A Scalable Approach to Mining Communication Motifs from Dynamic Networks

A THESIS

submitted by

GURUKAR SAKET GHANSHYAM

for the award of the degree

of

MASTER OF SCIENCE

(by Research)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MARCH 2015

THESIS CERTIFICATE

This is to certify that the thesis titled **A Scalable Approach to Mining Communication Motifs from Dynamic Networks**, submitted by **Gurukar Saket Ghanshyam**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Science**, is a bonafide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. B Ravindran
Research Guide
Associate Professor
Dept. of CSE
IIT-Madras, 600 036

Dr. S Ranu
Research Guide
Assistant Professor
Dept. of CSE
IIT-Madras, 600 036

Place: Chennai

Date: July 4, 2015

ACKNOWLEDGEMENTS

Life is hard. This is what Ravindran sir says when you tell him it is difficult to manage all the things in short time. I would rather say that "Life is fun, if you are working under Ravindran sir". He is a perfect example of Lead by example. He always pushed me to achieve whatever I was able to achieve and at the same time whenever I faced any issues in insti he acted like a shield for me.

I was lucky enough to work under two talented guides. Sayan sir is the main reason for the SIGMOD publication that came out of this thesis. Whenever I faced any difficulties, he was available for me. It was not uncommon for me to receive a response mail of my query on even weekend nights. Under him, I learnt to prioritize among problems. He is a friend and guide to me.

I would like to thank my friend Renu Karule who supported me whenever I felt low. She always has been a source of motivation for me. I studied together with Avijit and Sai and they helped me clear lot of doubts in academics. Avijit and I shared same classes, same hostel, same lab and same guide. I will always remember the time we spent together studying, playing, dining, arguing and laughing.

This journey of IITM was pleasant because of my dear friends Karishma, Abhilash, Mandeep, Priya and Raj. I will always cherish the moments we spent together. I am glad that I met all of them. It is because of them I was able to enjoy my life both inside and outside IIT.

I want to thank Sarath for his repeated attempts to teach me Tamil language. Discussions with Biswa during our tea sessions were funny and enjoyable. I also want to

thank Arpita for being a good and kind friend. Lastly, I would like to thank Ericsson Research for funding my project.

Three years ago, I had a choice of either joining MS course in IITM or management course in NITIE. I chose IITM and my parents supported my decision without asking any questions. I want to dedicate this thesis to my parents for keeping faith in me.

ABSTRACT

KEYWORDS: Data Mining ; Graph Mining ; Social Networks ; Communication Motifs

Social networks have become an effective means of communication among people. Recently, there is a trend to analyze these social networks to infer the dynamics of human interaction. A fundamental problem in behavioral analysis of human interactions is to understand how communications unfold. In this thesis, we propose and solve the problem of mining *Communication motifs* from dynamic interaction networks. Simply stated, a communication motif is a recurring subgraph that has a similar sequence of information flow. Communication motifs provide a powerful mechanism to capture the dynamics of human interactions.

Existing work show that communication motifs reveal how the functional behavioral patterns evolve with time, how the structures of these patterns change with the social network, and finally, how the social network influences the speed and amount of information exchanged in communications between individuals. However, no technique is proposed for mining these motifs in a scalable manner. Mining communication motifs requires us to explore the exponential subgraph search space where existing techniques fail to scale. To tackle this scalability bottleneck, we develop a technique called *COMMIT*. COMMIT converts a dynamic graph into a database of sequences. Through careful analysis in the sequence space, only a small portion of the exponential search space is accessed to identify regions embedding communication motifs. We also store the pointers to these regions as a result costly subgraph enumeration step is avoided.

We perform extensive experiments on three real world datasets and evaluate the proposed COMMIT based on accuracy and scalabilty. We find that COMMIT is up to two orders of magnitude faster than baseline techniques. COMMIT can also mine large size communication motifs where existing algorithms fails. Furthermore, qualitative analysis demonstrate communication motifs to be effective in characterizing the recurring patterns of interactions while also revealing the role that the underlying social network plays in shaping human behavior.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

