated in Figure 3.14.

$$\begin{aligned} C_1 &= \{C_1^1, C_1^2, \dots, C_1^k\} \\ C_2 &= \{C_2^{k+1}, C_2^{k+2}, \dots, C_2^l\} \\ &\dots \dots \\ C_{tsc} &= \{C_{tsc}^{z+1}, C_{tsc}^{z+2}, \dots, C_{tsc}^{tcc}\} \end{aligned}$$

Figure 3.14. Community vectors representation for time steps

The abbreviation tsc stands for "time step count", which is the available number of time steps for the dataset used. The t is the time index and $1 \leq t \leq tsc$. For each time step t , there is a finite number of communities and the average number of communities is represented by c . The "total number of communities" - tcc defines the maximum number of communities detected for all tsc time steps.

Step 2. Building Minhash Signatures

Originally, the minhashing implementation works with binary vectors, but it is extended to integer vectors and continuous variables (Chamberlain, Levy-Kramer, Humby, and Deisenroth 2018). Figure 3.15 (a) shows the representation of communities in the network with binary vectors. The length of each community vector is equal to the total number of unique nodes in the network. The presence of all unique node identification numbers (e.g., n) is scored as 0 or 1, where 1 indicates presence and 0 indicates absence in the vectors. Figure 3.15 (b) shows the representation of communities in the network with integer vectors. In an integer community vector, only the identification numbers of the existing nodes are included.

Therefore, the length of each vector is variable and corresponds to the number of nodes in the community. Note that tsc is the total number of steps (e.g., the total number of time steps that make up the network) and tcc is the total number of communities.

Node id	C_1^1	C_1^2	...	C_{tsc}^{tcc}
1	0	1		
2	1	0		
3	1	0		
4	1	1		
$\vdots$	$\vdots$	$\vdots$		
n	0	1		

↑
A binary vector

C_1^1	C_1^2	...	C_{tsc}^{tcc}
1	5		13
5	8		25
13	52		32
21			101
22			103
			249
			303

↑
An integer vector

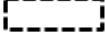
Figure 3.15. Vector representation of communities (a) Binary vector representations and (b) Integer vector representations

The general implementation with the use of bit vectors is described in reference (Leskovec, Rajaraman, and Ullman 2015). In this study, the minhashing implementation is performed using integer vectors since the member identifiers are represented by positive integers. Then community signatures are generated using these integer community vectors.

Minhashing uses several universal hash functions (H) such as $H_i(x) = ax + b \bmod p, i = 1, \dots, h$ where h is used number of hash functions, a, b are random integers and p is a prime where greater than or equal to the number of unique nodes in the dynamic network data set. Using minimum hash values for x 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 is obtained.

Figure 3.16 shows an example of a signature matrix. The columns of the matrix represent community signatures, the rows represent hash functions used, and each cell contains

the associated minimum hash result for the hash function over those community members. tcc (total community count) is the number of communities and tsc is the total time steps in the network data.

 A signature

	C_1^0	C_1^1	C_1^2	...	C_3^{537}	C_3^{538}	C_3^{539}	C_4^{540}	...	C_5^{2880}	C_5^{2881}	C_5^{2882}	...	C_{tcc}^{tsc}
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		..

Figure 3.16. An example of signature matrix

Step 3. Building LSH Table and Buckets

LSH with minhashing technique is used to identify the similarity of communities in Jaccard space. In LSH, the signature matrix generated by minhashing is divided into b bands, where each band contains r rows. This form of the signature matrix is called the "LSH table", which is shown in Figure 3.17(a). These r rows in each band and their intersecting cells of columns contain partial signatures of communities. Each of these signature parts is referred to as a "pickle". A set of buckets are created by grouping similar community IDs in a band. Each band has its own set of buckets as shown in Figure 3.17 (b). If there is no similar community under a band, the number of buckets for that band can be equal to the maximum number of communities. If two communities exist in at least one bucket, they are considered as a candidate pair for similarity. For example, in Figure 3.17(b), communities C_1^1, C_3^{537} and C_5^{2880} have the same pickle value (24, 34); therefore, they are grouped in the same bucket, which may represent similar communities.

Step 4. Tracking similar communities over time

The use of minhash signatures affects both the use of main memory and the ease of finding near neighbors of a given community query in LSH. Moreover, filtering over LSH buckets significantly reduces the computation time in community matching. To evaluate the contributions of using minhash signatures and LSH data structure, additional experimental studies are conducted. The results of these experiments are presented and discussed in Section 3.4.3. Furthermore, concrete measurements for the use of combining LSH with minhash signatures in TREC are given in Section 3.4.

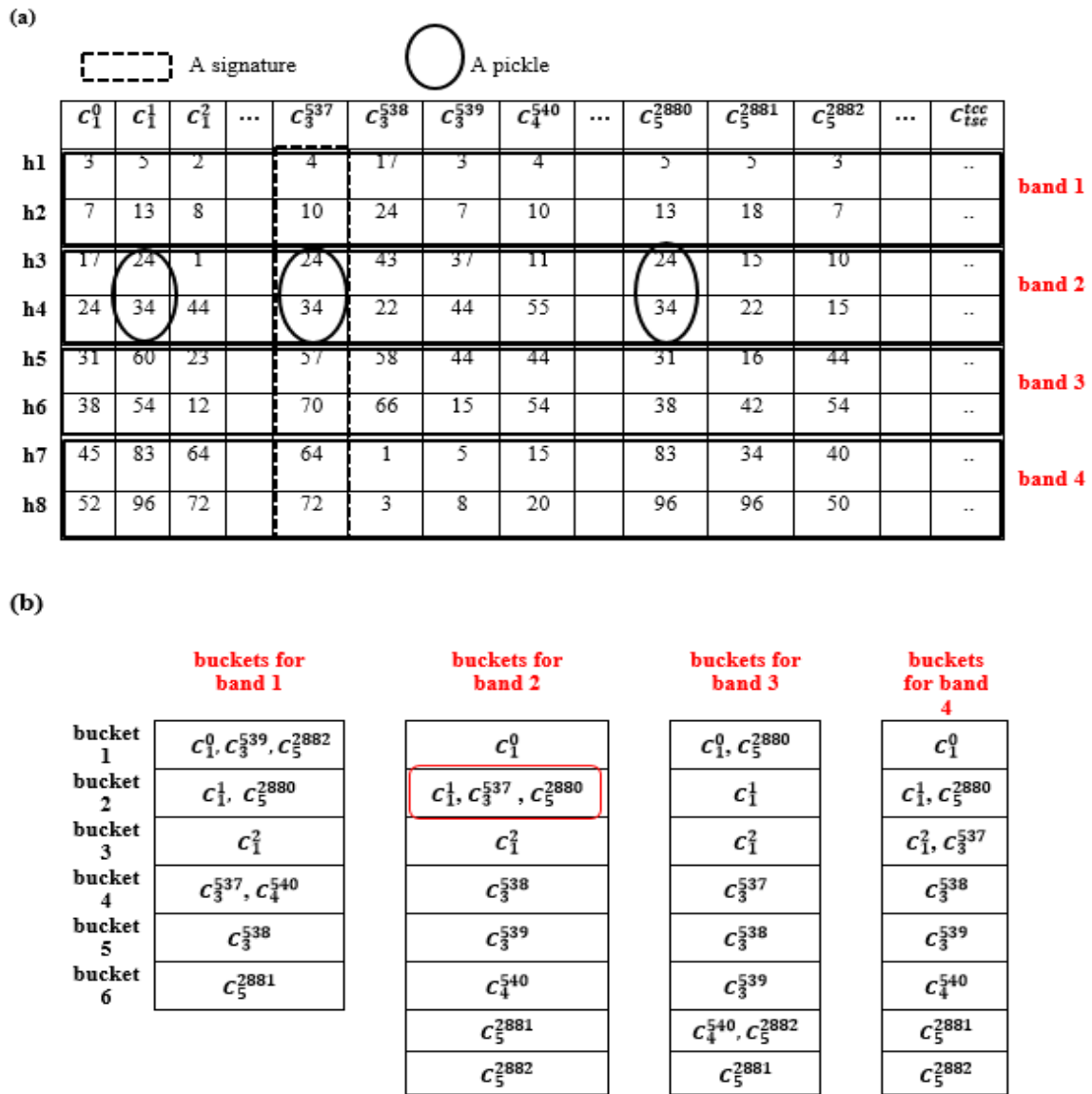


Figure 3.17. An example of “how LSH with minhashing works”. (a) LSH table and (b) LSH Bucket Collections

The main idea of the pseudocode shown in Figure 3.18 is to filter dissimilar communities and possibly provide similar communities for a given community via formed LSH buckets and create/save evolution chains in a text file. Then, the time steps of the filtered communities are checked to decide whether the community is consecutively or nonconsecutively evolved. Then, the evolutionary chains of the tracked communities are stored in a text file. The pseudocode is as follows.

Input: L (List of communities to be tracked), λ , LSH

Output: $tlistFile$ (A text file to store tracking information of similar communities)

```

1.  tlist ← {},
2.  for each community  $C_t^i$  in  $L$ 
3.       $S \leftarrow \{C_t^i\}$ 
4.       $t_j = t + 1$ 
5.      while ( $t_j \leq tsc$ ) // starts to detect consecutively and nonconsecutively evolving
6.          // communities
7.          endFlag = "false"
8.          rCommunities ← set of communities whose  $t$  is the highest in  $S$ 
9.          // for handling both k-splits and other events
10.         for each  $C_{tx}^k \in rCommunities$  // for tracking k communities after the split
11.             cCommunities ← LSH( $C_{tx}^k$ ) // get similar communities of  $C_{tx}^k$ 
12.             for each  $C_{tc}^e \in cCommunities$  where ( $t_j = tc$ ) // consecutively evolution
13.                 Loop D | if  $JS(C_{tx}^k, C_{tc}^e) \geq \lambda$ , then  $S \leftarrow S \cup C_{tc}^e$  and endFlag = "true"
14.                 if (endFlag == "false"), then //nonconsecutively evolution
15.                     Loop E | for each  $C_{tc}^e \in cCommunities$  where ( $t_j < tc$  and  $\min(tc - t_j)$ )
16.                         if  $JS(C_{tx}^k, C_{tc}^e) \geq \lambda$ , then  $S \leftarrow S \cup C_{tc}^e$ 
17.             rCommunities ← {}, cCommunities ← {}
18.              $t_j = t_j + 1$ 
19.         tlist ← tlist  $\cup$   $S$ ,  $S \leftarrow \{\}$ 
20.     write tlist to tlistFile

```

Figure 3.18. A Pseudocode of how to track similar communities over time

The tracking algorithm shown in Figure 3.18 takes L (a list of communities to track), the similarity threshold λ , LSH (LSH table and bucket collections) as input. Recall that this algorithm works with nonoverlapping communities detected by the Louvain algorithm (Criterion #1), and λ is a similarity threshold parameter. Loop A ensures that the tracking process is executed for each community to be tracked. That is, it is used to keep/store the

evolution chains of each community to be tracked in a separate row in *tlistFile*. Loop B is used to execute the next time steps after a time step in which the community was born, if the evolution is not continuous (Criterion #2). Loop C is used to continue tracking when a tracked community splits into new subcommunities (Criterion #4). In addition, the communities returned by the LSH query are examined for all tracked communities. If the community returned by LSH evolves consecutively, Loop D runs; otherwise, Loop E runs for nonconsecutively evolving communities. Both loops D and E test the similarities of community pairs with Jaccard similarity. After the evolution chain of a tracked community, it is stored in *tlist*. When all communities in *L* are processed, *tlist* is stored in *tListFile*.

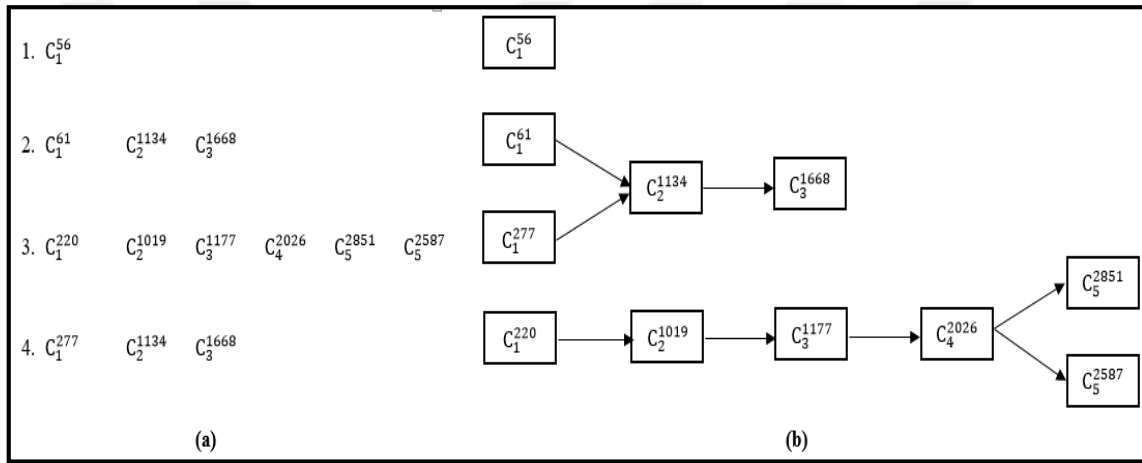


Figure 3.19. An example of (a) content of *tListFile* (b) its correspondent conceptual schema

Figure 3.19 (a) and 3.19 (b) show an example of the contents of *tListFile* and its conceptual scheme. Each line contains an evolutionary chain of a community. For example, C_1^{56} in Line 1 has no other following community in its evolutionary chain. That is, C_1^{56} was born and is never observed again during the observation; thus, it dissolves. Communities C_1^{61} and C_1^{277} are merged and C_2^{1134} is born. Community C_4^{2026} is split into two subcommunities C_5^{2587} and C_5^{2851} . Note that the evolution events are labelled with another subroutine that takes *tlistFile* as input and returns a text file containing the evolution of the communities with event labels (Criterion #3). Basically, it uses the definitions of the communities' evolution events

given in Table 3.1. Moreover, it strictly follows the event formulations in the implementation, which allows k-merge/k-splits cases (Criterion #4).

3.4. Experimental Study

This section presents the data sets used, the experimental setup, the performance and accuracy analysis of TREC, and the evaluation of the results.

3.4.1. Datasets

In this subsection, the datasets used in the experiments are explained in the following subsections. Subsection 3.4.1.1 includes the benchmark datasets and subsection 3.4.1.2 explains the real datasets.

3.4.1.1. Benchmark Datasets

The Greene et al. benchmark datasets (Greene, Doyle, and Cunningham 2010) are used for accuracy analysis. As they contain ground truth information on communities at all time steps and are accessible online at <http://mlg.ucd.ie/dynamic>. The datasets are constructed from four different synthetic graphs, each containing five static networks, meaning that there are five time steps (t) with 15000 nodes (n) to simulate nonconsecutively evolving communities, and containing all event types of community evolution. 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.

The similarity thresholds (λ), the number of hash functions (h), the number of rows (r) and the number of bands (b), and in the last column LSH similarity threshold (LSH_{ST}) for the benchmark datasets are given in Table 3.5. λ and LSH_{ST} values are assumed to be 0.10. That is, the minimum similarity ratio between two communities must be 10% to be called similar.

Table 3.5. Specific parameters used in the accuracy experiment

Benchmark dataset	λ	h	r	b	LSH_{ST}
BirthDeath	0.10	90	2	45	0.10
Expand					
MergeSplit					
Nonconsecutiveness					

3.4.1.2. Real Datasets

The execution time performance of TREC for dynamic large networks is evaluated on real networks. These datasets are:

- Autonomous Systems (AS) dataset (Leskovec, Kleinberg, and Faloutsos 2005) contains a daily communication network of routers from logs. Daily unweighted undirected graphs are constructed from December 1999 to January 2000, where each node has a router identification number and each edge indicates the relationship between each node.
- The DBLP dataset (Tang, Zhang, Yao, Li, Zhang, and Su 2008) contains the authors' co-publications. Data mining and artificial intelligence domains are used to construct unweighted and undirected graphs between the years 2001 and 2013, where each node represents the authors and each edge represents the co-authorships and citations of the publications.
- The Yelp dataset (Yelp Inc. 2019) contains user reviews about businesses, but user friendships are important for our study. Therefore, monthly unweighted friendship graphs

are constructed for each user who has at least one friend from October 2013 to July 2014, where each node represents users and each edge represents a friendship.

- The 2009 Digg friendship dataset contains friendship information on the Digg platform (Hogg and Lerman 2012). Bimonthly, undirected, and unweighted friendship graphs are created from July 2007 to July 2009. Each node represents a user and each edge represents a friendship. The dataset is available in the reference (Lerman, 2012).

3.4.2. Experimental Configuration

In these experiments, the rate of change of community size is assumed to be 5% to determine continuation, growth, and shrinkage events. The parameters λ, h, r, b and LSH_{ST} for TREC are assigned as $\{0.10, 90, 2, 45, 0.10\}$ respectively.

For the experimental analysis, a laptop with the following configuration is used. Intel (R) Core(TM) i7-4712MQ CPU @ 2.30 GHz. Processor, 64-bit Win10 operating system and 16 GB memory. All the methods of competitor tracking and TREC are implemented using C++.

3.4.3. Impact of using Minhashing and LSH

The following experiments are conducted to show and compare the effects of minhashing and LSH techniques separately on the proposed method TREC in terms of their accuracy and required runtimes. These experiments are used to verify and validate each method used in this proposed novel solution:

- Minhashing provides the ability to compute approximate Jaccard similarity values between two community signatures using a signature matrix. Therefore, the method "minhashing_effect_TREC" is developed to show the effects of using minhashing only, which uses the approximate similarity measures instead of the exact JS measures to determine the similarity of two communities. In this experiment, approximate similarities are used in community matching returns from LSH queries.

- In addition, the method "LSH_effect_TREC" is developed to show the effect of applying the LSH technique. Here, the LSH table and buckets are removed from TREC. Instead, approximate JS values of minhash signatures of communities are calculated for all community pairs and then for each community their similar communities are kept in a hash map instead of LSH table and bucketing. In this map structure, the community IDs are the keys and their similar communities are the corresponding values.

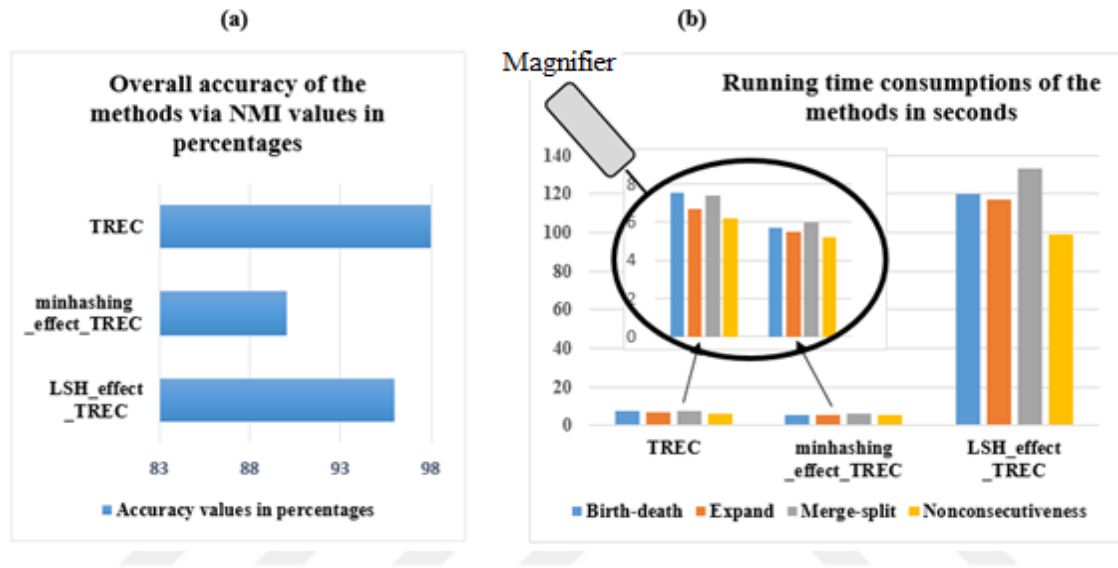


Figure 3.20. (a) Accuracy and (b) running time consumptions of TREC, minhashing_effect_TREC and LSH_effect_TREC methods

The TREC, minhashing_effect_TREC, and LSH_effect_TREC methods are run separately on the benchmark datasets. Figure 3.20(a) and Figure 3.20(b) show the accuracy and runtime consumption of the methods.

The runtime consumption of TREC and minhashing_effect_TREC are very close. Minhashing_effect_TREC consumes slightly less time because it only checks the similarity of the signatures. As can be seen in Figure 3.20(b), the time consumption of LSH_effect_TREC is ten times higher than the others. As expected, the time consumption of LSH_effect_TREC is too high because it checks the signatures of each community pair. Therefore, the hybrid usage of minhashing and LSH in TREC provides better accuracy with reasonable running times and is evaluated in Section 3.4.4 below.

3.4.4. Performance and Evaluation of TREC

The studies of Asur et al. (Asur, Pathasary, and Ucar 2009), Greene et al. (Greene, Doyle, and Cunningham 2010), and Takaffoli et al. (Takaffoli, Fagnan, Sangi, Zaïane 2011) are not considered competitors of TREC because they cannot detect some evolutionary events such as shrink, growth, and continue. However, TREC can detect all types of evolutionary events. In addition, the GED (Bródka, Saganowski, and Kazienko 2012) and SCGI (Gliwa et al. 2012) methods are not considered competitors of TREC because they work with overlapping community structures and only with continuously evolving communities. TREC, on the other hand, operates with a disjoint community structure and can track both consecutive and non-consecutive evolving communities.

Therefore, the method of Tajeuna et al. (Tajeuna, Bouguessa, and Wang 2016) and the ICEM method (Mohammadmosaferi and Naderi 2020) are considered as competitors of TREC because they share the same functional properties, such as tracking type (e.g., both consecutive and non-consecutive evolutions), k-community merging/splitting, tracking capability (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 new TREC method is investigated in terms of three main components such as complexity analysis, accuracy analysis, and execution time analysis by comparing the competitors' methods.

3.4.4.1. Complexity Analysis

Time and space complexity are studied as performance measures because they are very important in determining the computational limits of methods to determine the usefulness of the method.

Therefore, in this subsection, the required computation time and memory of TREC are calculated and the corresponding complexity values of the competitor's works are given to show the usefulness of TREC method compared to others. The network modeling and community detection steps are excluded from the complexity analysis because they are common steps for all tracking methods. Therefore, as mentioned earlier, TREC has four basic steps (as shown in

Figure 3.13). However, the computation of minhash signatures has the highest computation time as $O(hnt)$, where h is the number of hash functions used and n is the number of unique members in the network, and t is the total time steps. The community representations, LSH table, and buckets reside in main memory during the execution of TREC. Therefore, the total memory required is $O(n + 2v)$, where v is the number of total community vectors (or number of communities) over all time steps.

Table 3.6. Complexity analysis of TREC and competitor methods

Method	Time Complexity	Space Complexity
Tajeuna et al.'s method	$O(v^2 \log(v))$	$O(v^2)$
ICEM with Evolution Chain	$O(tn \log(n))$	$O(2n)$
TREC	$O(hnt)$	$O(n)$
GED	$O(tc^2n)$	$O(3n)$

Required computation time and memory TREC and competing methods can be seen in Table 3.6. Since the ICEM method (Mohammadmosaferi and Naderi 2020) does not generate evolutionary chains, a balance must be found between ICEM and TREC in terms of generating evolutionary chains of communities. Therefore, a subroutine is added to the ICEM method and its title is concatenated with "with Evolution Chain".

Note that the method GED is the most primitive method among the competing methods. That is, it cannot track nonconsecutively evolving communities, consumes the most memory and CPU time among them. Therefore, it is not compared with the others after this point.

3.4.4.2. Accuracy Analysis

Another performance measure is the accuracy of the tracking results of the TREC method. Since accuracy is as important as the other 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 (Blondel, Guillaume, Lambiotte, and Lefebvre 2008)) are used for all competing methods. NMI values (Danon, Diaz-Guilera, Duch, and Arenas 2005) are used to measure the accuracy of detected community structures. NMI 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. The higher the NMI values are, the more successful communities are detected.

Table 3.7. Accuracy scores of the methods per datasets

	Tajeuna et al.'s method	ICEM with Evolution Chain	TREC
BirthDeath	0.994451	0.966588	0.994537
Expand	0.98348	0.983544	0.982961
MergeSplit	0.938352	0.9567722	0.946782
Nonconsecutiveness	0.989222	0.9913834	0.990556
Average score	0.976	0.974	0.979

For each time step, a community structure is generated from the tracking list. NMI compares the detected community structures with the ground truth community information for each time step and generates some kind of similarity value representing accuracy. The NMI values for each dataset are calculated for all methods via dedicated software referred in (Lancichinetti, Fortunato, and Kertész 2009).

Table 3.8. The highest memory usage in Mega Bytes during their executions and Execution Time of the methods

		Methods								
		Dataset Characteristics			Tajeuna et al.'s method		ICEM		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	✖	✖	352.5	2.47s.	60.9	11.2 s.
	2009 Digg friends	12	64183	329	7.9	44 min. 46s.	10.3	5.1 s.	3.8	12.6 s.

Table 3.7 shows the accuracy results of the TREC method and its competitors for each data set in terms of NMI values. All methods show almost the same accuracy for the “Expand” and “Nonconsecutiveness” datasets. The Tajeuna et al. method and the TREC method are very close in terms of accuracy, while the ICEM with Evolution Chain method has lower accuracy than them for the “BirthDeath” dataset. The Tajeuna et al.’s method shows the lowest performance for the “MergeSplit” dataset. ICEM with Evolution Chain method shows the best performance for the dataset and TREC method is in between. Figure 3.21 illustrates the accuracy performance given in Table 3.7.

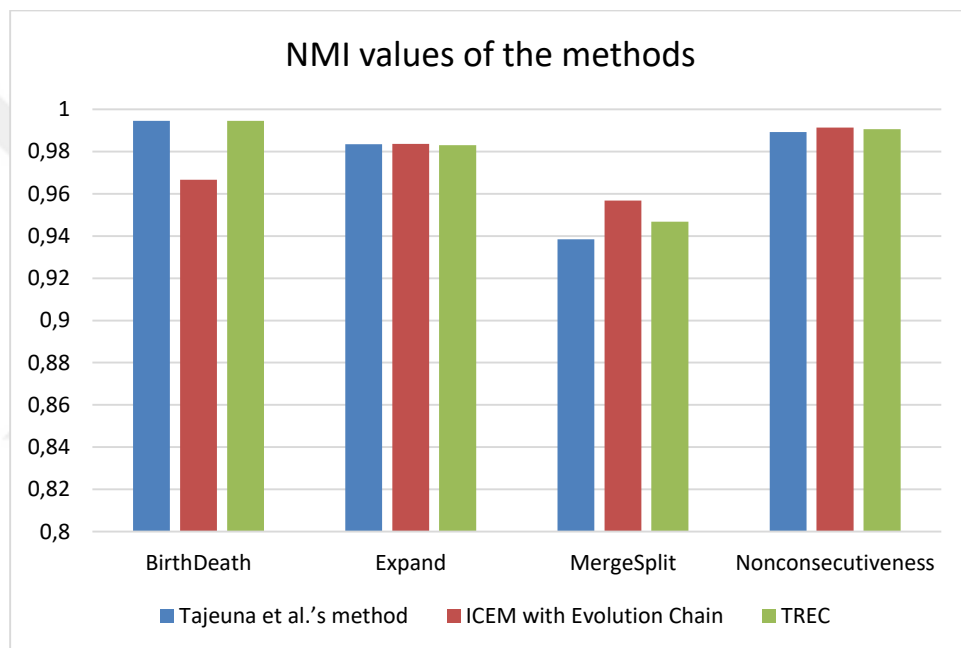


Figure 3.21. Accuracy values of the TREC and its competitors on the datasets

When the average accuracy performance scores of the methods are regarded, it is seen that Tajeuna et al.’s method, ICEM with Evolution Chain and the TREC method accuracy values are {97.6, 97.4 and 97.9} in percentages. Thus, it is concluded that their performances are almostly same.

3.4.4.3. Real Time Memory and Execution Time Analysis

The last performance measure considered is the real-time storage and execution time performance of TREC and its competitors. The reason is that both performances are important in the world of dynamic networks. The most common method of profiling software applications is to use "Diagnostic Tools". Since TREC and its competitors are implemented in C++, a free and reliable tool for this language is needed. After searching available diagnostic tools on the Internet, it was found that Visual Studio meets the requirements. Therefore, this profiling task is performed in Visual Studio 2019 Community Edition environment. With these tools, it is possible to analyze CPU and memory usage in both debugging and execution phases. In our analysis, we consider CPU time and memory usage in the execution phase.

Table 3.8 shows some properties of the dataset and the performance measures for the execution time of TREC and its competitors. Both benchmark datasets and real datasets are considered in the table. The column "*Dataset Characteristics*" contains the subcolumns "*t*", "*n*" and "*c*" for each dataset. The subcolumn "*t*" indicates the number of time steps that make up the dynamic network. The subcolumn "*n*" specifies the number of nodes that each dataset has. The subcolumn "*c*" specifies the number of average communities per time step. The "*Methods*" column contains the performance results of TREC and its competitors such as the Tajeuna et al.'s method and the ICEM method. The subcolumns "*(1) in MB*" show the highest memory consumption in Mega Bytes and the subcolumns "*(2)*" show the CPU consumption in seconds/minutes and hours during the execution time of each method.

Profiling using the Tajeuna et al.'s method is performed on the Yelp dataset until our storage space is exhausted. About six hours (eg., 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 "✖" sign in the corresponding cells of Table 3.9.

Examination of Table 3.8 shows that 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. The results are evaluated in detail in the following subsection.

3.4.5. Discussion on the Results

In this subsection, the algorithmic (time and space analysis) and analytical (accuracy and real time analysis including the memory space and execution time requirements) results of the TREC method and its competitor methods are evaluated.

Table 3.9 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 has a quadratic complexity in terms of time and space. Comparing the complexity order of the two methods, it can be seen 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 3.9. Complexity orderings of TREC and competitor methods

Time Complexity Ordering	$O(tn \log(n)) < O(hnt) < O(v^2 \log(v))$ i.e., ICEM with Evolution Chain < TREC < Tajeuna et al.'s method
Space Complexity Ordering	$O(n) < O(2n) < O(v^2)$ i.e., TREC < ICEM with Evolution Chain < Tajeuna et al.'s method

From Figure 3.21 and Table 3.7, it can be seen that all the methods show almost the same accuracy performance and the percentage accuracy on benchmark datasets is around 98%. The accuracy of the methods on real datasets cannot be measured because the datasets do not contain real community information.

Table 3.8 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 of

execution time for the same data sets. 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.

In terms of space required by the methods, the method of Tajeuna et al. is the most space-consuming among them, while the TREC method is the most space-efficient, as shown in Table 3.8. The ICEM with Evolution Chain method requires a memory space equal to the constant multiple of the memory requirement of the TREC method.

As shown in Table 3.8, the number of nodes and the number of time steps 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 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.

As shown in Table 3.8, the number of nodes and the number of time steps used to construct the network are different for real datasets. The Tajeuna et al.'s method tends to consume more execution time and memory as the number of communities increases. The TREC method and the ICEM with Evolution Chain method consume more memory when the number of nodes in the dataset increases, as shown in the table. The methods tend to consume more execution time when the communities are dense, as in the "2009 Digg friends" dataset.

In summary, all the methods compared are functionally the same but differ in their performance in terms of space requirements and execution time. As shown in Table 3.8 and Table 3.9, the Tajeuna et al.'s method is inefficient compared to the TREC method and the ICEM with Evolution Chain method in terms of both memory requirements and execution time. Moreover, there is a time-space trade-off between ICEM with Evolution Chain and the TREC method. While the ICEM with Evolution Chain method outperforms the TREC method in terms of execution time, the TREC method outperforms the ICEM with Evolution Chain method in terms of memory requirements and slightly better accuracy.

3.5. Concluding Thoughts and Future Work

In this section, we introduce a space-efficient new method for tracking community evolution in dynamic networks - TREC. TREC uses an independent community detection and matching approach. In this approach, communities can first be detected for each time step by any community detection method. Then, the detected communities are matched in terms of temporal order only if they are similar, which is called community matching in TREC. Most of the computational resources are used for the community matching step. Competitors' works focus either only on accuracy of tracking results or on fast computation, but none of them focus on low memory consumption. However, in the world of dynamic networks, resource consumption is crucial. TREC consumes less memory compared to competing works by extracting signatures of communities with minhashing and speeds up the task of finding similar communities by pruning the meaningless community comparisons with LSH. LSH with minhashing technique is used for the first time in tracking community evolution with TREC method. Compared methods and TREC method are functionally same but vary in terms of their performance characteristics. Compared with the work of competitors, TREC method is superior in terms of memory consumption and slightly more accurate tracking results with reasonable runtime.

In further research, TREC method can be extended as: (i) a fast comparison of similar communities in the same LSH buckets can be performed through parallelization, (ii) TREC can be modified to work with overlapping community structures, and (iii) TREC can be combined with machine learning models to predict the future evolution of communities.

CHAPTER 4

A CASE STUDY: PREDICTING COMMUNITY EVOLUTION WITH RESULTS of TREC METHOD

"Can TREC results be used to predict community evolution in dynamic networks?". This chapter is dedicated to answering this research question and does not introduce a novelty in either machine learning classifiers or prediction methodology.

Therefore, the task of predicting community evolution is first presented. Then a brief literature review is given. Then the workflow we use for this task is described. Then, the preliminary results are presented. Finally, the results are discussed and concluding considerations are given.

4.1. Predicting the Evolution of Communities

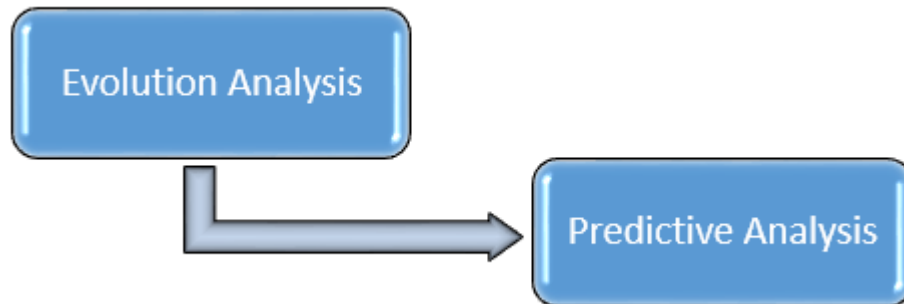


Figure 4.1. Fundamental steps of predicting community evolution

The task of predicting the evolution of communities can be divided into two basic steps; evolution analysis and predictive analysis (see Figure 4.1). In the evolution analysis step, the evolution of communities is tracked. Recall that the TREC method is one of the tools for community evolution analysis. The predictive analysis step takes the output of

the evolution analysis step (i.e., evolution chains with event labels) as input, processes it, and returns a list of predicted events such as "grow", "continue", "shrink", "merge", "split", and "dissolve", respectively.

Note that the length of an evolution chain (L) is measured by the number of communities in the chain. Suppose that we are given $\{C_1^4, C_2^6, C_4^{12}\}$ as an evolution chain, then $L = 3$.

4.2. Related Work

Similar to tracking the evolution of communities, predicting their future members or evolution events is one of the main topics. Currently, there are two different prediction tasks: "link prediction" and "evolution event prediction". Link prediction is about predicting the probability that two currently unconnected nodes will be connected in the next time step (Liben-Nowell and Kleinberg 2007). However, the focus of this case study is on predicting community evolution events.

Papers in the literature of the last decade (between 2011 and 2021 years) are examined. Since the TREC method uses the independent community detection approach, the focus is on the works that use this approach. The most widely used approach to predict community evolution is the use of machine learning classifiers. The works that use this approach follow a two-step methodology: (1) analyzing community evolution and (2) applying supervised classifiers based on selected community features. Related work is summarized below.

In their paper, Brodka et al. (Bródka, Kazienko, and Kołoszczyk 2012) present and evaluate a supervised learning method for predicting the evolution of communities with respect to six events of community evolution such as growing, shrinking, continuing, merging, splitting, and dissolving. They use the Group Evolution Detection method (GED) (Bródka, Saganowski, and Kazienko 2012) to detect events between successive time steps and create event sequences to describe the evolution of a given community. Each event sequence consists of the member sizes and events of all three previous communities. These sequences serve as input to the classifiers such as Naïve Bayes, Decision Tree, Random Forest and others provided by WEKA Data Mining Software

(Hall, Frank, Holmes, Pfahringer, Reutemann, and Witten 2009) to predict the next event for a given community.

İlhan and Ögüdücü (Ilhan and Ögüdücü 2013) propose a new approach to predict the next event of a community using a time series ARIMA model. In their study, community events are predicted by predicting community characteristics. The feature values are used to classify the possible events.

Takaffoli et al. (Takaffoli, Rabbany, and Zaïane 2014) use a two-step technique to predict the near future of a community through supervised learning. In this technique, they first decide whether the community survives, and then make the prediction whether the community survives. They diversify the type of features by using not only structural features of the community, but also features of influential members, temporal changes in features, contextual attributes, and features of past events. They consider only evolutionary sequences that have only two lengths.

Saganowski et al. (Saganowski, Bródka, Zygmunt, Kazienko, and Koźlak 2015) present two methods for predicting the following evolution event of a community. The first method uses the Stable Group Changes Identification (SGCI) method (Gliwa, Brodka, Zygmunt, Saganowski, and Kazienko et al. 2013) and the other uses the GED method (Bródka, Saganowski, and Kazienko 2013). They use the CPM method (Palla, Derényi, Farkas, Vicsek 2005) for community detection. The authors use evolution chain lengths, group features (e.g., size, density, leadership, etc.), node features (e.g., total degree, in-degree, etc.), and group aggregation (e.g., sum, average, minimum, and maximum). They then perform feature selection using ordinary (J48 and Random Forest) and ensemble classifiers (AdaBoost and Bagging). They conclude that longer group history leads to better prediction and the most recent group history has the largest impact on the next community change.

Diakidis et al. (Diakidis, Karna, Fasarakis-Hilliard, Vogiatzis, and Paliouras 2015) address the problem of predictability of community evolution as a task of supervised learning. However, they predict four events of community evolution, namely continuation, shrinking, growth and dissolution. They use both sequential (e.g., Conditional Random Fields with Linear Chain and with Skip Chain) and ordinary classifiers (e.g., Naïve Bayes, Bayes Net, Logistic Regression, SVM, etc.) for the prediction task and compare the performance of the classifiers. These classifiers were trained on structural (e.g., size, density, etc.), content (topic diversity with TF-IDF), and

contextual features (e.g., number of hashtags, size of tweets, and number of tweets with promotional URLs, etc.), as well as the previous state of a community as features for Twitter. They conclude that the sequential features are better than the ordinary ones because they also capture the past information first.

Pavlopoulou et al. (Pavlopoulou, Tzortzis, Vogiatzis, and Paliouras 2017) present a framework for predicting community evolution. They study how past evolutions of a community influence the prediction of four evolution events such as growth, continuation, shrink, and dissolution. They use some structural (e.g. density, cohesion, diameter, etc.) and temporal features (e.g. lifespan, aging, join nodes ratio and left nodes ratio, etc.) for prediction through supervised learning. They also specify the number of ancestors to be used for computing temporal features, e.g., two or four ancestors according to their dataset from Mathematics Stack Exchange. They used the GED method (Bródka, Saganowski, and Kazienko 2013) to track community evolution and Support Vector Machine (SVM) with RBF kernel (an exponential kernel) as a classifier for predicting the next evolution event. However, they did not consider merge and split events.

Dakiche et al. (Dakiche, Tayeb, Slimani, and Benatchba 2019b) proposed a method for predicting community evolution by capturing the interdependence of rates of change in characteristics describing a community over time instead of actual values. They looked only at rates of change in the structural and influential member characteristics of a community. They examined the length of evolution sequences and concluded that the length of the sequences directly affects the amount and quality of information obtained. However, the quality of information may decrease with long sequences.

Dakiche et al. (Dakiche, Tayeb, Benatchba, and Slimani 2021) propose a new framework for studying the distribution of activities over time to enable proper partitioning of the network. They claim that a properly partitioned network enables more accurate prediction of community events. After applying their novel network partitioning method, they proceed with a simple prediction method. That is, they apply the method GED (Bródka, Saganowski, and Kazienko 2013) to detect group evolutions. Then they proceed to the prediction part. For this task, they specify characteristics. In their study, structural (e.g., density, cohesion, size ratio, etc.) and influential member characteristics (e.g., average leadership degree, average leadership closeness, and average leadership

Table 4.1. Overview of the mainstream approaches for predicting community evolutions

Study	Brodka et al. (Bródka, Kazienko, and Kołoszczyk 2012)	İlhan& Ögüdücü (İlhan and Ögüdücü 2013)	Takaffoli et al. (Takaffoli, Rabbany, and Zaiane 2014)	Saganowski et al. (Saganowski, Bródka, Zygmunt, Kazienko and Koźlak 2015)	Diakidis et al. (Diakidis, Karna, Fasarakis-Hilliard, Vogiatzis, and Paliouras 2015)	Pavlopoulou et al. (Pavlopoulou, Tzortzis, Vogiatzis, and Paliouras 2017)	Dakiche et al. (Dakiche, Tayeb, Slimani, and Benatchba 2019b)	Dakiche et al. (Dakiche, Tayeb, Benatchba, and Slimani 2021)
Year	2012	2013	2014	2015	2015	2017	2019	2021
Tracking Method	GED	A specific method	A specific method	SCGI	GED	GED	GED	GED
Prediction Manner	C	C	C & N	C	C	C	C	C
Event missed	None	None	Continue	None	Merge Split	Merge Split	None	None
Software	CFinder ¹ Weka ²	Weka	MODEC Weka	CFinder Weka	CFinder Weka CRFSuite ³	Weka	CFinder Weka	Weka

¹ <http://www.cfinder.org/>

² <https://www.cs.waikato.ac.nz/ml/weka/>

³ <http://www.chokkan.org/software/crfsuite/>

eigenvector) are used. Later, well-known supervised learning classifiers such as J48, Random Forest, Bagging and SVM were used.

Table 4.1 summarizes related works, with some important criteria listed in the first column. For tracking, they mainly use GED (Bródka, Saganowski and Kazienko 2013), SCGI (Gliwa, Brodka, Zygmunt, Saganowski and Kazienko et al. 2013) and some specific methods developed by them. In the "Prediction Manner" row, the studies make predictions for consecutively or nonconsecutively evolving networks or both, where *C* stands for consecutively evolving communities and *N* for nonconsecutively evolving communities. Only the ML model of Takaffoli et al. (Takaffoli, Rabbany, and Zaïane 2014) can predict the next stage of a community either at the next time step or at later time steps. While the method of Brodka et al. (Bródka, Kazienko, and Kołoszczyk 2012), Saganowski et al. (Saganowski, Bródka, Zygmunt, Kazienko, and Koźlak 2015), the method of İlhan and Ögüdücü (Ilhan and Ögüdücü 2013), and the two methods of Dakiche et al. (Dakiche, Tayeb, Slimani, and Benatchba 2019b and Dakiche, Tayeb, Benatchba, and Slimani 2021) detect all possible events of community evolution, others cannot characterize all events for prediction. For software attributes, Weka is used for developing, training, and testing ML models, CFinder is used for applying CPM (Palla, Derényi, Farkas, Vicsek 2005) and MODEC (Takaffoli, Sangi, Fagnan, and Zaïane 2011) for community tracking, and CRFSuite is used for sequential classifiers such as Conditional Random Fields (CRF) (Lafferty, McCallum, and Pereira 2001).

After reviewing the literature, the following conclusions are drawn:

1. If the distribution of event classes is highly imbalanced, a balancing dataset process is required.
2. The features (i.e., properties of communities) used may be structural, temporal, contextual, and/or leadership specific. One, all, or a combination of the feature types may be used.
3. Since dependencies between features and the presence of unrelated features on the prediction event are possible, features must be eliminated.
4. All related works can use different methods to track community evolution, such as GED, SCGI or specific methods they introduce. So we can use TREC as well.
5. To be realistic, predictions must be made for both continuously evolving and nonconsecutively evolving communities, although all related works make

predictions only for continuously evolving communities, with the exception of the work of Takaffoli et al. (Takaffoli, Rabbany, and Zaïane 2014).

6. All types of evolutionary events must be predicted except "form". Since a community must be formed to predict its evolution.
7. All related works use WEKA as machine learning software. So we can use it as well.

4.3. Workflow for Prediction Community Evolution

In this section, we explain the workflow we use to predict community evolution. Before diving into our workflow, we summarize predictive analysis below.

Predictive analysis (or predictive modeling) is a technique that uses mathematical and computational methods to predict an event or outcome. It should be noted that it does not define what exactly will happen in the next time step, but helps us identify the pattern of behavior for prediction. That is, a machine learning model is used to predict an outcome in a future state based on the changes in the model inputs. These model inputs are called features, and the outcome variable is called a response variable. In our case study, a response variable (e.g., a class type) is a community-related property that can quantify changes in a community over time. A feature is a property (e.g., a community characteristic) that can influence the response variable.

Figure 4.2 illustrates the workflow we used. The numbers from 1 to 5 denote the processes. The community information files of a dynamic network serve as inputs to the first process (e.g., evolution analysis with TREC) and the second process (e.g., obtaining ground truth events for communities). The output of the first process is used for the third process (e.g., the preparation of training data). The training data is then used in the fourth process (e.g., training the classification models) for training the machine learning classifiers. In the fifth process (e.g., k-fold Cross Validation), the prediction success of the classifiers is tested and evaluated. At the end of the workflow, summary files are created for the prediction results. In the remainder of this section, the workflow is described and evaluated in detail.

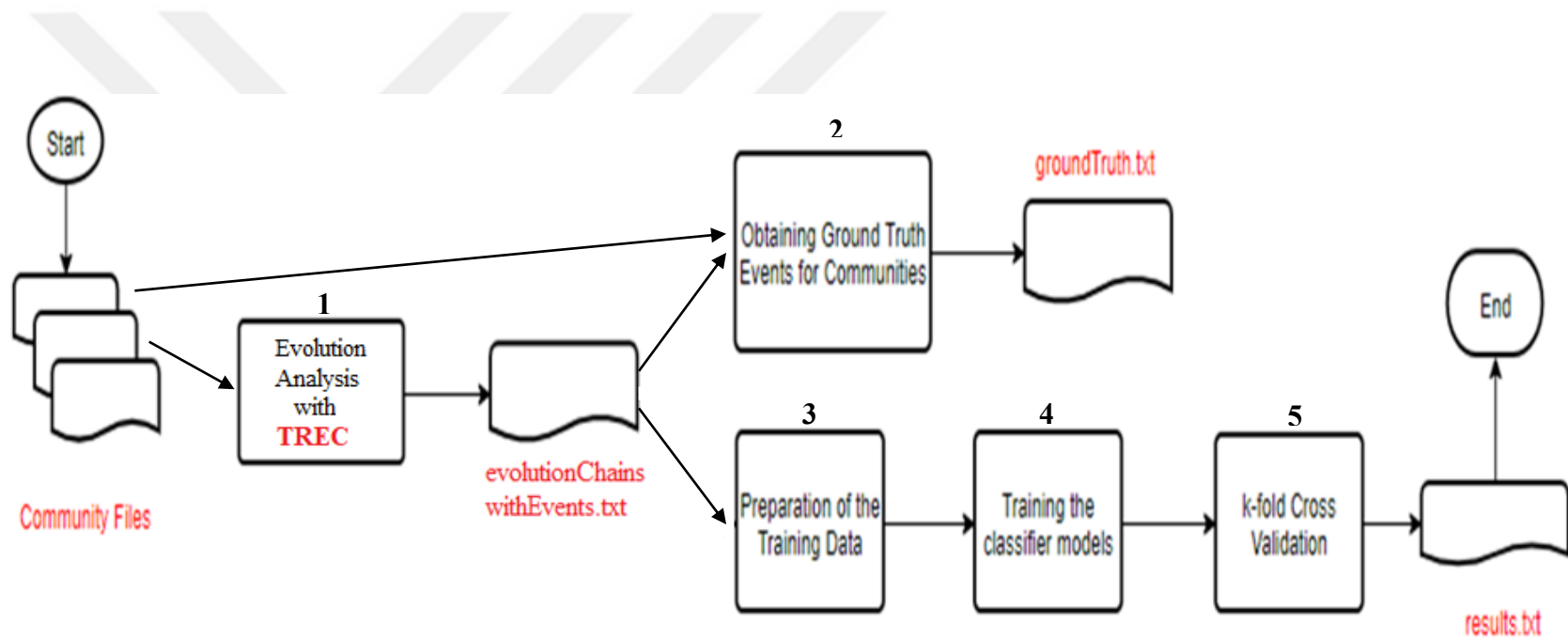
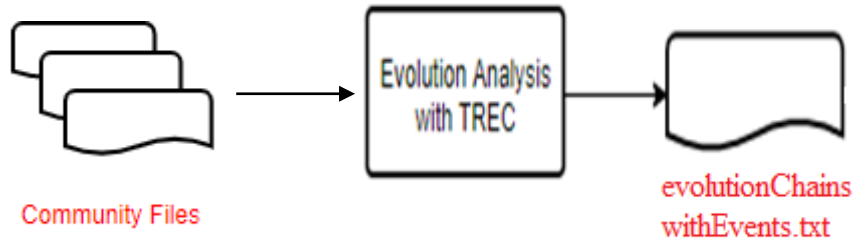


Figure 4.2. The workflow we follow for this case study

Process 1. Evolution Analysis with TREC



As a reminder, an evolving community is characterized by the sequence of communities that were followed during its lifetime. As for the evolution chain (i.e., evolution history) indicates the status of the evolving community at a particular time step. For example, consider the evolution chain shown in Figure 4.3. Since it contains split events, it is written as different evolution chains, as many as there are splits. Therefore, the following evolution chains occur.

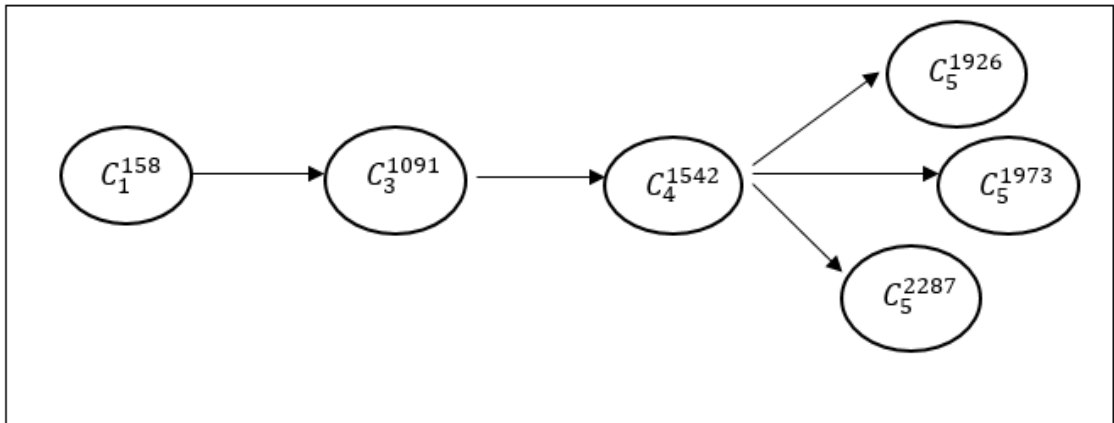


Figure 4.3. A sample evolution for community C_1^{158}

Chain 1: $\{ C_1^{158}, C_3^{1091}, C_4^{1542}, C_5^{1926} \}$

Chain 2: $\{ C_1^{158}, C_3^{1091}, C_4^{1542}, C_5^{1973} \}$

Chain 3: $\{ C_1^{158}, C_3^{1091}, C_4^{1542}, C_5^{2287} \}$

If one wants to label the evolution events in the evolution chain, then the same chains are written in the form below. This form is especially needed for preparing input for the prediction.

Chain 1: form C_1^{158} continue C_3^{1091} continue C_4^{1542} split C_5^{1926}

Chain 2: form C_1^{158} continue C_3^{1091} continue C_4^{1542} split C_5^{1973}

Chain 3: form C_1^{158} continue C_3^{1091} continue C_4^{1542} split C_5^{2287}

Therefore, this process takes information about the community structure of the dynamic network as input and produces a text file of evolution chains with events as output. The output file (e.g. *evolutionChainswithEvents.txt*), shown in the figure below, contains all evolution chains with different lengths for all tracked communities.

```

:
C1 form
C158 form
C158 form C1091 continue
C158 form C1091 continue C1542 continue
C158 form C1091 continue C1542 continue C1926 split
C158 form C1091 continue C1542 continue C1973 split
:

```

Process 2. Obtaining Ground Truth Events for Communities

Supervised machine learning classifiers learn from labeled data. After learning from the labeled data, they classify the test data by associating patterns to the unlabeled data. Later, their decisions are compared with the labels of the test data to measure the classification performance. Since we use supervised machine learning classifiers for prediction, we need ground truth data. The process for obtaining ground truth data is explained in Appendix B. The output of this process (e.g., *groundTruth.txt*) is shown in the figure below. This file contains information about the community and the ground truth pair on each line.

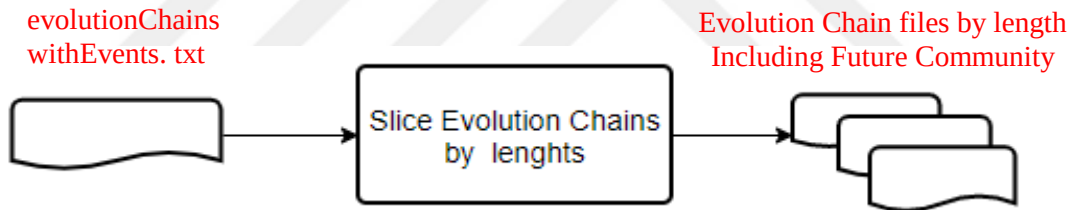
```

      :
C1 dissolve
C158 grow
C1091 continue
C1926 split
      :

```

Process 3. Preparation of the Training Data

This process takes the output of the TREC method (e.g. *evolutionChainswithEvents.txt*) and generates the training data. It contains some subprocesses such as slice evolution chains by lengths, feature determination and convert files by slicing events into arff format for WEKA. The slice evolution chains by lengths subprocess is fed with the output of the TREC method. This subprocess slices the evolution chains by chain length and adds the next community identification number and generates the output text files.



The output files are seen in the figure below for $L = 1$, $L=2$ and $L=3$.

```

      :
C1 form C1
C158 form C1091
C1091 continue C1542
C1542 continue C1926
C1542 continue C1973
      :

```

When $L = 1$

```

      :
C158 form C1091 continue C1542
C1091 continue C1542 continue C1926
C1091 continue C1542 continue C1973
      :

```

When $L = 2$

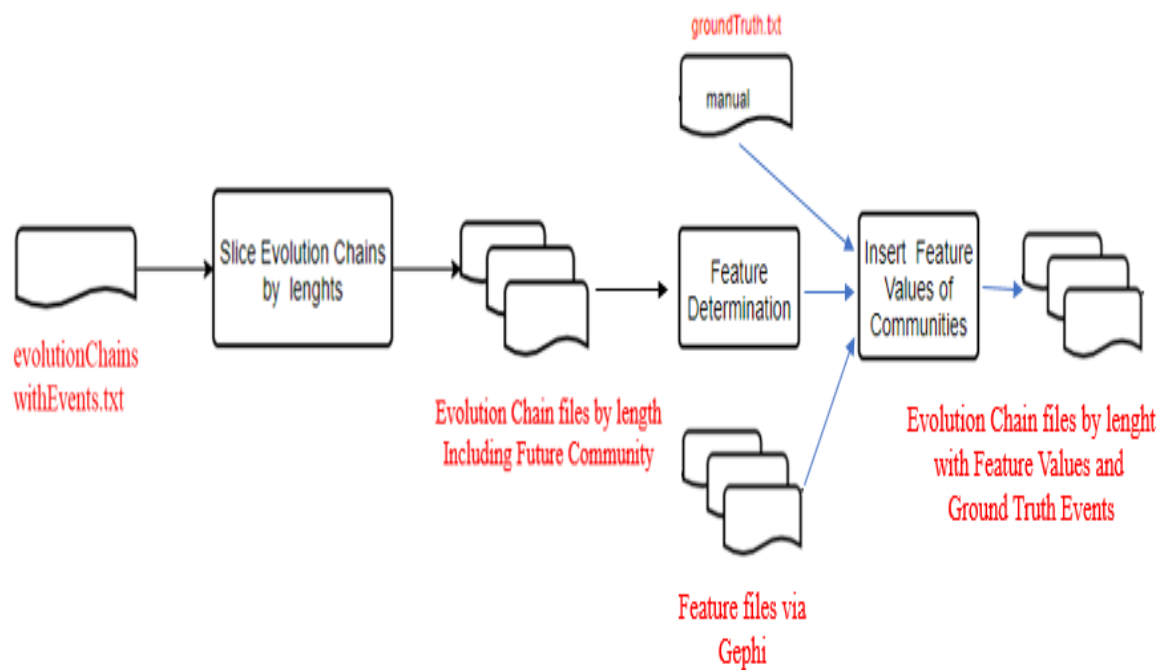
```

      :
C158 form C1091 continue C1542 continue C1926
C158 form C1091 continue C1542 continue C1973
      :

```

When $L = 3$

Later, the feature determination subprocess comes into play. After reviewing the literature, it is found that structural, temporal and leadership features of communities are considered. Structural features such as size, density, and cohesiveness of communities, temporal features such as the extent to which structural features change over time, and leadership features are generic features. That is, these types of features are calculated based on the structure of the network and are not dependent on a specific area. The features identified are explained in Appendix C.



After identifying the features, we can focus on the flow of the process, which is shown in the following figure. The values of the identified features are obtained using the graph visualization tool Gephi⁴. Then the process replaces the community identification numbers with the feature values and the next community identification number with the ground truth events. At this point, some files are generated for different chain lengths. In the figure below, one of the generated files can be seen (Evolution Chain file of length 1 with Feature Values and Ground Truth Events text file).

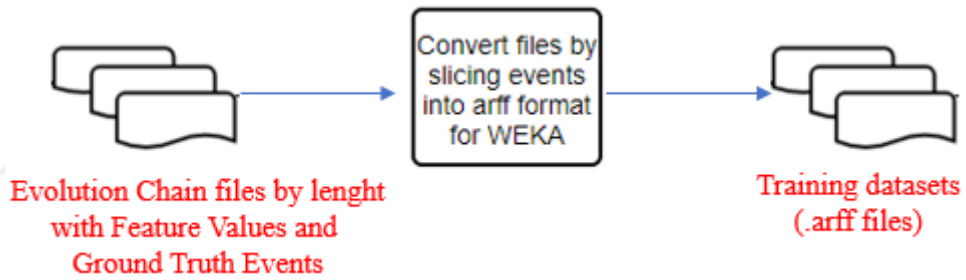
⁴ <https://gephi.org/>

```

11,0.490909,0.364547,785.714,6.27273,0,0,0,1,dissolve
60,0.136723,0.189552,85.9394,9.95,0,0,0,1,grow
64,0.131944,0.192392,76.3814,10.3906,1,36,0.538995,2,shrink
56,0.148701,0.270003,89.3142,10.1964,1,36,0.538995,3,shrink
28,0.267196,0.424487,211.64,8.85714,0,0,0,1,continue
28,0.26455,0.374702,213.09,8.78571,0,0,0,4,grow
35,0.211765,0.311494,164.268,8.97143,1,29,0.659186,2,continue
11,0.527273,0.461449,785.714,6.45455,0,0,0,1,grow
13,0.448718,0.422494,625,6.69231,0,0,0,2,shrink
12,0.484848,0.428542,654.545,6.83333,1,17,0.58046,3,shrink
16,0.433333,0.25605,438.356,7.8125,0,0,0,1,grow
29,0.236453,0.222822,223.535,8.10345,0,0,0,2,grow

```

Next, the third subprocess comes into play, shown in the figure below. This subprocess slices the text files according to their evolution event types. Then, these sliced text files are converted to the arff (e.g., attribute-relation file format) extension to feed the classifiers from WEKA.



The arff files contain two main segments, namely attributes and data. In the attribute segment, the name and type of each attribute must be specified. The data segment consists of the records obtained from the evolution chain files by length with the feature values and the ground truth event files, where only the ground truth event is replaced by a binary value by the sliced event. This is because community evolution prediction is modeled as some binary classification problems. The following figure shows some example lines of the arff file for the merge event with $L=1$.

Note that in this study, six binary classification problems are defined such that a response variable for each evolution event is a binary categorical variable. That is, each response variable takes yes or no values such as continue{yes,no}, grow{yes,no}, shrink{yes,no}, merge{yes,no}, split{yes,no} and dissolve{yes,no}.

```

@relation GreeneMergeSplitBenchmarkWithTREC
@attribute 'C1_Size' numeric
@attribute 'C1_Density' numeric
@attribute 'C1_CC' numeric
@attribute 'C1_Cohesion' numeric
@attribute 'C1_AvgdegreeCentrality' numeric
@attribute 'C1_LeadersCount' numeric
@attribute 'C1_LeadersDegreeCentrality' numeric
@attribute 'C1_LeadersEigen' numeric
@attribute 'C1_event' numeric
@attribute 'merge' { yes, no}
@data
11,0.490909,0.364547,785.714,6.27273,0,0,0,1,no
60,0.136723,0.189552,85.9394,9.95,0,0,0,1,no
5,0.9,0.7,3125,4.2,0,0,0,5,yes
12,0.469697,0.395058,861.244,5.75,0,0,0,1,no
5,0.9,0.461111,2083.33,5.4,0,0,0,3,no
10,0.622222,0.484234,680.272,7.7,0,0,0,5,yes
20,0.326316,0.52438,421.053,6.85,0,0,0,1,no
11,0.490909,0.404429,717.391,6.63636,0,0,0,5,yes
10,0.444444,0.397301,813.008,6.1,0,0,0,6,yes
20,0.326316,0.52438,421.053,6.85,0,0,0,1,no

```

Process 4. Training the Classifier Models

This process includes some subprocesses such as (i) determining the classifiers, (ii) balancing the training data, (iii) feature selection, and (iv) executing the classifiers as shown in the following figure.



In the first subprocess (e.g., determine classifiers), the supervised machine learning classifiers for the prediction task are determined. The selected classifiers are the most common ones such as Decision Tree, Random Forest, Bagging, k-NN. Table 4.2 shows the general names of the identified classifiers and their specific names at WEKA. It should be noted that all of them are interpretable models (i.e., people can easily understand the decisions made by the classifier). A summary of how the classifiers work and examples of each can be found in Appendix D.

In this case study, the MergeSplit benchmark dataset is used. Even though the name of the dataset suggests that it contains only merge and split events, all types of events are included. To decide whether to perform the second subprocess (e.g., balance

the training dataset), we first need to check whether the event distributions in the dataset are fair or not.

Table 4.2. Name of the Classifiers and their correspondents in WEKA

General Name	Correspondent Name in WEKA
Instance-based learner (K-nearest neighbors classifier, k-NN)	IBk
C4.5 Decision Tree	J48
Random Forest	Random Forest
Bootstrap Aggregating	Bagging
Random Tree	Random Tree

Table 4.3. Event frequencies according to the chain lengths of the benchmark dataset

Chain Length (L)	# of instances	Future Events					
		Grow	Shrink	Continue	Split	Merge	Dissolve
L=1	2130	663	501	392	269	162	143
L=2	1341	363	311	240	168	159	100
L=3	640	162	136	113	73	109	47
L=4	3	0	0	0	1	0	2

Table 4.3 shows the event frequencies as a function of the chain length of the dataset. The first column shows the possible chain lengths used to create the training datasets. The second column shows the number of instances that were used to train the classifiers. The last column "Future Events" lists the number of occurrences in the ground truth. As a reminder, the process of labeling the ground truth data is explained in Appendix B. As can be seen from the table, there is an imbalance in the distribution of the number of future events. To avoid the problem of overfitting (e.g., a classifier learns

better the future events "grow" that have the highest number of instances with respect to "dissolve"), it is necessary to balance the training datasets. For the second subprocess, we used the function WEKA class balancer. This balancer changes the weights of the events so that they have the same importance. As shown in the table, there are only three instances for training the classifiers when the chain length is four. Since there are few instances, there is no chance to train the classifiers to get accurate results.

The third subprocess (e.g., feature selection) is necessary because there may be irrelevant or correlated features in the data sets. The presence of irrelevant features and correlations between features may have some negative effects, such as a decrease in prediction success, a decrease in the performance of the training and execution process, and complex prediction models. By using feature selection methods in WEKA (e.g., wrapper subset evaluator over J48), these effects are eliminated. For different evolution chain lengths, different features are automatically selected by the wrapper method. The *wrapper* method performs learning-based feature selection and is a combination of a search method (for combining features to create different feature subsets) and attribute evaluator components (for evaluating the feature subset provided by the search method). In this study, the *BestFirst* method is used as the search method and J48 is used as the attribute evaluator.

Table 4.4. Selected features that gives the highest success score per future event

Event	Chain Length	# of SF	Selected Features (SF)
Grow	1	6	size, density, cc, cohesion, leadersEigen, event
Shrink	3	6	density, cc, cohesion, leasdersDegreeCentrality, LeadersEigen, event
Continue	3	6	C1 and C2 LeadersEigen, C3Size, C3LeadersDegreeCentrality, C3Event, Delta2LeadersCount
Split	1	7	size, density, cc, avgDegreeCentrality, leadersCount, LeadersEigen, event
Merge	3	4	C3Size, C3Density, C3AvgDegreeCentrality, C3event
Dissolve	3	6	C1Event, C2LeadersDegreeCentrality, C2LeadersEigen, C3Density, C3event, delta1LeadersEigen

Table 4.4 lists the selected features that have the highest predictive success per event. In the first column, "Event", the event values to be predicted are given. In the second column, the lengths of the evolution chains leading to this value are given. The third and final column lists the number of features selected and the features selected, respectively.

As the last subprocess, the classifiers on the dataset with selected features are executed.

Process 5. K-fold Cross Validation

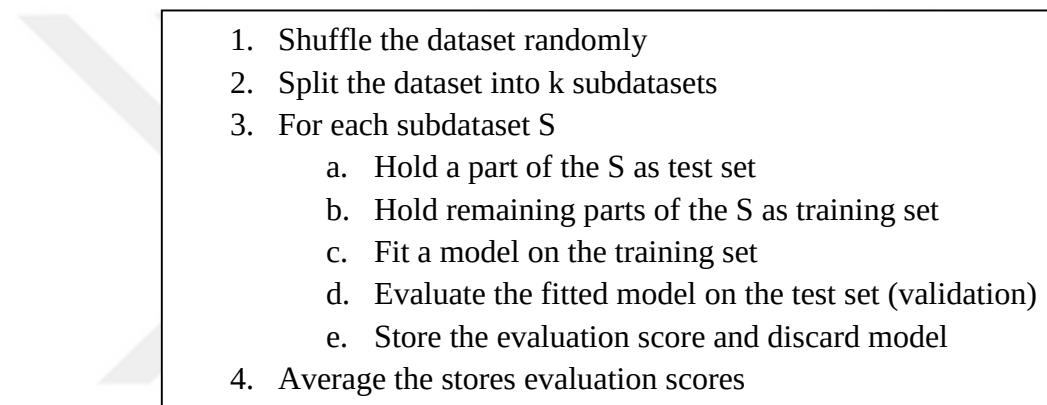


Figure 4.4. K-fold Cross Validation process

Doing training and testing processes on the same data is a methodological mistake. Since the classifiers already learn the labels of the instances in the data, they will perform perfectly in the data but fail on prediction on new instances unseen yet (overfitting). Holding a part of the dataset as a test set to avoid from overfitting is a way. However, the prediction scores of this method will not be very reliable because there is only one test set yet. **K-fold cross validation** provides a complete solution to this problem. It is a process that follows the procedure shown in Figure 4.4.

The k values are chosen as ten in this study because in the related works doing k-fold cross validation use k as ten. In Figure 4.5, as an example, the 5-fold cross-validation process is visualized. There is an assumption that the dataset is shuffled randomly. Then dataset is split into k (five) subsets. Then a subset is held as a test set (validation set) while

the other subsets are held as the training set. On the current training set, a classifier model is trained and tested the current test set. For k times, test and training subsets are hold, a model is trained and the model performance is measured. Finally, performance scores are averaged.

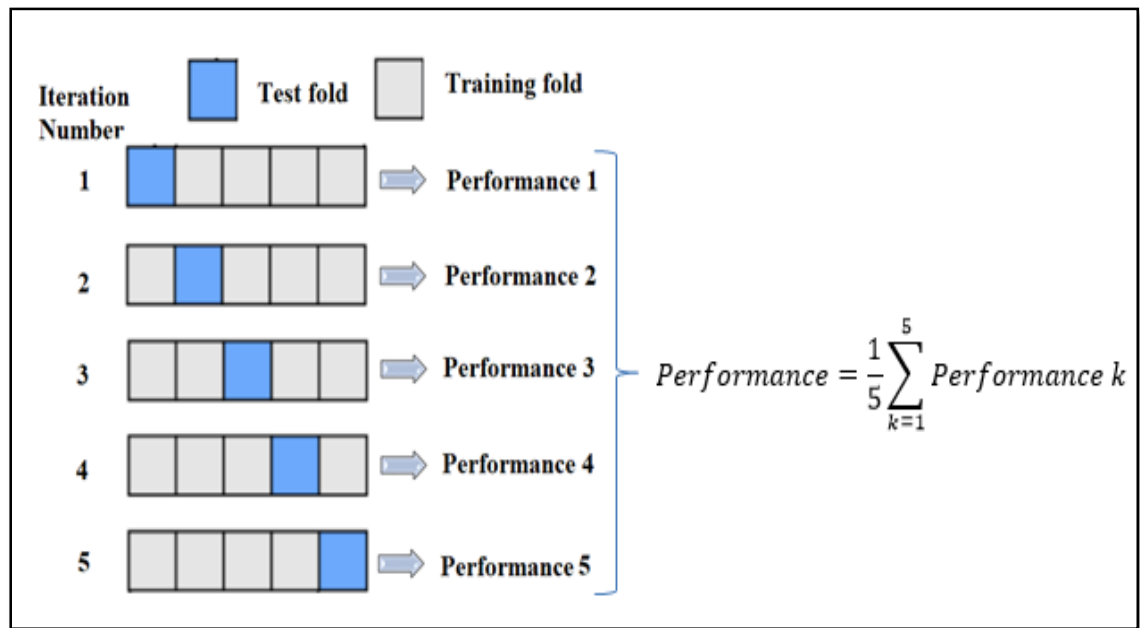


Figure 4.5. Visualization of an example of 5-fold Cross Validation

Prediction task inputs and outputs in WEKA

For the completeness of predictive analysis explanation, input and output of this analysis should be specified. In Figure 4.6, the input and output of the machine learning classifier for the prediction task in WEKA are illustrated. Input files contain training data. They are the files with the extension “. arff” and they include feature values and ground truth events for specified evolution chains. Classifiers such as Ibk, J48, Random forest, Bagging, and Random tree are the machine learning models to be used for prediction tasks in the context of this study. In general, the output of the prediction task is the labels per instance in training data.



Figure 4.6. Prediction task inputs and outputs in WEKA

```

@relation GreeneMergeSplitBenchmarkWithTREC
@attribute 'Cl_Size' numeric
@attribute 'Cl_Density' numeric
@attribute 'Cl_CC' numeric
@attribute 'Cl_Cohesion' numeric
@attribute 'Cl_AvgdegreeCentrality' numeric
@attribute 'Cl_LeadersCount' numeric
@attribute 'Cl_LeadersDegreeCentrality' numeric
@attribute 'Cl_LeadersEigen' numeric
@attribute 'Cl_event' numeric
@attribute 'merge' { yes, no}
@data
11,0.490909,0.364547,785.714,6.27273,0,0,0,1,no
60,0.136723,0.189552,85.9394,9.95,0,0,0,1,no
64,0.131944,0.192392,76.3814,10.3906,1,36,0.538995,2,no
56,0.148701,0.270003,89.3142,10.1964,1,36,0.538995,3,no
28,0.267196,0.424487,211.64,8.85714,0,0,0,1,no
5,0.9,0.7,3125,4.2,0,0,0,5,yes
12,0.469697,0.395058,861.244,5.75,0,0,0,1,no
5,0.9,0.461111,2083.33,5.4,0,0,0,3,no
10,0.622222,0.484234,680.272,7.7,0,0,0,5,yes
20,0.326316,0.52438,421.053,6.85,0,0,0,1,no
11,0.490909,0.404429,717.391,6.63636,0,0,0,5,yes
10,0.444444,0.397301,813.008,6.1,0,0,0,6,yes
20,0.326316,0.52438,421.053,6.85,0,0,0,1,no
12,0.590909,0.630733,606.061,7.75,0,0,0,5,yes
10,0.444444,0.397301,813.008,6.1,0,0,0,6,yes
11,0.563636,0.446402,687.5,7.18182,0,0,0,1,no
  
```

Figure 4.7. A sample training set for predicting merge event for WEKA

For illustration, a sample WEKA input (arff- Attribute Relation File Format) from merge evolution event as training set is provided in Figure 4.7. This file contains two main parts such header and data. In the header, the name of the relation and a list of attributes (the columns in the data) and their types. In the data part, the feature values of each tracked community are written as a separate line/vector. Just a kind reminder, the only attribute merge is the response variable. As is seen from the figure, features have values with different scales. Normalization is required before calculating Euclidean

distance when classifiers use this distance for the decision process. Otherwise, the features have large numbers dominates when measuring the distance.

```

=== Summary ===
Correctly Classified Instances      1910.5101      89.6953 %
Kappa statistic                    0.7939
Mean absolute error                 0.1038
Root mean squared error             0.3207
Relative absolute error             20.7642 %
Root relative squared error         64.1441 %
Total Number of Instances          2130

=== Detailed Accuracy By Class ===

```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,809	0,015	0,982	0,809	0,887	0,807	0,897	0,895	yes
	0,985	0,191	0,837	0,985	0,905	0,807	0,897	0,833	no
Weighted Avg.	0,897	0,103	0,910	0,897	0,896	0,807	0,897	0,864	

```

=== Confusion Matrix ===
  a    b    <-- classified as
861.2 203.8 |    a = yes
15.69 1049.31 |    b = no

```

Figure 4.8. A sample output for decision task in WEKA

However, the output of the process is seen as a little bit different in the figure because the output of the prediction process in WEKA is a text file including some statistics such as accuracy, errors made by the classifiers, and a confusion matrix showing True Positives, True Negatives, False Positives, and False Negatives. To be concise, a sample of produced output files by IbK in WEKA for merge evolution events is shown in Figure 4.8. The output file includes a detailed summary about the errors done by the classifier and a detailed accuracy information. The evaluation metric F-measure used to evaluate the success of prediction is explained in the next subsection.

4.4. Experimental Results

Several measures have been suggested to evaluate the classification performance. The most often used are accuracy, precision, recall, and F-measure for binary classifiers. The F-measure is the harmonic mean of precision and recall, which is calculated as in equation (4.1). It takes a value between 0 and 1 where the higher F-measure values show that predictions are more successfully done. Note that F-measure can be calculated because ground truth event information for each evolution is already known. The role of TREC in producing ground-truth dataset is explained in Appendix B.

$$F - measure = 2 \times \frac{precision \times recall}{precision + recall} \quad (4.1)$$

Table 4.5. The highest prediction success values in terms of F-measure per events

Classifiers	Events						Over all
	Grow	Shrink	Continue	Split	Merge	Dissolve	
Bagging	0.70	0.67	0.74	0.91	0.96	0.80	0.79
J48	0.70	0.68	0.76	0.92	0.97	0.82	0.80
Random Forest	0.72	0.65	0.71	0.96	0.98	0.78	0.78
Random Tree	0.70	0.62	0.60	0.94	0.96	0.76	0.76
IBk	0.70	0.62	0.64	0.94	0.96	0.77	0.76

Table 4.5 shows the classification performance for each particular evolution event provided by each classifier used in the experiment. In the first column, the classifiers used are listed. The second column (“Events”) shows the F-measures per event. The last column shows the overall prediction performance per the classifiers.

Table 4.6 shows the performance of the classifiers for all chain lengths. In the first column, the classifiers reside. The other columns list F-measure values according to the

chain lengths from one to three, respectively. Note that evaluation of these results are evaluated in the following subsection.

Table 4.6. Overall performance of the classifiers with respect to chain lengths

Classifiers	L=1	L=2	L=3
Bagging	0.79	0.77	0.77
J48	0.78	0.78	0.80
Random Forest	0.78	0.75	0.77
Random Tree	0.76	0.73	0.73
IBk	0.76	0.77	0.72

4.4.1. Discussion on the Results

According to the experimental results in Table 4.4, the performance of the classifiers is close to each other. However, the highest score is belonging to J48 and in the second-order Bagging classifier comes.

According to the results shown in Table 4.5, chain lengths make difference in prediction results, but there is no general pattern obeyed by all classifiers. For example, Bagging, Random Tree, and Ibk show an inverse relation with chain length on the performance while J48 shows a direct relation with chain length.

Table 4.7. summarizes the highest accuracy in F-measure at which chain length and by which classifier, which makes it easier to discuss the experimental results. As seen from the table, split and merge events are predicted with a great performance which is higher than 0.95. However, the lowest prediction success occurs for shrink events. Prediction performance of dissolve, continue, and grow events lies between prediction accuracy of merge and shrink events.

After examining summary Table 4.7, one question comes to mind as usual:” Why Random Forest and J48 performed better than other classifiers?” The answer lies in the data in our study because both classifiers are better to reduce the variance in the dataset

than the other classifiers. In our training dataset, some features used such as size, density and cohesion, and degree centrality of leaders have high variance.

Table 4.7. The highest success in terms of accuracy in percentage with specific chain length and classifier

	Success (F-measure)	Len	Classifier
Grow	0.73	1	RF
Shrink	0.68	3	J48
Continue	0.76	3	J48
Split	0.96	1	RF
Merge	0.98	3	RF
Dissolve	0.82	3	J48

Random Forests reduces variance in two ways: (i) training on different samples of the data and (ii) using a random subset of features. In the first way, the data are selected for decision tree construction with replacement. That is, each sample has an equal chance of being selected and can be selected more than once, which is called bagging. In our experiment, bagging with 100 iterations is used by default. In the second way, Random Forests uses a certain number of features in each decision tree. Suppose five features are selected and we have twenty features. Unfortunately, we left out fifteen features. However, the Random Forest is a collection of decision trees and five random features are used in each tree. If there are enough decision trees, all features are already used. In this study, the number of attributes is randomly examined by default. Therefore, the correlation between the created decision trees is low, resulting in low variance.

J48 reduces the variance in the following way. It aims to build the smallest possible decision tree. It uses information gains for branching. For this reason, it calculates the information gains for all features and then sorts them according to their gains. According to their ranks, they are placed in the decision tree. The information gain can be considered as inversely proportional in some sense in the decision tree domain. That is, the highest information gain implies the lowest variance. Therefore, J48 performs its branching by reducing the variance.

4.4.2. Concluding Thoughts

In summary, this chapter is devoted to investigating whether the TREC method can be extended to include a simple strategy for predicting the evolution of communities. That is, this chapter does not aim at novelty in the form of a new machine learning classifier and prediction method. The experimental results provide hope, and based on these results, it can be said that TREC has been extended for community evolution prediction. The most difficult to detect events such as merge and split events are predicted with very high performance. However, the success in predicting growing, shrinking, and continuing events can be increased by adding new structural or temporal features or by tuning the parameters of the classification models in future work. A friendly note: The parameters of the classifiers used are default values given by WEKA.

CHAPTER 5

CONCLUSION

The main objective of this study is to solve the problem of resource consumption in tracking community evolution, since resource consumption is as important as accuracy. However, this problem has not been addressed before our study. The most common solution to the tracking problem involves two main steps: (i) the independent detection of communities at each time step of the network, and (ii) the matching of detected communities over time. In studying these steps, it has been shown that the highest resource consumption occurs in community matching. Therefore, this research focuses on this part of the problem.

Therefore, our novel space-efficient method TREC (presented in Chapter 3) focused on solving the community matching problem. The key idea is to combine LSH and minhashing methods. Minhashing is in detail responsible for storing communities in much less memory by extracting their signatures. LSH is responsible for quickly finding similar communities of a queried community. With the guarantee that LSH definitely detects the similar communities with 40% or more. Moreover, a detailed banding analysis of LSH is performed in this chapter. Moreover, the effect of using minhash signatures and LSH is also evaluated.

Table 5.1 lists our goals and shows whether TREC achieves them. As can be seen from the figure, our novel TREC satisfies all goals according to our motivation. The functional goals are met by all competing methods, but they differ with respect to the performance goal. They have almost the same accuracy values. However, TREC has the lowest memory requirement. In terms of execution time consumption, TREC is the second best method and the execution time results are reasonable. In summary, TREC is the best method for evolution analysis on local computers due to its memory efficiency. To show the computational limitations of TREC, both space and time complexity analysis are performed. Then, the real-time analysis of memory usage and execution time is

measured using a diagnostic tool. It is found that the real-time analysis results are consistent with the theoretical results.

Table 5.1. Our goals of a method for evolution of communities tracking

	Goals	TREC
Functional Goals	Tracking both consecutively and nonconsecutively evolving communities	✓
	Identifying all evolution event types	✓
	Supporting k-community merge/split	✓
Performance Goals	Providing the highest accuracy	✓
	A reasonable execution time	✓
	The least memory consumption	✓
	Showing its computational boundaries	✓

As a complementary task, a case study is developed in Chapter 4. The study aims to answer the question whether the TREC method can be extended to the prediction of community development. Thus, it is not the aim of this chapter to present a new machine learning method or a method for predicting the evolution of a community. In order to predict the future of a community, it is necessary to know its evolution history. This is where the TREC method comes in. It tracks the evolution of communities and creates evolutionary chains. Then it is the turn of predictive analysis. For the analysis, a database application is developed that automatically determines the ground truth evolution events. Experimental results show that TREC can be used for this purpose.

The contributions of this dissertation is listed below.

- A novel, space-efficient TREC (TRacking Evolution of the Communities) method for tracking community evolution is proposed.
- A combination of LSH and minhashing techniques is proposed in community matching to solve the problem of high resource consumption.

- A complexity analysis of TREC is performed to evaluate its computational limitations.
- A dataset of evolution events is constructed to evaluate the success of prediction.

The research can continue, with some challenges still to be overcome, as follows.

- Reducing resource consumption in tracking evolution of communities can be tackled for a dynamic network with an overlapping community structure. If TREC is to be extended by researchers, the event detection rules and tracking algorithm to handle overlapping community structure need to be updated. Then the tracking algorithm needs to be updated.
- There are no publicly available benchmark datasets for evolution events to serve as a ground truth in prediction. Therefore, such benchmarks are a valuable contribution.
- Neither open source codes nor executables are available for the latest tracking methods (published between 2011 and 2021). If researchers want to conduct a comparative study or evaluate the success of their new method, they need to implement the relevant work. Therefore, it would be a valuable contribution for many researchers to make the implementations of such methods available as libraries and datasets, especially for comparative studies.

REFERENCES

Agarwal, Prerna, Verma, Richa, Agarwal, Ayush, and Chakraborty, Tanmoy. “*DyPerm: Maximizing Permanence for Dynamic Community Detection.*” Lecture Notes in Computer Science 10937 (2018). 437-449. https://doi.org/10.1007/978-3-319-93034-3_35

David, W. Aha, Kibler, Dennis, and Albert, Marc K. “*Instance-based learning algorithms.*” Machine Learning 6(1991). 37-66. <https://doi.org/10.1007/BF00153759>

Aktunc, Rıza, Toroslu, I. Hakkı, Ozer, Mert and Davulcu, Hasan. "A dynamic modularity based community detection algorithm for large-scale networks: DSLM." Presented at 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), Paris, France, Aug. 25-28, 2015. 1177-1183. <https://doi.org/10.1145/2808797.2808822>

Al-Sharoa, Esraa, Al-khassaweneh, Mahmood and Aviyente, Selin. "A tensor based framework for community detection in dynamic networks." 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), New Orleans, USA, Mar. 5-9, 2017. 2312-2316. <https://doi.org/10.1109/ICASSP.2017.7952569>

Arredondo, Raquel. “Relational Networking: Not Just Collecting Contacts.” <https://www.lebow.drexel.edu/news/relational-networking-not-just-collecting-contacts>. (accessed February 2, 2021)

Aston, Nathon and Hu, Wei. “*Community detection in dynamic social networks.*” Communications and Network 6,2 (2014). 124-136. <https://doi.org/10.4236/cn.2014.62015>

Asur, Sitaram, Parthasarathy, Srinivasan., Ucar, Duygu. “*An event-based framework for characterizing the evolutionary behavior of interaction graphs.*” ACM Transactions on Knowledge Discovery from Data 3,4 (2009). 1-36. <https://doi.org/10.1145/1631162.1631164>

Aynaud, Thomas and Guillaume, Jean-Loup. 2010a. "Static community detection algorithms for evolving networks." Presented at 8th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, Avignon, France, May 31- June 4, 2010. <https://ieeexplore.ieee.org/document/5520221> (accessed 2017)

Aynaoud, Thomas and Guillaume, Jean-Loup. 2010b. "Long range community detection." Presented at *Latin-American Workshop on Dynamic Networks*, Buenos Aires, Argentina, Nov. 4, 2010. <https://hal.inria.fr/inria-00531750/document> (accessed 2017)

Bhat, Sajid Y., and Abulaish, Muhammad. "*HOCTracker: Tracking the evolution of hierarchical and overlapping communities in dynamic social networks.*" *IEEE Transactions on Knowledge and Data engineering* 27, 4 (2015). 1019-1013. <https://doi.org/10.1109/TKDE.2014.2349918>

Blondel, Vincent D, Jean-Loup, Guillaume, Lambiotte, Renaud and Lefebvre, Etienne. "*Fast unfolding of communities in large networks.*" *Journal of Statistical Mechanics: Theory and Experiment* 2008, 10 (2008). P10008. <https://doi.org/10.1088/1742-5468/2008/10/P10008>

Bourqui, Romain, Gilbert, Frédéric, Simonetto, Paolo, Zaidi, Faraz, Sharan, Umang and Jourdan, Fabien. "Detecting structural changes and command hierarchies in dynamic social networks." Presented at *2009 International Conference on Advances in Social Network Analysis and Mining (ASONAM 2009)*, Athens, Greece, July 20-22, 2009. <https://doi.org/10.1109/ASONAM.2009.55>

Breiman, Leo. "*Random Forests.*" *Machine Learning* 45,1(2001). 5-32. <https://doi.org/10.1023/A:1010933404324>

Breiman, Leo. "*Bagging predictors.*" *Machine Learning* 24, 2(1996). 123-140. <https://doi.org/10.1007/BF00058655>

Bródka, Piotr, Kazienko, Przemysław, and Kołoszczyk, Bartosz. "Predicting group evolution in the social network." *Lecture Notes in Computer Science* 7710 (2012). 54-67. https://doi.org/10.1007/978-3-642-35386-4_5

Bródka, Piotr, Saganowski, Stanisław and Kazienko, Przemysław. "*GED: the method for group evolution discovery in social networks.*" *Social Network Analysis and Mining* 3, 1 (2012). 1-14. <https://doi.org/10.1007/s13278-012-0058-8>

Browet, Arnaud. "Community detection and Role extraction in Networks". *Complex Networks*. <http://www.complexnetworks.fr/wp-content/uploads/2014/09/ABROWET-LIP6.pdf> (accessed 2020)

Broder, Andrei Z. "On the resemblance and containment of documents." Presented at *Compression and Complexity of SEQUENCES 1997* (Cat. No. 97TB100171), Salerno, Italy, June 13, 1997. <https://doi.org/10.1109/SEQUEN.1997.666900>

Calderoni, Francesco, Brunetto, Domenico, and Piccardi, Carlo. "*Communities in criminal networks: A case study.*" *Social Networks* 48 (2017). 116-125. <https://doi.org/10.1016/j.socnet.2016.08.003>

Calvó-Armengol, Antoni, and Zenou, Yves. "*Social networks and crime decisions: The role of social structure in facilitating delinquent behavior.*" *International Economic Review* 45, 3 (2004). 939-958. <https://www.jstor.org/stable/3663642>

Cazabet, Remy, Amblard, Frederic, and Hanachi, Chihab. "Detection of overlapping communities in dynamical social networks." Presented at *2010 IEEE Second International Conference on Social Computing*, Minneapolis, USA, Aug. 20-22, 2010. <https://doi.org/10.1109/SocialCom.2010.51>

Cazabet, Rémy, and Rossetti, Giulio. "*Challenges in community discovery on temporal networks.*" *Temporal Network Theory* (2019). 181-197. https://doi.org/10.1007/978-3-030-23495-9_10

Cazabet, Rémy, Rossetti, Giulio and Amblard, Frédéric. "*Dynamic Community Detection.*" *Encyclopedia of Social Network Analysis and Mining* (2017). 1-10. <https://hal.archives-ouvertes.fr/hal-01665098/document>

Chakrabarti, Deepayan, Kumar, Ravi, and Tomkins, Andrew. "Evolutionary clustering." Presented at *the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Philadelphia, USA, Aug. 20-23, 2006. 554-560. <https://doi.org/10.1145/1150402.1150467>

Chakraborty, Tanmoy, Dalmia, Ayushi, Mukherjee, Animesh and Ganguly, Niloy. "*Metrics for community analysis: A survey.*" *ACM Computing Surveys (CSUR)* 50,4 (2017). 54. <https://doi.org/10.1145/3091106>

Chamberlain, B. Paul, Levy-Kramer, Josh, Humby, Clive, and Deisenroth, M.Peter. "*Real-time community detection in full social networks on a laptop.*" *PLOS ONE* 13,1 (2018). e0188702. <https://doi.org/10.1371/journal.pone.0188702>

Chen, Zhengzhang , Wilson, Kevin A., Jin, Ye, Hendrix,William, and Samatova, Nagiza F. "Detecting and tracking community dynamics in evolutionary networks." Presented at *IEEE International Conference on Data Mining Workshops (ICDM Workshops)*, Sydney, Australia, Dec. 13, 2010. <https://doi.org/10.1109/ICDMW.2010.32>

Chen, Jiyang, Zaïane, Osmar R and Goebel, Randy. "Local community identification in social networks." Presented at *International Conference on Advances in Social Networks*

Analysis and Mining (ASONAM 2009), Athens, Greece, July 20–22, 2009. <https://doi.org/10.1109/ASONAM.2009.14>

Cordeiro, Mário, Sarmiento, R. Portocarrero, and Gama, João. “*Dynamic community detection in evolving networks using locality modularity optimization.*” *Social Network Analysis and Mining* 6,1(2016). 15. <https://doi.org/10.1007/s13278-016-0325-1>

Dakiche, Narimene, Benbouzid-Si Tayeb, Fatima, Slimani, Yahya and Benatchba, Karima.2019a. “*Tracking community evolution in social networks: A survey.*” *Information Processing & Management* 56,3 (2019). 1084-1102. <https://doi.org/10.1016/j.ipm.2018.03.005>

Dakiche, Narimene, Benbouzid-Si Tayeb, Fatima, Benatchba, Karima, and Slimani,Yahya. “*Tailored Network Splitting for Community Evolution Prediction in Dynamic Social Networks.*” *New Generation Computing* 39(2021). 303–340. <https://doi.org/10.1007/s00354-021-00122-6>

Dakiche, Narimene, Benbouzid-Si Tayeb, Fatima, Slimani, Yahya and Benatchba, Karima. 2019b. “Community evolution prediction in dynamic social networks using community features' change rates.” Presented at *the 34th ACM/SIGAPP Symposium on Applied Computing(SAC'19)*, Limassol, Cyprus, Apr. 8-12, 2019. <https://doi.org/10.1145/3297280.3297484>

Danon, Leon, Diaz-Guilera, Albert, Duch, Jordi and Arenas, Alex. “*Comparing community structure identification.*” *Journal of Statistical Mechanics: Theory and Experiment* 2005,09 (2005). P09008. <https://doi.org/10.1088/1742-5468/2005/09/P09008>

Diakidis, Georgios, Karna, Despoina, Fasarakis-Hilliard, Dimitris, Vogiatzis, Dimitrios, and Paliouras, George. “Predicting the evolution of communities in social networks.” Presented at *the 5th International Conference on Web Intelligence, Mining and Semantics (WIMS' 15)*, Larnaca, Cyprus, July 13-15, 2015. <https://doi.org/10.1145/2797115.2797119>

Dinh, Thang, Xuan, Ying, and Thai, My T. "Towards social-aware routing in dynamic communication networks." Presented at *IEEE 28th International Performance Computing and Communications Conference*, Scottsdale, USA, Dec. 14-16, 2009. <https://doi.org/10.1109/PCCC.2009.5403845>

Fan, Wei, Yeung, Kai Hau Alan, and Wong, Kin Yeung. “*Assembly effect of groups in online social networks.*” *Physica A: Statistical Mechanics and its Applications* 392,5 (2013). 1090-1099.

<https://doi.org/10.1016/j.physa.2012.11.017>

Ferrara, Emilio, De Meo, Pasquale, Catanese, Salvatore and Fiumara, Giacomo. "Detecting criminal organizations in mobile phone networks." *Expert Systems with Applications* 41, 13 (2014). 5733-5750. <https://doi.org/10.1016/j.eswa.2014.03.024>

Folino, Francesco and Pizzuti, Clara. "*An evolutionary multiobjective approach for community discovery in dynamic networks.*" *IEEE Transactions on Knowledge and Data Engineering* 26, 8(2014). 1838-1852. <https://doi.org/10.1109/TKDE.2013.131>

Fortunato, Santo. "*Community detection in graphs.*" *Physics Reports* 486, 3-5 (2010). 75-174. <https://doi.org/10.1016/j.physrep.2009.11.002>

L. Gauvin, A. Panisson, and C. Cattuto, "*Detecting the community structure and activity patterns of temporal networks: a non-negative tensor factorization approach.*" *PloS one* 9,1 (2014). e86028. <https://doi.org/10.1371/journal.pone.0086028>

Gao, Wenhao, Luo, Wenijan and Bu, Chenyang. "Evolutionary community discovery in dynamic networks based on leader nodes." Presented at *2016 International Conference on Big Data and Smart Computing (BigComp)*, Hong Kong, China, Jan 18-20, 2016. <https://doi.org/10.1109/BIGCOMP.2016.7425801>

Girvan, Michelle, and Newman, Mark EJ. "*Community structure in social and biological networks.*" *Proceedings of the National Academy of Sciences* 99, 12 (2002). 7821-7826. <https://doi.org/10.1073/pnas.122653799>

Gliwa, Bogdan, Saganowski, Stanislaw, Zygmunt, Anna, Bródka, Piotr, Kazienko, Przemyslaw and Kozak, Jaroslaw. "Identification of group changes in blogosphere." Presented at *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM'12)*, Istanbul, Turkey, Aug. 26-29, 2012. <https://doi.org/10.1109/ASONAM.2012.207>

Greene, Derek, Doyle, Dónal, and Cunningham, Pádraig. "Tracking the evolution of communities in dynamic social networks." Presented at *the 2010 International Conference on Advances in Social Networks Analysis and Mining (ASONAM'10)*, Odense, Denmark, Aug. 9-11, 2010. <https://doi.org/10.1109/ASONAM.2010.17>

Guo, Chonghui, Wang, Jiajia and Zhang, Zhen. "Evolutionary community structure discovery in dynamic weighted networks," *Physica A: Statistical Mechanics and its Applications* 413(2014). 565-576. <https://doi.org/10.1016/j.physa.2014.07.004>

Hogg, Tad and Lerman, Kristina. “*Social Dynamics of Digg*”. EPJ Data Science 1,5 (2012). <https://doi.org/10.1140/epjds5>

Hopcroft, John, Khan, Omar, Kulis, Brian and Selman, Bart. “Tracking evolving communities in large linked networks.” *Proceedings of the National Academy of Sciences* 101,1(2004). <https://doi.org/10.1073/pnas.0307750100>

Ilhan, Nagehan, and Ögüdücü, Sule Gündüz. “Community event prediction in dynamic social networks.” *Presented at 12th International Conference on Machine Learning and Applications*. Miami, USA, Dec. 4-7,2013. <https://doi.org/10.1109/ICMLA.2013.40>

Indyk, Piotr, and Motwani, Rajeev. “*Approximate nearest neighbors: towards removing the curse of dimensionality*.” Presented at of the thirtieth annual ACM symposium on Theory of Computing (STOC’98), Texas, USA, May 1998. <https://doi.org/10.1145/276698.276876>

Jdidia, M. Ben, Robardet, Celine, and Fleury, Eric. “Communities detection and analysis of their dynamics in collaborative networks.” Presented at *2nd International Conference on Digital Information Management*, Lyon, France, Oct. 28-31, 2007. <https://doi.org/10.1109/ICDIM.2007.4444313>

Karataş, Arzum, and Serap Şahin. 2018a. “Application areas of community detection: A review.” Paper presented at *2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)*, Ankara, Turkey, Dec. 3-4, 2018. <https://doi.org/10.1109/IBIGDELFT.2018.8625349>.

Karataş, Arzum, and Serap Şahin. 2018b. “A comparative study of modularity-based community detection methods for online social networks.” Paper presented at the 12th Turkish National Software Engineering Symposium (UYMS 2018), Istanbul, Turkey, Sep. 10-12, 2018

Kempe, David, Kleinberg, Jon, and Tardos Éva. “*Maximizing the spread of influence through a social network*.” Paper presented at the *ninth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD’03)*, Washington D.C., USA, August 24-27, 2003. <https://doi.org/10.1145/956750.956769>

Kullback,S.and Leibler, R.A. “*On Information and Sufficiency*.” *The Annals of Mathematical Statistics* 22, 1 (1951). 79–86. <https://doi.org/10.1214/aoms/1177729694>

Lafferty, John D., McCallum, Andrew K., and Pereira, Fernando C.R. “Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data”.

Presented at *Eighteenth International Conference on Machine Learning(ICML'01)*, Williamstown, USA, June 28 - July 1, 2001. <http://dl.acm.org/citation.cfm?id=645530.655813>

Lancichinetti, Andrea, Fortunato, Santo, Kertész, János. “*Detecting the overlapping and hierarchical community structure in complex networks.*” *New Journal of Physics* 11, March 2009 (2009). 033015. <https://doi.org/10.1088/1367-2630/11/3/033015>

Lerman, K.: Digg 2009 Dataset. <https://www.isi.edu/~lerman/downloads/digg2009.html> (2012). Accessed 8 October 2020

Leskovec, Jure, Kleinberg, John, Faloutsos, Christos. “Graphs over time: densification laws, shrinking diameters and possible explanations.” Presented at *the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '05)*, Chicago, Illinois, USA, Aug. 21-24, 2005. <https://doi.org/10.1145/1081870.1081893>

Leskovec, Jure, Rajaraman, Anand and Ullman, Jeffrey D. “Finding Similar Items.” In *Mining Of Massive Datasets*. 2015. Accessed 2017. <http://infolab.stanford.edu/~ullman/mmds/bookL.pdf>

Liben-Nowell, David and Kleinberg, Jon. “*The link-prediction problem for social networks.*” *Journal of the Association for Information Science and Technology* 58,7 (2007). 1019-1031, 2007. <https://doi.org/10.1145/956863.956972>

Lin, Yu-Ru, Chi, Yun, S. Zhu, Shenghuo, Sundaram, Hari, and Tseng, Belle L. “*Analyzing communities and their evolutions in dynamic social networks.*” *ACM Transactions on Knowledge Discovery from Data (TKDD)* 3,2 (2009). 1-31. <https://doi.org/10.1145/1514888.1514891>

Ma, Xiaoke and Dong, Di. “*Evolutionary nonnegative matrix factorization algorithms for community detection in dynamic networks.*” *IEEE transactions on knowledge and data engineering* 29,5 (2017). 1045-1058. <https://doi.org/10.1109/TKDE.2017.2657752>

Mitra, Bivas, Tabourier, Lionel, and Roth, Camille. “*Intrinsically dynamic network communities.*” *Computer Networks* 56,3(2012). 1041-1053. <https://doi.org/10.1016/j.comnet.2011.10.024>

Mohammadmosaferi, K. Kadkhoda, Naderi, Hassan. "*Evolution of communities in dynamic social networks: An efficient map-based approach.*" Expert Systems with Applications 147, (2020). 113221. <https://doi.org/10.1016/j.eswa.2020.113221>

Mucha, Peter J., Richardson, Thomas, Macon, Kevin, Porter, Mason A., Onnela, Jukka-Pekka. "*Community structure in time-dependent, multiscale, and multiplex networks,*" Science 328,5980 (2010). 876-878. <https://doi.org/10.1126/science.1184819>

Newman, Mark EJ. "*Fast algorithm for detecting community structure in networks.*" Physical Review E 69, 6 (2004). 066133. <https://doi.org/10.1103/PhysRevE.69.066133>

Nguyen, Nam P., Dinh, Thang N., Xuan, Ying, and My T. Thai. "Adaptive algorithms for detecting community structure in dynamic social networks." Represented at 2010 IEEE Annual Joint Conference INFOCOM, Shanghai, China, Apr. 10-15, 2010. <https://doi.org/10.1109/INFCOM.2011.5935045>

Nguyen, Nam P., Dinh, Thang N., Xuan, Ying, and My T. Thai. "Overlapping communities in dynamic networks: their detection and mobile applications." Presented at the 17th annual international conference on Mobile computing and networking, Las Vegas, USA, Sep. 19-23, 2011, <https://doi.org/10.1145/2030613.2030624>

Palla, Gergely, Derényi, Imre, Farkas, Illés and Vicsek, Tamás. "*Uncovering the overlapping community structure of complex networks in nature and society.*" Nature 43,7043 (2005). 814-818. <https://doi.org/10.1038/nature03607>

Pavlopoulou, Maria. E. G., Tzortzis, Grigorios, Vogiatzis, Dimitrios, and Paliouras, George. "Predicting the evolution of communities in social networks using structural and temporal features." Presented at 12th International Workshop on Semantic and Social Media Adaptation and Personalization (SMAP), Bratislava, Slovakia, July 9-10, 2017. <https://doi.org/10.1109/SMAP.2017.8022665>

Pons, Pascal and Latapy, Matthieu. "*Computing communities in large networks using random walks.*" Lecture Notes in Computer Science 3733(2005). 284-293. https://doi.org/10.1007/11569596_31

Quinlan, Ross. C4.5: programs for machine learning. San Mateo, CA: Morgan Kaufmann Publishers,1993.

Rossetti, Giulio, and Cazabet Rémy. "*Community discovery in dynamic networks: a survey.*" ACM Computing Surveys (CSUR) 51, 2 (2018). 1-37. <https://doi.org/10.1145/3172867>

Rossetti, Giulio, Pappalardo, Luca, Pedreschi, Dino and Giannotti, Fosca. “*Tiles: an online algorithm for community discovery in dynamic social networks.*” *Machine Learning* 106, 8(2017). 1213-1241. <https://doi.org/10.1007/s10994-016-5582-8>

Rosvall, Martin, Bergstrom, Carl T. “Maps of random walks on complex networks reveal community structure.” *Proceedings of the National Academy of Sciences* 105,4 (2008). 1118-1123. <https://doi.org/10.1073/pnas.0706851105>

Saganowski, Stanislaw, Gliwa, Bogdan, Bródka, Piotr, Zygmunt, Anna, Kazienko, Przemyslaw, and Koźlak, Jaroslaw. “*Predicting community evolution in social networks.*” *Entropy* 17,5 (2015). 3053-3096. <https://doi.org/10.3390/e17053053>

Saganowski, Stanislaw, Bródka, Piotr and Kazienko, Przemysław. “*Community Evolution*” *Encyclopedia of Social Network Analysis and Mining* 2017 (2017). 1-14. https://doi.org/10.1007/978-1-4939-7131-2_22

Sun, Yang, Tang, Junhua, Pan, Li and Li, Jianhua. "Matrix Based Community Evolution Events Detection in Online Social Networks." Presented at *IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, Chengdu, China, Dec 19-21, 2015. <https://doi.org/10.1109/SmartCity.2015.114>

Tajeuna, E. Gael, Bouguessa, Mohamed and Wang, Shengrui. “Tracking communities over time in dynamic social network.” Presented at *the 12th International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM 2016)*, New York, USA, July 16-21, 2016. https://doi.org/10.1007/978-3-319-41920-6_25

Takaffoli, Mansoureh, Fagnan, Justin, Sangi, Farzad and Zaïane, Osmar R. ”Tracking changes in dynamic information networks.” Presented at the *2011 International Conference on Computational Aspects of Social Networks (CASoN 2011)*, Salamanca, Spain, Oct. 19-21. <https://doi.org/10.1109/CASON.2011.6085925>

Takaffoli, Mansoureh, Rabbany, Reihaneh, and Zaïane, Osmar. R. “Community evolution prediction in dynamic social networks.” At presented *the 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*, Beijing, China, Aug. 17-20, 2014. <https://doi.org/10.1109/ASONAM.2014.6921553>

Takaffoli, Mansoureh, Sangi, Farzad, Fagnan, Justin and Zaïane, Osmar. R. “MODEC — Modeling and Detecting Evolutions of Communities.” Presented at *AAAI Conference on Web and Social Media(ICWSM 2011)*, Catalonia, Spain, July 17-21, 2011. <http://www.aaai.org/ocs/index.php/ICWSM/ICWSM11/paper/view/2853>

Tang, Jie, Zhang, Jing, Yao, Limin, Li, Juanzi, Zhang, Li and Su, Zhong. "Arnetminer: extraction and mining of academic social networks." Presented at *the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '08)*, Las Vegas, Nevada, USA, Aug. 24-27, 2008. <https://doi.org/10.1145/1401890.1402008>

Traag, Vincent A, Waltman, Ludo, and Jan van Eck, Nees. "*From Louvain to Leiden: guaranteeing well-connected communities.*" *Scientific Reports* 9, 1 (2019). 1-12. <https://doi.org/10.1038/s41598-019-41695-z>

Waltman, Ludo, and van Eck, N. Jan. "A smart local moving algorithm for large-scale modularity-based community detection". Welcome to the homepage of Ludo Waltman. <http://www.ludowaltman.nl/slm/> (accessed 2018).

Wang, Quinna and Fleury, Eric. "Mining time-dependent communities." Presented at *Latin-American Workshop on Dynamic Networks*, Buenos Aires, Argentina, Nov. 4, 2010. https://hal.inria.fr/inria-00531735/file/lawdn2010_submission_1.pdf (accessed 2017)

Wang Yi, Wu Bin and Du, Nan. "Community Evolution of Social Network: Feature, Algorithm and Model.", 2008. <https://arxiv.org/abs/0804.4356> (accessed 2017)

Van Dongen, Stijn Marinus. 2000. "Graph clustering by flow simulation.", PhD Dissertation, Utrecht University. <http://dspace.library.uu.nl/bitstream/handle/1874/848/full.pdf?sequence=1&isAllowed=y>. Accessed 20 June 2018.

Yang, Zhao, Algesheimer, René, Tessone, Claudio J. "*A comparative analysis of community detection algorithms on artificial networks.*" *Scientific Reports* 6 (2016). 30750. <https://doi.org/10.1038/srep30750>

Yelp Inc. "Yelp Open Dataset An all-purpose dataset for learning." <https://www.yelp.com/dataset/>. Accessed 12 May 2019

Zanin, Massimiliano, Cano, Pedro, Buldú Javier M, and Celma, Oscar. "*Complex networks in recommendation systems.*" Paper presented at the 2nd WSEAS International Conference on Computer Engineering and Applications: World Scientific Advanced Series In Electrical And Computer Engineering. Acapulco, Mexico, Jan. 25-27, 2008. <https://dl.acm.org/doi/10.5555/1373936.1373955>

Zhu, Zhichao, Cao, Guohong, Zhu, Sencun, Ranjan, Supranamaya, and Nucci, Antonio. "*A social network based patching scheme for worm containment in cellular networks.*"

Handbook of Optimization in Complex Networks (2012). 505-533.
https://link.springer.com/chapter/10.1007/978-1-4614-0857-4_17



APPENDIX A

A TAXONOMY OF METHODS FOR TRACKING EVOLUTION OF COMMUNITIES

In this section, a summary for each method for tracking evolution of communities by framing them in the Dakiche et al.'s taxonomy (Dakiche, Tayeb, Slimani and Benatchba 2019a). The taxonomy is a three-level classification of existing methods for tracking community evolution in dynamic social networks with respect to their network models (first level of tree structure), their functioning principles (second level) and algorithmic techniques (third level). The chart of taxonomy in 3.10 is repeated just for the sake of completeness of the section.

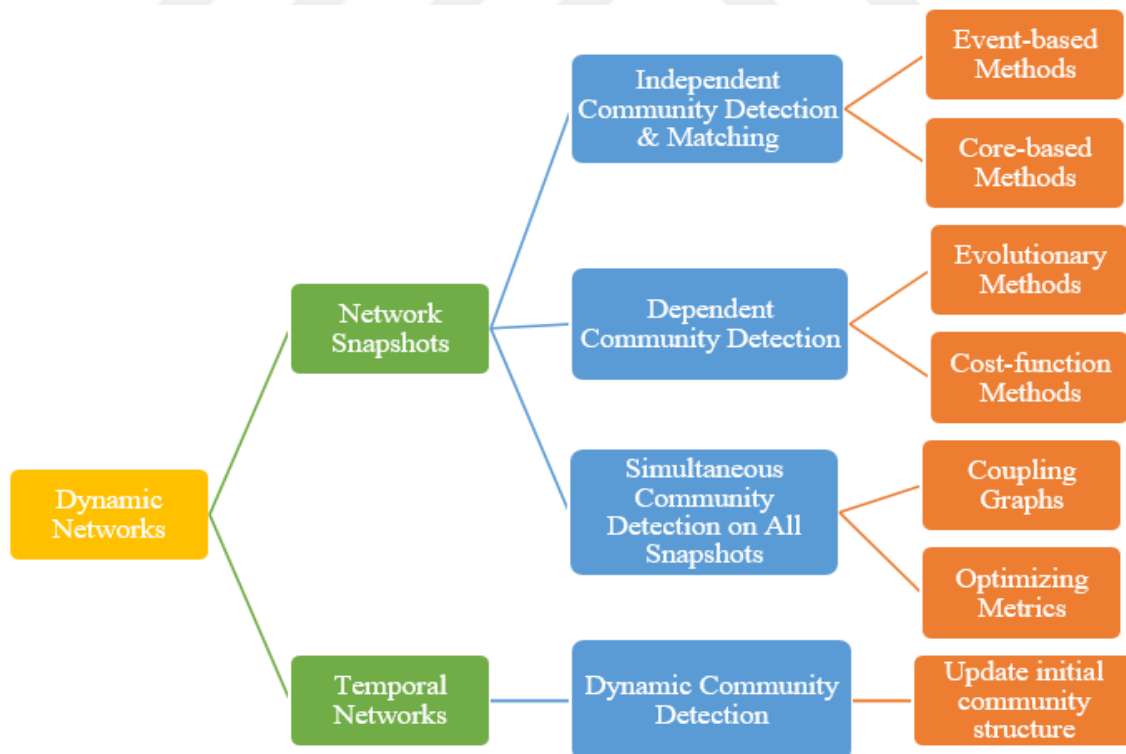


Figure 3.10. Classification chart of the existing tracking community evolution methods in dynamic social networks

The taxonomy divides methods for tracking evolution of communities into four categories according to their methodological principles; the methods using (i) independent community detection and matching approach, (ii) dependent community detection; (iii) simultaneous community detection and (iv) dynamic community detection on time-dependent networks.

I) The Methods Using Independent Community Detection and Matching Approach

Any method for tracking evolution of communities adopted this approach contains two-stage. In first stage, the method takes a dynamic network with some time steps as seen in Figure A.1(a) and detect community structures on all time steps independently (in Figure A.1(b)). In the second stage, communities found on successive time steps are matched as seen in Figure A.1. (c)-(d). Evolution of communities is seen in Figure A.1. (e). Our novel method TREC can be categorized under this type. Therefore, the methods under this category is investigated in Section 3.2.

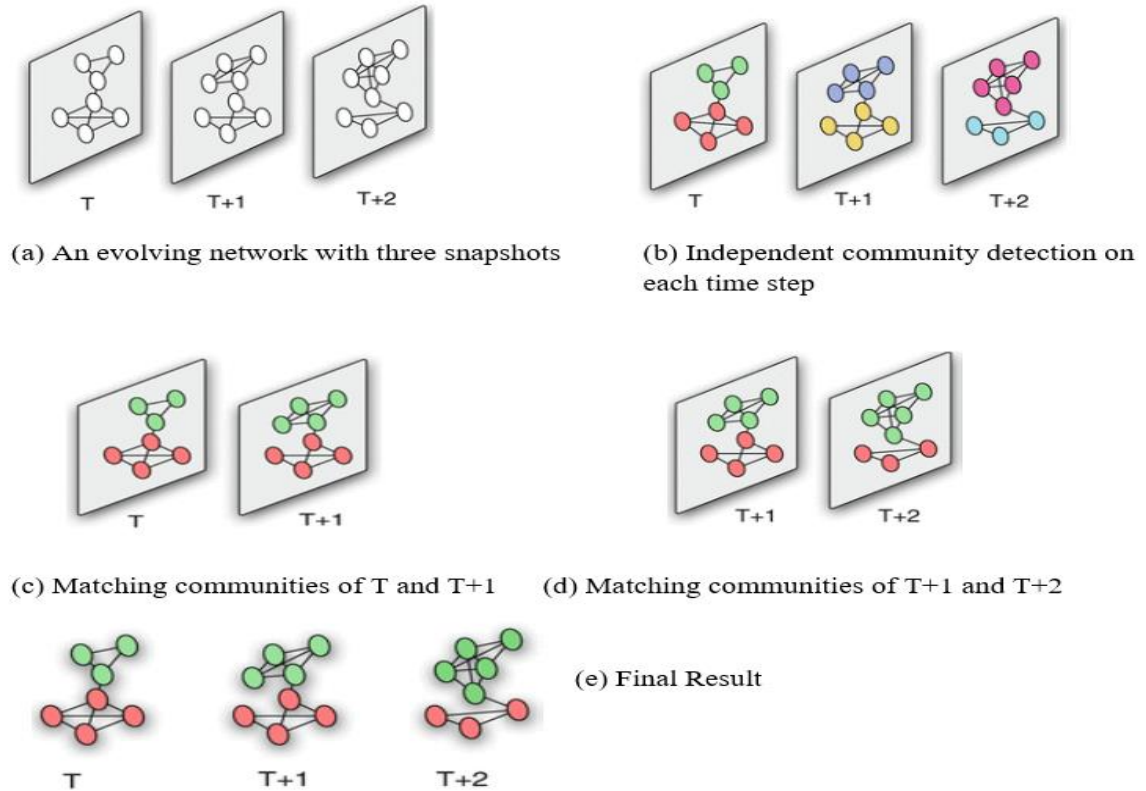


Figure A.1. Independent community detection and matching approach (Source: Cazabet, Rossetti and Amblard 2017)

Hopcroft et al.'s work (Hopcroft, Khan, Kulis and Selman 2004) can be regarded as one of the pioneer work for event-based methods. It first detects implicit communities. It makes run a community detection algorithm for each time step for several times, then takes into consideration most appeared communities during detection. Then it does agglomerative hierarchical clustering between successive snapshots via their matching function.

Palla et al. (Palla, Derényi, Farkas and Vicsek 2005) regard communities as composed of cliques. They use Clique Percolation Method (CPM) to detect communities. Then, they build joint graphs for each two sequential time steps and apply CPM again. They use a correlation function and a stationary parameter to find overlap between two states of a community and to denote average correlation of community states, respectively.

Bourqui et al.'s program (Bourqui, Gilbert, Simonetto, Zaidi, Sharan and Jourdan 2009) detects major structural changes over time via an overlapping community detection algorithm. It identifies community events by using community overlaps. Then, by using Minimum Spanning Tree (MST) to reveals hierarchies between communities.

Asur et al. (Asur, Parthasarathy and Ucar 2009) proposed a simple method to identify community events via matching approach on time step community membership matrices. They use MCL (Van Dongen 2000), a modularity-based algorithm, to detect community structures.

Greene et al. (Greene, Doyle and Cunningham 2010) propose a model for evolution of communities over time. They obtain community structures on each time step as step communities. They represent each dynamic community by a timeline of its constituent step communities. The most recent observation in a timeline is referred to as the front of the dynamic community. After authors make matching between communities via Jaccard similarity, they update front elements and timelines for each dynamic community.

Brodka et al. (Bródka, Saganowski and Kazienko 2012) develop Group Evolution Discovery (GED) method. Heart of this method is inclusion measure, allows to evaluate the inclusion of one community in another one, which combines both group quantity (what number of members of first group in second group) and quality (all member importance in the group) of community members. Based on inclusion measure, they put some rules to determine community events.

Gliwa et al. (Gliwa, Saganowski, Zygmunt, Bródka, Kazienko and Kozak 2012) propose another method so-called Stable Group Changes Identification(SCGI). They used CPM method (Palla, Derényi, Farkas and Vicsek 2005) for community detection and they propose modified Jaccard similarity instead of inclusion metric. They assume that two communities are similar if similarity is greater than 50%.

Takaffoli et al. (Takaffoli, Fagnan, Sangi and Zaïane 2011) develop a method for improve the tracking evolution of communities by defining rules. They do matching operations communities not only between sequential time steps but also among different time steps by using a simple greedy matching algorithm.

Sun et al. (Sun, Tang, Pan and Li 2015) focus on event detection on online social networks on successive snapshots. They find communities using Louvain algorithm (Blondel, Guillaume, Lambiotte and Lefebvre 2008) on each time step. Then, they create a correlation matrix to see relations between communities at time T and $T + 1$. According to this matrix, they define some evolution rules for community events.

Tajeuna et al. (Tajeuna, Bouguessa and Wang 2016) propose a new method to model and track community evolution. They first independently identify community structure in each time step. For each detected community, they find the number of shared members with all other communities. Then, they estimate a representative vector that includes shared members with remaining communities for each detected community. Next, compare representative vectors by using a new similarity measure so-called mutual transition. Via this measure, they create rules to capture community events.

Mohammadmosaferi and Naderi (Mohammadmosaferi and Naderi 2020) propose a novel method ICEM (Identification of Community Evolution by Mapping) for tracking evolution of communities. ICEM can identify both consecutive and nonconsecutive evolutions of communities. It basically determines the evolution by tracking members of the communities within a global hash map. It maps each member to a $\langle \text{time}, \text{community} \rangle$ pair where *time* represents the last observed time step and *community* represents the last observed community. Additionally, it keeps similarity of each community at a specific time step. By using both similarity lists and the hash map, it determines the evolutions.

As for, core-based methods in this approach, they evaluate community evolution over time based on core (special) nodes. Wang et al. (Wang, Wu and Du 2008) proposed CommTracker software. They determine core nodes according to their centrality values

in the network. The higher centrality value is better nominee to be a core node. The mapping of communities is done according to the mutual core nodes between pairs of communities at different snapshot networks. Finally, the tracking of communities is done by tracking the core nodes.

Another core-based method is proposed by Chen et al. (Chen, Wilson, Jin, Hendrix and Samatova 2010). They define communities as some maximal cliques of network. They introduce Graph representatives (e.g., common members of two selected graphs) and Community representatives (e.g., common members of same community on all snapshot networks) concepts to prune search space. They first find graph representatives to avoid redundant communities and detect communities. Then, for each community selects a member as community representative. Next, they use community representatives to establish a relationship between communities of different snapshots. Finally, they apply some decision rules to determine community events.

II) Dependent Community Detection

The methods adopted this approach process iteratively evolution of the network with two steps: bootstrap and update. Both evolutionary and cost-based methods, in first step (bootstrap), take an evolving network with some snapshots as seen in Figure A.2. (a) and detect community structures on first time step (in Figure A.2. (b)). However, they adopt different ways in the second step (update). Evolutionary methods found successive communities by using community structure or core nodes are found previous step and successive time step network while cost-based methods try to do minimum modification on communities on two consecutive snapshots. Update step is seen in Figure A.2. (c)-(d). Evolution of communities is seen in Figure A.2. (e).

The methods existing in the literature using dependent community detection is below.

Dinh et al.'s method (Dinh, Xuan and Thai 2009) is based on Clauset Newman Moore (CNM) community detection algorithm (Clauset, Newman and Moore 2004). Authors regard last snapshot communities as initial state and put each newly incoming member into a singleton community. After, CNM is re-run to obtain the new communities at the current snapshot network.

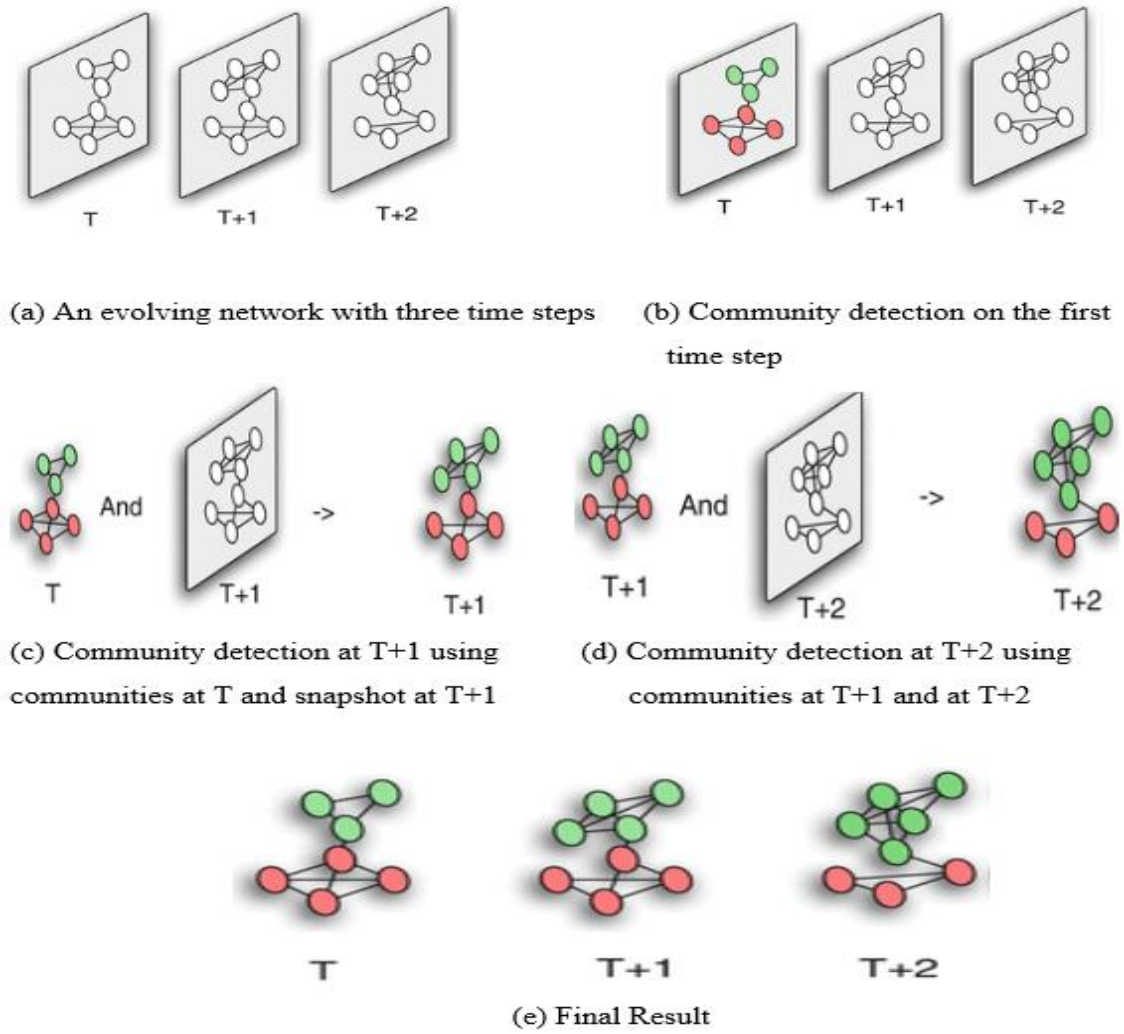


Figure A.2. Dependent community detection approach (Source: Cazabet, Rossetti and Amblard 2017)

Wang and Fleury (Wang and Fleury 2010) initialize Louvain algorithm (Blondel, Guillaume, Lambiotte and Lefebvre 2008) with core nodes found previous snapshot network. They define core nodes as survival members of the same community after some re-runs of a community detection algorithm. After, they re-run Louvain to obtain the new communities at the current snapshot network.

Similar to Wang and Fleury's work (Wang and Fleury 2010), Aynaud and Guillaume (Aynaud and Guillaume 2010a) propose a method based on Louvain algorithm (Blondel, Guillaume, Lambiotte and Lefebvre 2008). They propose initialize algorithm at time T+1 with the communities found at time T. In addition, they add a parameter to avoid community drift (e.g. to evolve invalid communities).

Guo et al. (Guo, Wang and Zhang 2014) detect communities via modularity optimization for dynamic weighted network snapshots. For each snapshot, they build an input matrix, represents a trade-off between adjacency matrices of current and previous snapshots. Then, they apply a modified modularity optimization community detection algorithm.

Aston and Hu (Aston and Hu 2014) propose Dynamic Structural Clustering Algorithm for Networks (DSCAN), which is an improved version of SCAN (Xu, Yuruk, Feng and Schweiger 2007) algorithm that works for static networks. DSCAN performs SCAN on the first time step. Then for all consecutive time steps, it obtains the difference in edges between the two time steps. If there are edge changes, then the network is updated from the nodes of each edge change of the networks. During the update of edges, a node can become core, or a node can no longer be a core node. When a change in the network is detected and needs to be updated, an existing cluster id or a new cluster id is propagated through all structurally connected nodes to form a new community.

Aktunc et al. (Aktunc, Toroslu, Ozer and Davulcu 2015) propose Dynamic Smart Local Moving (dSLM) algorithm for dynamic networks, which is an improvement of SLM (Waltman and Van Eck 2013) algorithm that works for static networks. They regard dynamicity in the form of edge insertions and deletions. dSLM performs SLM on first time step. Then for all consecutive time steps, it obtains the difference in edges between the two time steps. If there is new-coming node, dSLM needs to try only one node movement which is to move newly added node from its singleton community to its only neighbor community. Since it appears to increase the modularity of the network, dSLM places the new node to another community. Because the initial community structure is known to be the one that maximizes the modularity of the network, there is no other node movement trying that can increase modularity. By this way, the dSLM runs faster than SLM.

Gao et al. (Gao, Luo and Bu 2016) propose an evolutionary algorithm for community evolution based on leader nodes. Leader members are an adopted concept from Khorasgani et al.'s work (Khorasgani, Chen and Zaiane 2010). A leader member is the most central member her community. Each community is regarded as a union of a leader member and set of followers that come together and close to their leaders. Authors' algorithm builds communities from the leader members to track communities.

Some works (Gauvin, Panisson and Cattuto 2014; Ma and Dong 2017; Al-Sharoa, Al-khassaweneh and Aviyente 2017) represent temporal relations or activities among users as a 3-way tensor (e.g., an adjacency matrix and time dimension), and they examined to community evolution with non-negative factorization the tensors in recent years. Some of them use cost functions as well.

As for cost-based methods, Chakrabarti et al.'s work (Chakrabarti, Kumar and Tomkins 2006) is one of the pioneer works introduce community evolution over time. Authors introduce snapshot quality and history cost which quantifies the accuracy of partitioning of the current snapshot and quantifies the difference between partitioning on current time step and previous time step correspondingly. Their method calculates snapshot and cost quality for each time step, then choose the partition is the one with high snapshot and low history cost.

Lin et al. (Lin, Chi, Zhu, Sundaram and Tseng 2009) introduce FacetNet software that allows multi-membership (e.g. belonging to more than one communities) property for the nodes for a specific time step. Unlike Chakrabarti et al.'s method (Chakrabarti, Kumar and Tomkins 2006), they introduce snapshot cost instead of snapshot quality to quantify the accuracy of the current snapshot. They use Kullback-Leibler method (Kullback and Leibler 1951) to count for snapshot cost and history cost. Unfortunately, FacetNet says nothing much what happens in group level. Rather, it says what happen to specific nodes. Therefore, a user need to analyze the result of the program and assign community event on by own.

Another cost-based algorithm is DYNMOGA, a genetic algorithm to optimize multi-objective quality function, proposed by Folino and Pizzuti (Folino and Pizzuti 2014). The authors aim to maximize partition quality for current snapshot via modularity maximization while maximize Normalized Mutual Information (NMI) (Danon, Diaz-Guilera, Duch and Arenas 2005) between community structure of the current time step and previous time step to ensure a smooth evolution of communities.

III) Simultaneous Community Detection on All Snapshots

The methods adopted this approach aim to detect community structure on all time steps given. In this category, methods are divided into two sub groups: methods based on coupling graphs and methods based on optimization metric. Both methods take an evolving network with some time steps as seen in Figure A.3.(a) as an input. Coupling

graph-based methods build a unique network by binding all time steps in a manner that keeps community structures aligned over time. Then, runs a single community detection algorithm on this new network. On the other hand, optimization metric-based methods build metrics that can be optimized on all time steps over time. Detection of communities relevant on all time steps is seen in Figure A.3.(b). Finally, evolution of communities is seen in Figure A.3. (c).

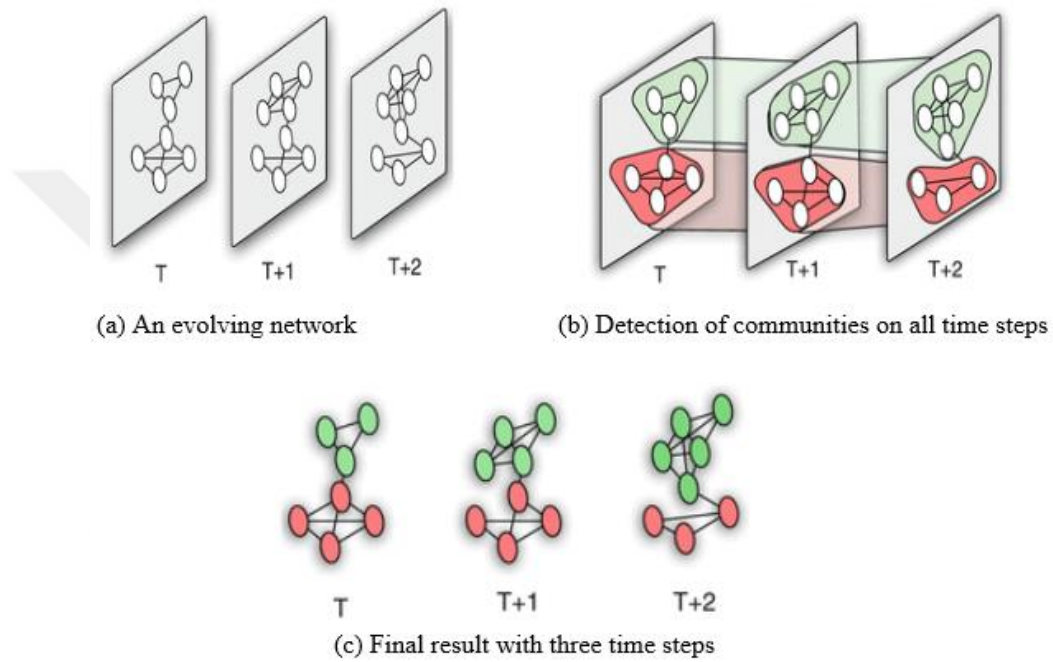


Figure A.3. Simultaneous Community Detection approach (Source: Cazabet, Rossetti and Amblard 2017)

Jdidia et al. (Jdidia, Robardet and Fleury 2007) propose to add edges between nodes in successive time steps to obtain a coupling graph. They define two types of edges; identity edges and transversal edges. Identity edges are occurred between same nodes in different snapshots whereas transversal edges connect the nodes at least one common neighbor in consequent snapshots. After building joint graph, authors of the work run Walktrap, a static community detection algorithm, (Pons and Latapy 2005) to trace group membership over time.

Similarly, Mucha et al. (Mucha, Richardson, Macon, Porter and Onnela 2010) add links between same nodes in different snapshots, and run a generalized version of Louvain algorithm (Blondel, Guillaume, Lambiotte and Lefebvre 2008) to optimize modularity metric.

Additionally, Mitra et al. (Mitra, Tabourier and Roth 2012) work on specific networks composed of links like citation networks. They show that it is possible that run a community detection algorithm for static networks for these kinds of networks. They use Louvain algorithm (Blondel, Guillaume, Lambiotte and Lefebvre 2008) to detect community structure due to its speed and being scalable.

As for optimizing metric methods, Aynaud and Guillaume et al. (Aynaud and Guillaume 2010b) use a modified version of Louvain algorithm to detect community structures. They optimize average modularity of group of nodes over several time steps.

IV) Dynamic Community Detection on Temporal Networks

The methods in this category work on temporal networks and aim to detect community structure in an online manner by following series of modifications (addition or deletion) of members(nodes) or relations (edges) for each time step. The methods take an evolving network with an initial time step and two modifications as seen in Figure A.4. (a) as an input. First, they determine initial community structure on the first snapshot as seen in Figure A.4 (b)). They do modifications sequentially on the community structure for each incoming update as seen in Figure A.4. (c)-(d). Finally, evolution of communities is seen in Figure A.4. (e).

Cazabet et al. (Cazabet, Amblard and Hanachi 2010) propose iLCD (intrinsic Longitudinal Community Detection) algorithm. iLCD modifies community structure by regarding path lengths and its robust second neighbors. It can identify overlapping communities as well. However, it considers only addition of members or relations not removals of them.

Nguyen et al. (Nguyen, Dinh, Xuan, Thai 2011) propose QCA (Quick Community Adaptation) method that contains update strategies. QCA method is based on modularity maximization by assigning a “force”- attracts a node towards to a community. It finds initial community structure using Louvain algorithm (Blondel, Guillaume, Lambiotte and Lefebvre 2008).

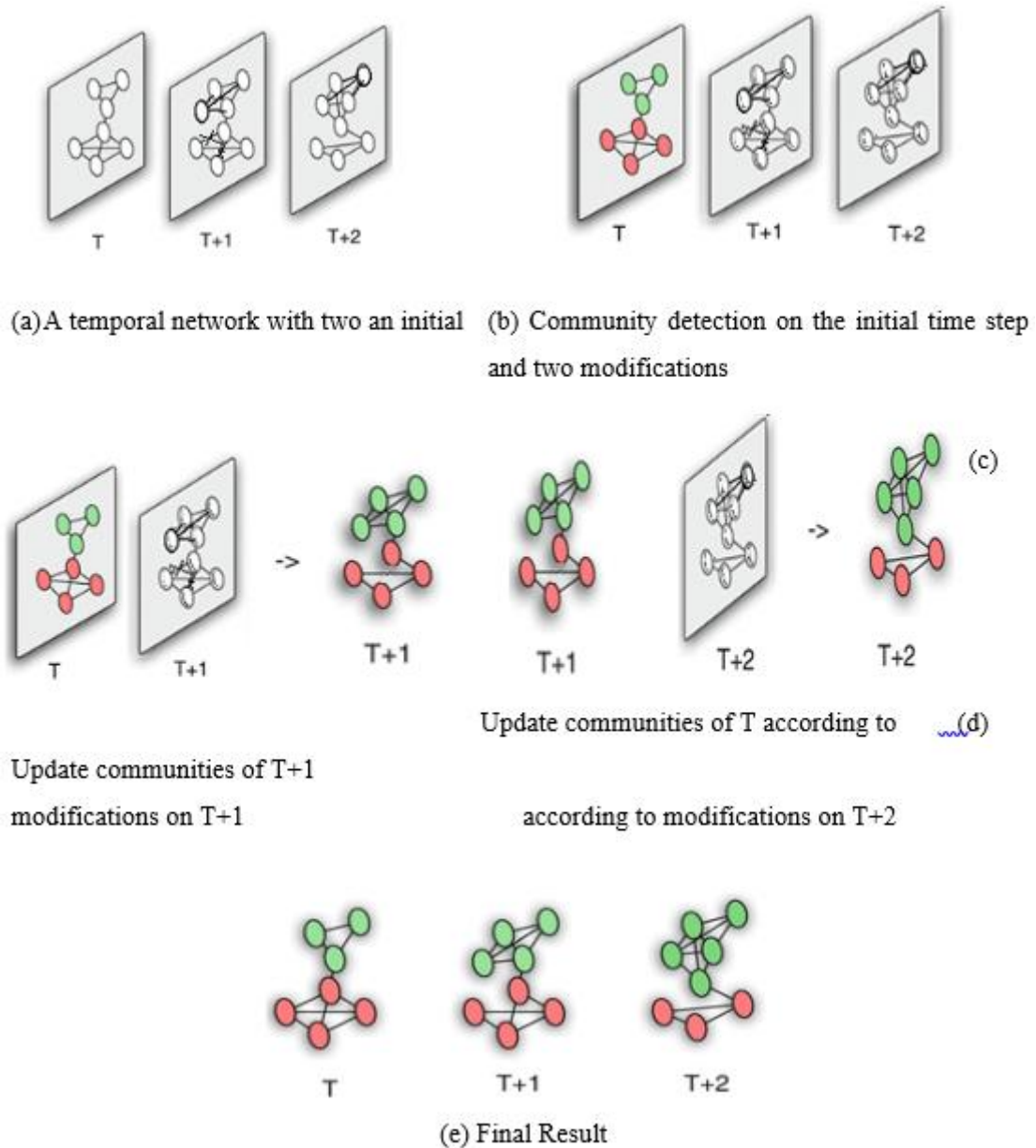


Figure A.4. Dynamic Community Detection on Temporal Networks approach (Cazabet, Rossetti and Amblard 2017)

Additionally, Nguyen et al. (Nguyen, Dinh, Tokala and Thai 2011) propose another method so-called AFOCS (Adaptive FOCS) algorithm. AFOCS adopts same strategies with QCA (Nguyen, Dinh, Xuan, Thai 2011) but it is modified to handle overlapping algorithms. The modification is done via their FOCS (Finding Overlapping Community Structure) procedure which finds dense subgraphs that high possibly have overlaps.

Bhat and Abulaish (Bhat and Abulaish 2015) propose HOCTracker software for tracking the evolution of hierarchical and overlapping communities in online social networks. The program a log-based approach for matching operations for evolutionary community events.

Cordeiro et al. (Cordeiro, Sarmiento and Gama 2016) propose a method, which modifies Louvain algorithm (Blondel, Guillaume, Lambiotte and Lefebvre 2008). The method aims to maximize modularity gain function by just manipulating updated members or relations and left unchanged the other parts of the network.

Rossetti et al. (Rossetti, Pappalardo, Pedreschi and Giannotti 2017) propose TILES algorithm that follows an online iterative procedure for extracting and tracking overlapping communities. Their algorithm takes their name domino tiles because it follows a domino effect strategy. That is, whenever an update occurs, it exploits label propagation procedure locally.

Agarwal et al. (Agarwal, Verma, Agarwal and Chakraborty 2018) propose a novel dynamic community detection method, DyPerm, that optimizes permanence community scoring metric. Permanence is a vertex-based community quality metric that measures the probability of a vertex to already assigned community and the pulled degree by the neighbor communities (Chakraborty, Dalmia, Mukherjee and Ganguly 2017). DyPerm only modifies communities that incoming update effects by keeping other parts of the network unchanged. Authors also give a theoretical guarantee how can achieve permanence maximization via local updates.

Discussion on the Approaches

In this chapter, an adopted taxonomy and framed methods for tracking evolution of communities in dynamic networks in the taxonomy are presented. Then, most related works on the related approaches are presented. In what follows, there is a discussion about the main approaches explained above.

The methods using Independent Community Detection can use any static community detection algorithms and matching functions directly. They can parallelize community detection on each snapshot for time saving as well. However, those methods suffer from unstable nature of community detection algorithms. That is, those community detection algorithms can produce different partitioning even on same dataset if works multiple times.

The methods using Dependent Community Detection approach do not need to detect communities from scratch. Instead, they use previous community structure. However, neither they have a chance of using parallelization nor use of classical community detection algorithms. They try but cannot guarantee stability.

As for the methods using Simultaneous Community Detection on All Snapshots, they do not suffer from instability problem. However, they are not convenient methods for real time or highly dynamic networks.

The methods using Dynamic Community Detection on Temporal Networks approach do not suffer instability problem and low complexity of community updates because they only concern the communities they update. They are suitable for highly dynamic networks. On the other hand, they cannot use static community detection algorithms without modification, and they can evolve invalid communities, i.e. community drift problem.

As it is clear that each approach has its own pros and cons. According to requirements, more appropriate one is chosen.

APPENDIX B

DECIDING GROUND-TRUTH EVENTS

F1-score is used to evaluate the accuracy of the prediction results, the harmonic mean of precision (in Equation 1), and recall (in Equation 2). True Positives (TP) occur when the response variable (evolution event) is classified as positive (yes) by the classifiers while it is positive in fact. Similarly, False Negatives (FN) occur where the response variable is classified as negative (no) by the classifiers while it is positive in fact. As for False Positives (FP), they occur where the response variable is classified as positive by the classifier while it is negative.

$$Precision = \frac{TP}{(TP + FP)} \quad (1)$$

$$Recall = \frac{TP}{(TP + FN)} \quad (2)$$

To evaluate the success of the prediction results, it is necessary to know the ground truth evolution events, since ground truth information is the basis for determining TP, FP, and FN. For this reason, it may be helpful to use a process like the one in Figure A.5 to determine the events. That is, the process takes the community structure of all the time steps that make up the dynamic network and generates a text file that contains ground truth events for communities as output. This process also includes community matching, as it can determine the evolution events based on whether they match or not. Therefore, this section explains the process for determining ground truth events in this study.

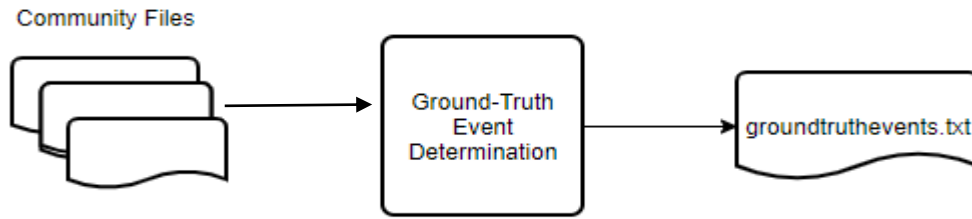


Figure A.5. Input and output of the ground-truth event determination process

To implement such a process, a specific database application is developed. The application first stores the information about members (*Nodes* table) and communities (*Communities* table) as our base dataset. Then, some auxiliary tables (other tables except *Communities* table and *Nodes* table) are created during the process. Figure A.6. shows a *ERD* (entity-relationship diagram) of this database. As can be seen in the figure, the *Communities* table contains the identification numbers of the communities, the time step at which the community forms, and the number of nodes in the community. The *Nodes* table stores a member's time information that specifies its community. The *Matches* table stores ground truth events including community pairs, their respective time steps, and the member size of the communities. The *Intersections* table temporarily stores the values used to calculate the Jaccard similarity between community pairs, as well as the similarity values. Finally, the *NotMatchedCommunities* table temporarily stores the identification numbers of communities that do not match another community. Note that all columns except the *Event* column are numbers and the *Event* column is a string.

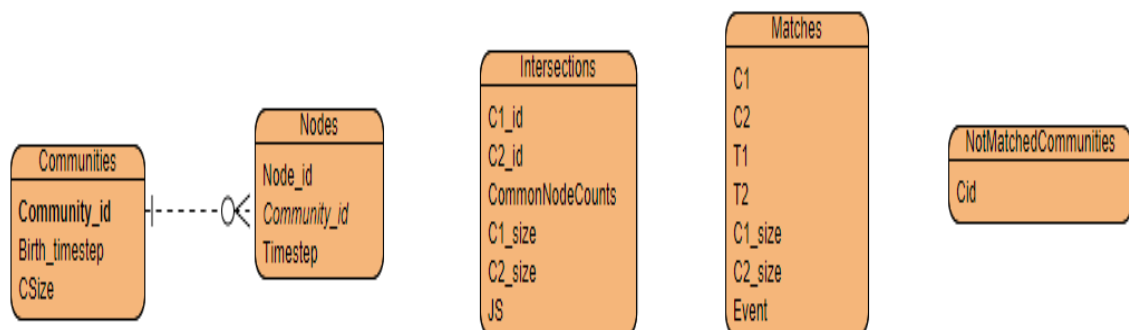


Figure A.6. ERD diagram of the database application

Since we only have information about communities and nodes that form communities as input to the process, the *Communities* and *Nodes* table is populated except for the *csize* column of the *Communities* table. Note that the data filling is done by software that we developed in the Java language.

Input: The database DB, event decision rules

Output: groundtruth.txt (A text file for storing community identification numbers and their respective ground-truth events)

1. Fill *csize* column of DB.Communities by querying them
2. $t = 1$ and $j = 1$
3. DB.NotMatchedCommunities $\leftarrow$ identification numbers of C_t
4. while ($t < tsc$) {
5. while ($j < tsc$)
6. $j = j + 1$
7. DB.Intersections (except *JS* column) $\leftarrow C_t$ from DB.NotMatchedCommunities and C_j
8. Calculate JS values for each row of DB.Intersections
9. Copy calculated JS values into *JS* column of DB.Intersections
10. Populate DB.Matches for community pairs in DB.Intersections records where ($JS \geq 0.1$)
11. Find and insert identification numbers of communities not matched from C_t into DB.NotMatchedCommunities
12. Clear DB.NotMatchedCommunities,
13. Clear DB.Intersections
14. Determine “dissolve” events and append Matches table
15. $t = t + 1$, $j = t$
16. Determine the ground-truth events using the event decision rules on the DB.Matches

Figure A.7. A pseudocode for the process to determine ground-truth events

A pseudocode for this process can be seen in Figure A.7. The process takes the database whose ERD diagram is shown in Figure A.6 and the event decision rules in Table 3.1. Note that Table 3.1 and the figure referenced in the table (Figure 3.2) are

repeated only for completeness. Next, the process generates `groundtruth.txt`, a text file containing the community identification number and the associated ground truth event on each line as output.

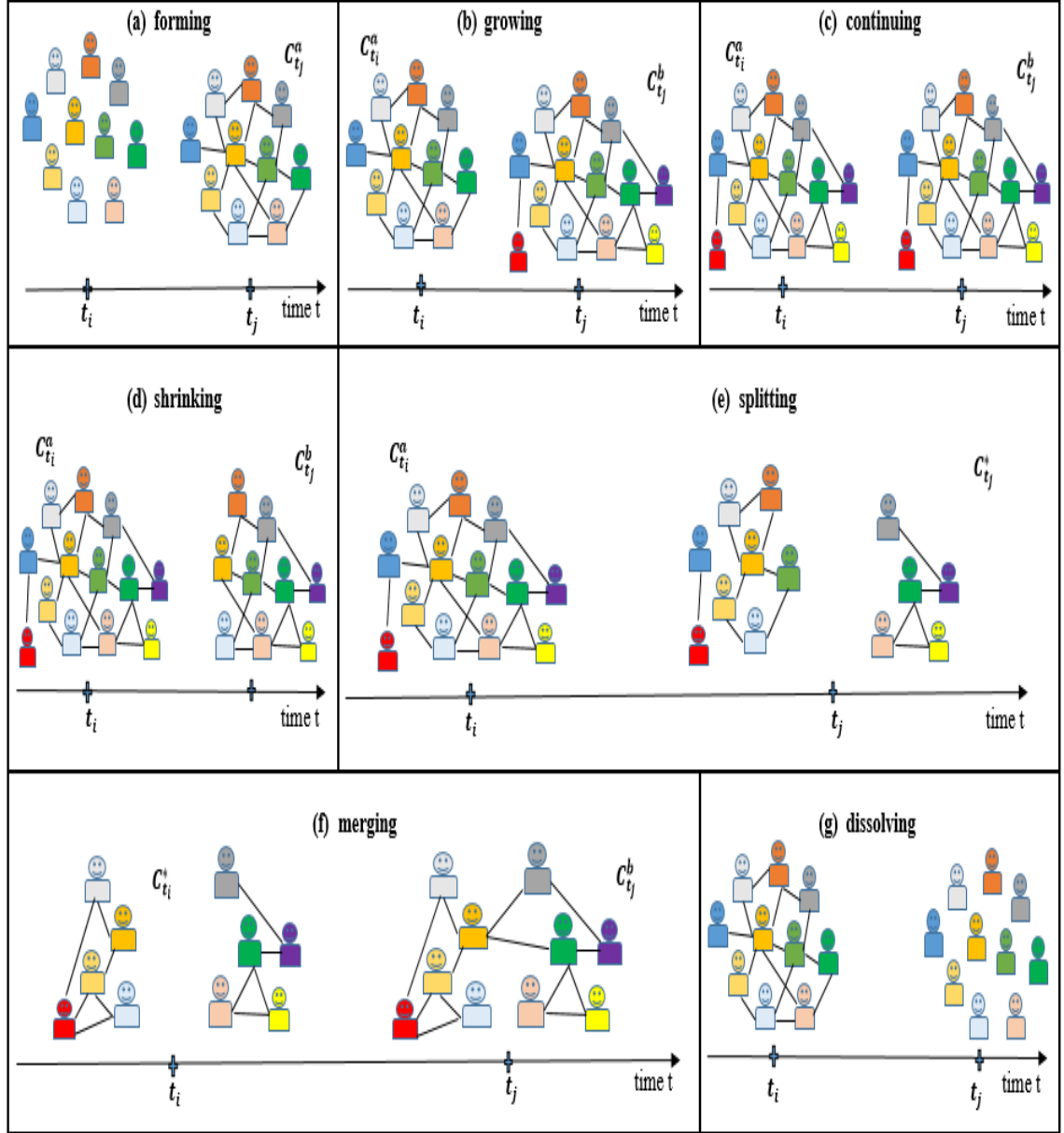


Figure 3.2. An illustration of community evolution events where i and j represent time steps where $i < j$

In Line 1 of the pseudocode, the number of members of the communities is queried with the following query. Then, the column *csize* of the *Communities* table is filled with the results of *nodeCount*.

```
select COMMUNITY_ID, count(node_id) as nodeCount from nodes
where nodes.community_id IN (select community_id from communities)
group by community_id
ORDER BY community_id ASC;
```

Table 3.1. Definition and illustration of community evolution events

Definitions of community evolution event types	Reference To Figure 3.2.
Form: A new community C_{t_i} forms at time t_i. $form(C_{t_j}^a) = true$; if $JS(C_{t_i}^*, C_j^a) < \lambda$ for $\forall C_{t_i}^* \in G$ at time t_i and $t_i < t_j$ and $V_{C_{t_i}^a} \geq 3$.	(a)
Grow: New members may join the community or some existing members may move between communities in graph G over time; hence, some of the communities may grow. $growth(C_{t_i}^a) = true$; if $\exists C_{t_j}^* \in G$ at time t_j and $t_i < t_j$ and $JS(C_{t_i}^a, C_{t_j}^b) \geq \lambda$ and $1.05 V_{C_{t_i}^a} \leq V_{C_{t_j}^b}$	(b)
Continue: Communities can continue their lives either without any change or with changes within tiny upper or lower limits as 0.05 rate of change in community size. $continue(C_{t_i}^a) = true$; if $\exists C_{t_j}^b \in G$ at time t_j and $t_i < t_j$, and $JS(C_{t_i}^a, C_{t_j}^b) \geq \lambda$ and $0.95 V_{C_{t_i}^a} < V_{C_{t_j}^b} < 1.05 \times V_{C_{t_i}^a}$.	(c)
Shrink: A portion of the members may leave a community and cause it to shrink. $shrink(C_{t_i}^a) = true$;	(d)

(cont. on next page)

Table 3.5 (cont.)

$\text{if } \exists C_{t_j}^b \in G \text{ at time } t_j \text{ and } t_i < t_j, \text{ and } JS(C_{t_i}^a, C_{t_j}^b) \geq \lambda, \text{ and } V_{C_{t_j}^b} \leq 0.95 V_{C_{t_i}^a} .$	
Split: A community can split into subcommunities if the similarity threshold λ between the community and the set of subcommunities is satisfied at the following time step. $split(C_{t_i}^a) = true;$ $\text{if } \exists S_{C_{t_j}^*} = \{C_{t_j}^1, C_{t_j}^2, \dots, C_{t_j}^m\} \text{ for } C_{t_i}^a \text{ at time } t_i \text{ and } t_i < t_j, \text{ and } \forall C_{t_j}^* \in S_{C_{t_j}^*}, JS(C_{t_i}^a, C_{t_j}^*) \geq \lambda.$	(e)
Merge: Communities can form a new and larger community by merging. A set of communities $S_{C_{t_i}} = \{C_{t_i}^1, C_{t_i}^2, C_{t_i}^3, \dots, C_{t_i}^n\}$ merge, and form a community $\exists C_{t_j}^b \in G$ at time $t_j > t_i$. The similarity threshold λ is exceeded by each community of $S_{C_{t_i}}$ and $C_{t_j}^b$. $merge(S_{C_{t_i}}) = true;$ $\text{if } \exists S_{C_{t_i}} = \{C_{t_i}^1, C_{t_i}^2, C_{t_i}^3, \dots, C_{t_i}^n\} \text{ at time } t_i < t_j \text{ and } \exists C_{t_j}^b \in G, \forall C_{t_i}^* \in S_{C_{t_i}}, JS(C_{t_i}^*, C_{t_j}^b) \geq \lambda.$	(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 λ in the following steps. $dissolve(C_{t_i}^a) = true; \text{ if } \nexists C_{t_j}^* \in G \text{ at time } t_j > t_i \text{ with } JS(C_{t_i}^a, C_{t_j}^*) \geq \lambda \text{ and } V_{C_{t_i}^a} < 3$	(g)

Two pointers, t and j , are used to point to the time steps including communities to be compared. The pointer t points to the time step to be processed, and the pointer j points to the incoming time steps for matching in both consecutive and nonconsecutive time steps. Initially, both pointers point to the first time step, as seen in Line 2.

The identification numbers of the C_t (e.g. communities in time step t) are retrieved with the following query. Then they are manually copied to the table *NotMatchedCommunities*, as seen in the Line 3.

```

select distinct community_id
from nodes
where timestep = 1
order by community_id ASC;

```

In the Loop A (Line 4 - Line 15) the pointer t goes chronologically through the time steps except for the last one, which tells us the current time step. In the Loop B (Line 5 - Line 13), the pointer j points to the incoming time steps to determine the communities to match with the communities in the time step specified by t . For example, the communities in the time step 1 are compared with the communities to match with the communities in the time step 2. If there are communities that do not match, then they are compared to the communities in time step 3. This task continues in all time steps as long as there are communities that have not yet been matched. After all possible comparisons have been made, t points to the next time step as long as it points to the immediately preceding time step of the last time step (tsc), and j is matched with t , as seen in line 15. Loop B is explained in more detail in the following section.

In the Line 6, pointer j is moved to next time step. The information of the communities whose identification numbers are in the table *NotMatchedCommunities* and C_j is inserted into the table *Intersections*, except for the column JS, as seen in Line 7. The following query is used for this task.

```

insert into intersections (c1_id, c2_id, commonnodecounts)
select t1.community_id as c_t1, t2.community_id as c_t2, count(t1.node_id) /*for
finding shared number of nodes in a community pair*/
from nodes t1, nodes t2
where t1.timestep=1 and t2.timestep=2 and t1.node_id=t2.node_id
group by t1.community_id, t2.community_id
ORDER BY t1.community_id ASC;

```

The size of the first and second communities is determined by the following two queries. They are then manually copied to the column *c1size* and the column *c2size* of the *Intersections* table.

```

select t1.c1_id, t2.csize as c1size from intersections t1, COMMUNITIES t2
where t1.c1_id = t2.COMMUNITY_ID; /*to obtain the size of the first community in
the record.*/
select t1.c2_id, t2.csize as c2size from intersections t1, COMMUNITIES t2
where t1.c2_id = t2.COMMUNITY_ID; /*to obtain the size of the second community
in the record.*/

```

Then, the values of Jaccard Similarity (JS) are calculated using the query below for the community pairs in the *Intersections* table (see Line 8). Then the values of JS are manually copied to the *JS* column in the *Intersections* table (see Line 9).

```

select c1_id, c2_id, commonnodecounts, c1size, c2size,
commonnodecounts/(c1size+c2size-commonnodecounts) as js
from intersections; /*to calculate js*/

```

Next, the table *Matches* is populated with the required values for its columns from the table *Intersections* and the table *Communities* if the value *JS* of a community pair is greater than or equal to 0.1 (e.g., similarity threshold - the lowest value above which a community pair is considered similar). The task can be seen in Line 10.

```

/*fill the Matches table*/
insert into matches
select t1.c1_id, t1.c2_id, t2.birth_timestep, t3.birth_timestep, t1.c1size, t1.c2size
from intersections t1, communities t2, communities t3
where t1.js >= 0.1 and t1.c1_id=t2.community_id and t1.c2_id=t3.community_id
order by c1_id ASC;

```

Later, the records in the table *NotMatchedCommunities* are deleted with the following query, as seen in Line 11.

```

delete from notmatchedcommunities;

```

Next, the communities in time step t that do not match the communities in time step j are found and inserted into the table *NotMatchedCommunities*, as seen in Line 12.

```
insert into notMatchedCommunities
select distinct(c1_id) from intersections
where intersections.c1_id not in(select c1 from matches)
order by intersections.c1_id;
```

The records of the table *Intersections* are cleared with the query below as seen in Line 13.

```
delete from intersections;
```

If there are still nonmatching communities from C_t , after loop B finishes, it means that these communities are dissolved. They are appended to the table *Matched* with $t1$ and $t2$ as "1", as seen in Line 14. These "1" values are sentinel values that inform us about the dissolve events.

```
/*to add dissolved communities to Matches table*/
insert into matches
select t1.cid, t1.cid, 1 as t1, 1 as t2, t2.csize, t2.csize
from notmatchedcommunities t1, communities t2
where t1.cid=t2.community_id;
```

At the end of Loop A, ground-truth events are determined by the decision rules, as seen in Line 16 of the Figure A.7.

Form events are not considered because a community must be formed to predict community evolution. To populate the *Event* column in the table *Matches*, the following query is used. For example, sentinel values are searched for dissolve events. When found, the event of the associated record is updated as "dissolve".

```
update matches
set event = 'dissolve'
where t1 = 1 and t2 = 1;
```

Later, the split events are determined using the following query. In this query, the communities divided into more than or equal to two are searched, and then the column *Event* in the table *Matches* is updated as "split".

```
update matches
set event = 'split'
where c1 IN (Select c1 from matches
             group by c1
             having count(C1) > 1);
```

Next, merge events are identified using the query below. This query looks for the communities of which more than one has been merged into a larger community, and then updates the column *Event* in the table *Matches* as "merge".

```
update matches
set event = 'merge'
where c2 IN (Select c2 from matches
             where event is null
             group by c2
             having count(C2) > 1);
```

Finally, the events "grow", "shrink", and "continue" are recognized in order. Corresponding queries can be found below. In these queries, the events are determined based on the change in the number of members of the matching communities. The column *Event* in the table *Matches* is then updated.

```
/*GROW*/
update matches
set event = 'grow'
where event is null and t1 < t2 and (c1size * 1.05) < c2size;
```

```
/*SHRINK*/
update matches
```

```
set event = 'shrink'
where event is null and t1<t2 and c2size< (c1size*0.95);

/*CONTINUE*/
update matches
set event = 'continue'
where event is null;
```

After labeling events in the table *Matches*, the second community (C2) and the column *Event* in the table are copied into a text file (groundtruth.txt). In this way, the ground-truth events for the prediction task are obtained. For the repetition of this analysis and further use of these tables and codes by other researchers; they are shared in the link: <https://github.com/akaratas/groundTruthEvents> .

APPENDIX C

FEATURE DETERMINATION SUBPROCESS

In this context of this case study, structural, temporal, and leadership features of the communities are considered as well, which will be relevant for the prediction of community evolution.

Table A.1. Structural features of the communities

No	Type	Feature	Description	Definition
1	Structural	Size (S)	Number of nodes in community k at time t (C_t^k)	n_t^k
2	Structural	Density (D)	Ratio of edges (E) to the maximum possible edges.	$\frac{2 \times E_t^i }{ V_t^i \times (V_t^i - 1) }$
3	Structural	Clustering Coefficient (CC)	Ratio of the sum of the clustering coefficient(cc) of the community nodes to the number of nodes(n) in the community.	$\frac{cc_t^i}{n_t^i}$
4	Structural	Cohesion (Ch)	Strength of connections inside the community in relation to the connections outside(OE) of it.	$\frac{\frac{2 \times E_t^i }{ V_t^i \times (V_t^i - 1) }}{\frac{ OE_t^i }{ V_t^i \times (V_t^i - 1) }}$
5	Structural	Average Degree Centrality (ADC)	Ratio of the sum of degrees of the nodes in the community D_t^i to the number of nodes in the community.	$\frac{D_t^i}{n_t^i}$
6	Structural	event	Currently occurred event for the community such as dissolve, grow, shrink, merge and split	

The name of the features, their types, description, and definitions in Table A.1, Table A.2, and Table A.3 are listed. In the first column “No”, the selected features are numbered. In the second column “Type”, the type of the feature as structural, temporal, or leadership features are specified. In the third column “Feature”, the name and abbreviation of the features are given. In the fourth column “Description”, the explanation for each feature is provided. In the last column, the definitions of the specific features reside.

In Table A.1, regarded structural features of the communities are listed. In the table, feature titles, descriptions of the features, and definitions of the features are placed. In Table A.2, regarded temporal features between communities are listed. As is seen from the table, feature names start with a Δ symbol. This Δ symbol indicates the difference among the features written in curly brackets. For illustration, $\Delta \{size\}$ feature gives the difference between size.

Influential members (e.g., leaders of communities) can play important roles in the evolution of communities. For example, if a leader leaves one community, her followers may become less active or disperse to other communities as well. In this case, the community may lead to community shrink or even dissolve. Because of this intuition, some features of influential members are decided to be regarded.

Influential members of communities are determined according to their Eigenvector centrality values. These values are obtained via the Gephi graph visualization library⁵. If the value is equal to or higher than 0.5, then it is regarded as one of the leaders in this study. The features of influential members specified in Table A.4. are considered because of the number of leaders in a community (LC), the average values of Eigenvalues (LE), and the average of degree values (LDC) of leaders of a community can affect and tell about node transitions among communities. The node transitions may affect evolution events that a community may undergo. For example, if the number of leaders decreases, it may mean this community split, shrink or dissolve events.

In our study, all considered features are real values, and only “event” feature is a categorical variable and it is converted to integer values with the encoding below in the training dataset. In the first column of Table A.4., the values of *the event* feature and in the second column their corresponding integer codes used for encoding are listed.

⁵ <https://gephi.org/>

Table A.2. Temporal features of the communities

No	Type	Feature	Description	Definition
7	Temporal	$\Delta\{size\}$	The change amount in size between current community i at time t and previous community j at time step $t - 1$	$s(c_t^i) - s(c_{t-1}^j)$
8	Temporal	$\Delta\{density\}$	The change amount in density between current community i at time t and previous community j at time step $t - 1$	$D(c_t^i) - D(c_{t-1}^j)$
9	Temporal	$\Delta\{CC\}$	The change amount in average clustering coefficient between current community i at time t and previous community j at time step $t - 1$	$CC(c_t^i) - CC(c_{t-1}^j)$
10	Temporal	$\Delta\{cohesion\}$	The change amount in cohesion between current community i at time t and previous community j at time step $t - 1$	$Ch(c_t^i) - Ch(c_{t-1}^j)$
11	Temporal	$\Delta\{ADC\}$	The change amount in average degree centrality between current community i at time t and previous community j at time step $t - 1$	$ADC(c_t^i) - ADC(c_{t-1}^j)$

Table A.3. Leadership features of the communities

No	Type	Feature	Description	Definition
12	Leadership Features	Leaders Count (LC)	the number of leaders of current community i at time t	$\sum L(C_t^i)$
13	Leadership Features	Leaders Eigen (LE)	is the average of Eigen values of leader nodes in the current community i at time t	$\frac{\sum LE(C_t^i)}{\sum L(C_t^i)}$
14	Leadership Features	Leaders Degree Centrality (LDC)	the average of degree values of leader nodes in the current community i at time t	$\frac{\sum LDC(C_t^i)}{\sum L(C_t^i)}$

Table A.4. Codes of Values of Event feature

Event	Correspondent Integer Value
Form	1
Continue	2
Grow	3
Shrink	4
Merge	5
Split	6
Dissolve	7

APPENDIX D

SELECTED MACHINE LEARNING CLASSIFIERS

In this section, we summarize the operation of the selected machine learning classifiers. Moreover, for the sake of completeness, each of them is exemplified in this study.

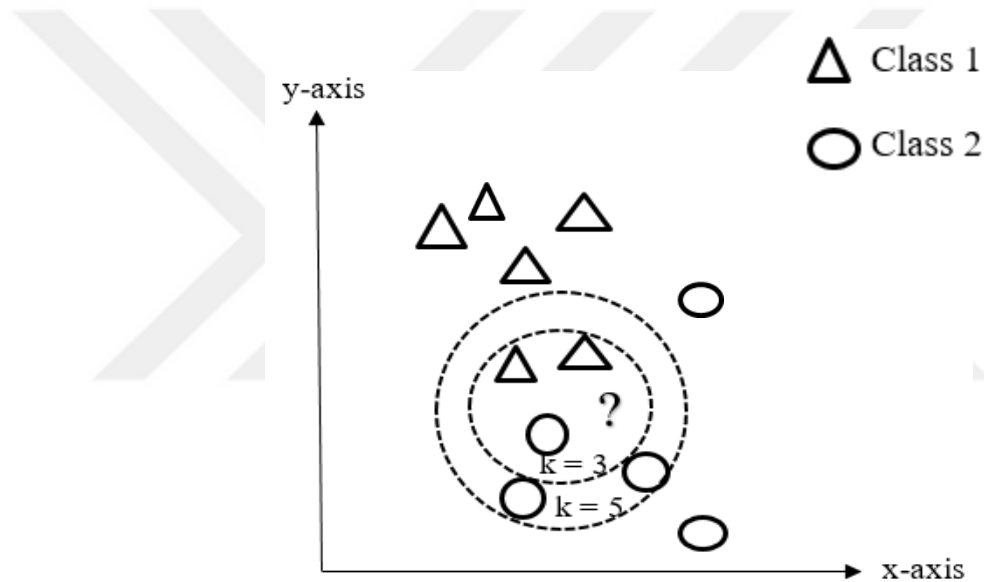


Figure A.8. An example of “how Ibk (k-NN) works”

IBk (K-nearest neighbor classifier) (Aha and Kibler 1991) is one of the common supervised learning classifiers. Additionally, it makes lazy learning where there is no training of the model but the model memorizes the training set. At the decision time, the model searches for the k-nearest neighbors and decides in favor of the majority of class labels that neighbors belong to. The distance among the neighbors is gauged with distance functions. In this study, the Euclidean distance function is used because our features are real-valued vectors (floating-point and integer values). Just a kind note, since Ibk automatically does the normalization process, there is no need to do it explicitly.

Otherwise, there is a need for a normalization process. On the other hand, according to the characteristic of the dataset worked on other distance metrics such as Manhattan (city blocks) and Hamming can be used. Manhattan distance is usually preferred when high dimensionality comes to two integer-valued vectors. Hamming distance is used to measure the distance between two binary vectors.

In Figure A.8, there is an illustration of how Ibk (Aha and Kibler 1991) classifier makes the decision. In the example, there are some labeled instances either Class 1 or Class 2. Let us assume that there is a new instance, indicated with a question mark symbol in the figure, to be classified by the classifier. If k is chosen as 3, then the three neighbors that have the least distance to the new instance (e.g., 3-near neighbors) are determined. According to the figure, there are two Class 1 instances while there is one instance from Class 2 and the classifier decides in favor of Class 1 because of its majority in terms of instance counts. Likewise, k is chosen as five, then the classifier decides in favor of Class 2 because of its majority in near neighbors.

Table A.5. Weather.arff file

Outlook	Temperature	Humidity	Windy	Play?
sunny	85	85	false	no
sunny	80	90	true	no
overcast	83	86	false	yes
rainy	70	96	false	yes
rainy	68	80	false	yes
rainy	65	70	true	no
overcast	64	65	true	yes
sunny	72	95	false	no
sunny	69	70	false	yes
rainy	75	80	false	yes
sunny	75	70	true	yes
overcast	72	90	true	yes
overcast	81	75	false	yes
rainy	71	91	true	no

J48 (C4.5 Decision Tree) (Quinlan 1993) is a Decision Tree Classifier that can be used to make a decision based on a sample of data. The classifier aims to create as a small tree as possible. The classifier decides what feature is placed at the root according

to the information gain (IG) of the features. That is, the feature that has the highest IG is placed on the root. In Table A.5, there is a weather dataset provided in Weka 3.8 example datasets. On this dataset, how J48 creates a decision tree is explained in the next paragraphs.

The features {Outlook, Temperature, Humidity, Windy} are feature variables in type {categorical, integer, integer, binary} respectively. J48 can work with real-valued variables, categorical variables, and binary-valued variables. First, J48 calculates IG for all features, then sorts them according to their IG accordingly. According to their ranks, they are placed on the decision tree. The calculation of IG values for each feature is shown below in detail. To calculate IG, the entropy of playing golf and entropy of play golf due to the outlook feature must be known.

Therefore,

$$\begin{aligned}
 \text{Entropy (Play Golf)} &= -\sum_1^C p_i \log_2 p_i \\
 &= -[P(\text{yes}) \log_2 P(\text{yes}) + P(\text{no}) \log_2 P(\text{no})] \\
 &= -\left[\left(\frac{9}{14}\right) \log_2 \left(\frac{9}{14}\right) + \left(\frac{5}{14}\right) \log_2 \left(\frac{5}{14}\right)\right] \\
 &= 0,94
 \end{aligned}$$

$$\text{Entropy (Play Golf, Outlook)} = \sum_1^C P(C)E(C)$$

$$= P(\text{Sunny})\text{Entropy}(\text{Sunny}) + P(\text{Overcast}) \text{Entropy}(\text{Overcast}) + P(\text{Rainy}) \text{Entropy}(\text{Rainy})$$

$$= 0,693$$

$$\text{IG (Outlook)} = \text{Entropy (Play Golf)} - \text{Entropy (Play Golf, Outlook)}$$

$$= 0,94 - 0,693$$

$$= 0,247$$

The same steps for other features are repeated. Just for the simplification, in Figure A.9., frequency tables are IG of each feature is given. When the features are ranked according to their information gains, the final decision tree is created and visualized in WEKA as seen in Figure A.8.

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1
IG (Temperature) = 0,03			

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1
IG (Humidity) = 0,152			

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3
IG (Windy) = 0,05			

Figure A.9. Frequency tables and information gains of the features in the Weather dataset⁶

In Figure A.10, the main feature is *outlook*. If **the outlook is sunny or rainy** then J48 further analyzes the humidity or windy, correspondently. If *humidity* is high, then class label of the instance is yes (play). **If outlook is rainy**, further classification takes place to analyze *windy* feature. **If the outlook is overcast**, then the label of instance is yes (play). The numbers written on the leaf nodes and in the parenthesis show the instances which obey the classification. For example, four instances play golf when the overlook is overcast.



Figure A.10. Visualization of the decision tree created by J48 classifier in WEKA

⁶ https://www.saedsayad.com/decision_tree.htm

Random Forest (Breiman 2001) classifier consists of a large number of relatively uncorrelated decision trees as a committee. The key concept behind it is the wisdom of crowds. That is, each tree in the forest makes its decision, then the forest decides in favor of the class with the majority of the votes. In Figure A.11, there is an example random forest that makes a prediction. When a new instance comes to prediction, the classifier sends the instance to each decision tree inside, then each tree make its own decision. The classifier counts the votes and make final decision in favor of decision of the majority.

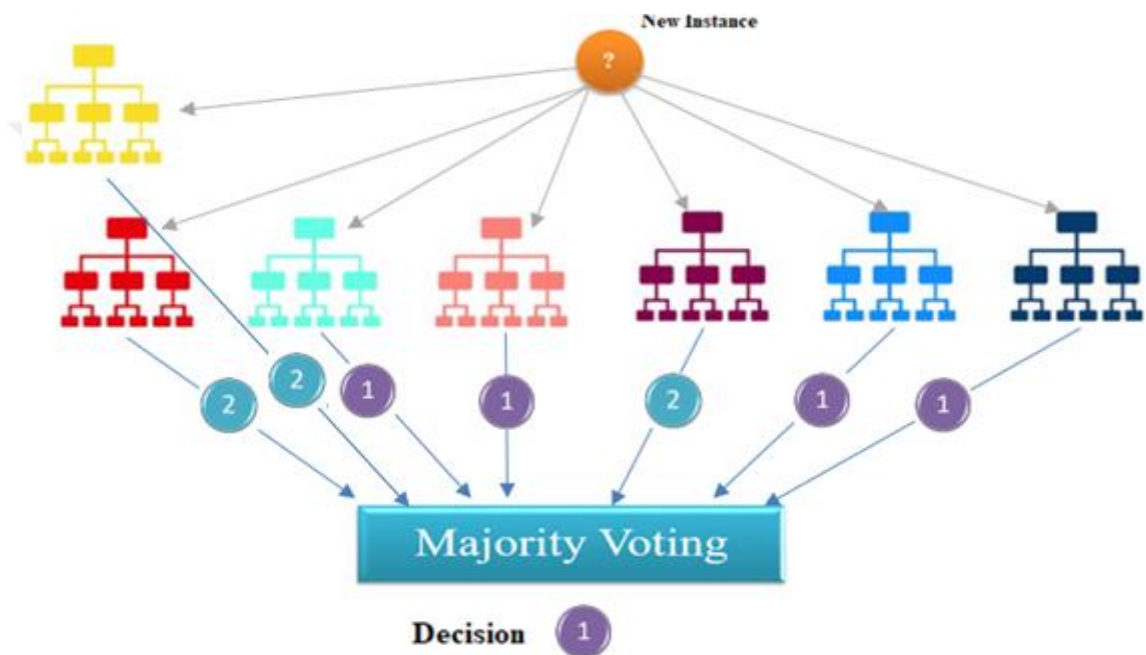


Figure A.11. An illustration of decision process for an example random forest classifier

There is a prerequisite for a random forest classifier to perform well. The prerequisite is that decision trees building the forest need to be relatively uncorrelated (or low correlated). In this way, the individual errors of the trees do not affect others. This prerequisite is provided via bagging and feature randomness processes. Both processes together ensure each decision tree in the forest diversifies each other, which leads to low correlation/relatively uncorrelation among the trees.

Bagging (Breiman 1996), short for Bootstrap Aggregation, is a process that allows each tree to randomly sample from the dataset with replacement. This process

brings in different trees because decision trees are very sensitive to the training data. That is, small changes on the training set result in significantly different trees. Bagging neither split the training set into smaller chunks nor trains decision trees on these chunks. Rather, it allows random samples from the training set with replacement with the size of the training set. That is, each decision tree is trained on different data coming from the training set but the same size. **Feature randomness** is a process that allows each tree to pick only a random subset of features. In this way, different features are used to make decisions. This process forces even more variation among the trees, which results in more diversification amongst the trees.

As for **Bagging** as a classifier model, a bagging process is run to create multiple random samples to train the classifiers. Then, classifiers selected as many as the number of samples are run on these samples. Each classifier makes its own decision and the results are averaged. Bagging as a classifier model is data-dependent and it well performs when the training data has low bias and high variance.

Random Trees⁷ are the combination of single model trees and Random Forest ideas. Model trees are trees considering K randomly chosen features at each node to calculating the entropy of each node. This approach increases diversity among the decision trees in the forest and improves generalization in general.

⁷ <https://weka.sourceforge.io/doc.stable-3-8/weka/classifiers/trees/RandomTree.html>

VITA

Arzum Karataş was born and grew up in Turkey. She attended Dokuz Eylül University Izmir Vocational School in 2002 and placed 1st in Computer Technology and Programming Department in 2004. She earned a Bachelor of Science degree with High Honors and ranked Rector List in Izmir Institute of Technology in 2012. In 2015, she earned a Master of Science degree from the same university.

She worked as a teaching assistant in Gediz University between 2013 and 2016 years on many courses such as "Introduction to Programming", "Object Oriented Programming", "Database Management Systems", "Software Engineering" and "Microprocessors". She was also a grader for "Formal Languages and Automata". In addition, she was a mentor for "Industrial Training" and "Senior Design Projects". She has also been a co-organizer of seminars and activities of the department.

She has been a student member of the Association for Computing Machinery (ACM) and the Institute of Electrical and Electronics Engineers (IEEE) since 2014. In addition, she has been a professional member of Turkey Computer Engineers Association since 2016.

The List of the Titles of Her Publications:

- A Novel Efficient Method for Tracking Evolution of Communities in Dynamic Networks. (Under Review)
- Sosyal Bot Algılama Teknikleri ve Araştırma Yönleri Üzerine Bir İnceleme.
- A Comparative Analysis of Two Most Recent Dynamic Community Tracking Methods.
- Application Areas of Community Detection: A Review
- A comparative study of modularity-based community detection methods for online social networks.
- EDU-VOTING: An Educational Homomorphic e-Voting System.
- A Review on Social Bot Detection Techniques and Research Directions

The Scholarships:

This dissertation has been supported by both Turkey Council of Higher Education (YÖK) through 100/2000 PhD Scholarship Program and The Scientific and Technological Research Council of Turkey (TÜBİTAK) through 2211-C Doctoral Scholarship program since 2018.