COMMIT COMmunications Motifs in InTeraction networks

CHAPTER 1

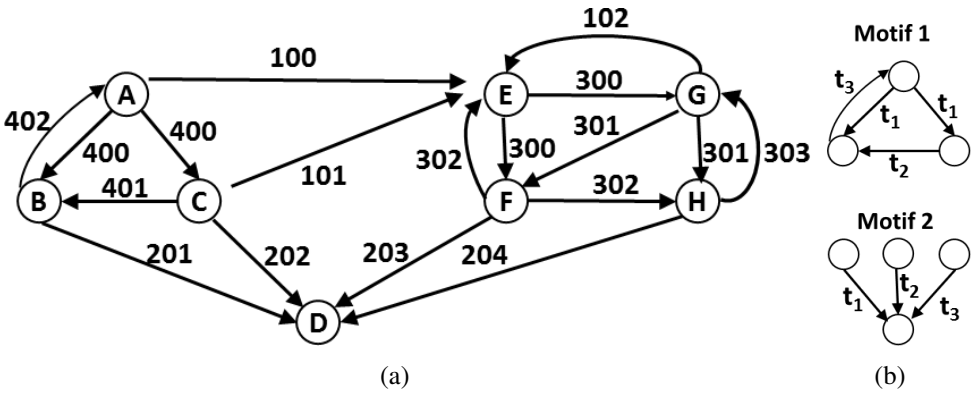
Introduction

Interactions in social networks are typically studied using graphs where users are represented as nodes and interactions between them are represented as edges. A fundamental task in social network analysis is to understand how communications unfold. Are there patterns that recur time to time? What role does the underlying social network play in the progression of human communication? In this thesis, we study the behavioral aspects of interactions within social networks by mining *communication motifs* from large dynamic networks.

1.1 Importance of communication motifs

To illustrate the concept of communication motifs in a dynamic network, consider Fig. 1.1(a). In this dynamic network, an edge with a timestamp t between nodes A and B represents an interaction event between A and B at time t. Interaction events can be phone calls, e-mails, Facebook wall posts, tweets, etc. Note that an individual can interact with multiple individuals at same time. For example in Fig. 1.1(a), A interact with B and C at time 400.

Due to the intrinsic social nature of human beings, it is common for an interaction between two individuals to spurt further communication activities. For example, a person claiming Real Madrid to be the best soccer club in Facebook is likely to encourage further interactions from Real Madrid fans supporting the claim and possibly, stiff opposition from Barcelona fans. To capture this dependency between interactions,



(a) (b)

Figure 1.1: (a) A dynamic network denoting interactions between its users (b) The two largest communication motifs at a frequency threshold of 3 and $\Delta T = 1$. Timestamp $t_i < t_j$ if $i < j$. Motif 1 involves nodes $\{A, B, C\}$ $\{E, F, G\}$, and $\{G, H, F\}$. Motif 2 involves $\{B, C, D, F\}$ $\{C, D, F, H\}$, and $\{A, C, E, G\}$.

we assume that two edges in a social network are related if they involve a common user and the difference in their timestamps is within some threshold ΔT. In Fig. 1.1(a), for example, A sends a message to B and C simultaneously at time 400. This initiates an interaction between C and B at time 401 and then subsequently, B responding to A at time 402. At $\Delta T = 1$, this sequence of interactions are considered related. At the same time, the interaction between A and E is not related to these since it occurs at timestamp 100, which is more than ΔT away from the other interactions of A. Now, notice that two other exact same *sequences* of related interactions also exist between E, F, G, and G, H, F. These interactions are explicitly shown in Fig. 3.1. In other words, this pattern of interaction is frequent in the social network and characterizes one of the common communication patterns. We call such a pattern as a *communication motif* if its frequency is higher than a user-defined threshold θ. At $\theta = 3$, the two largest communication motifs are shown in Fig. 1.1(b). While the first motif is likely capturing some group discussion, the second motif is the structure that is typically generated while wishing a person on a special occasion such as birthday, marriage, etc.

Communication motifs provide a powerful mechanism to capture the dynamics of human interactions. A similar line of work was explored by Zhao et al.[Zhao *et al.*, 2010]. They show that communication motifs reveal how the functional behavioral patterns evolve with time, how the structures of these patterns change with the social network, and finally, how the social network influences the speed and amount of information exchanged in communications between individuals. However, no technique is proposed for mining these motifs in a scalable manner. A communication motif is essentially a frequent subgraph in a dynamic network with some additional properties. First, the edges in each embedding of the subgraph must form a chain of related interactions based on a user-provided threshold ΔT. Second, the edges in each embedding of the subgraph must have the same sequence of interactions.

1.2 Issues in mining communication motifs

Mining frequent subgraphs from large networks is a hard problem since the number of subgraphs in a network grows exponentially with the size of the network. In addition, to compute the frequency of a subgraph, we need to perform subgraph isomorphism, which is NP-complete [Zeng *et al.*, 2009]. Owing to its hardness, frequent subgraph mining has received significant interest in the research community with GRAMI [Elseidy *et al.*, 2014] being the state-of-the-art technique in this space. However, the following aspects of communication motifs render the existing methods inapplicable to our problem.

•**Incorporating temporal information:** Existing frequent subgraph mining techniques ignore the temporal aspect. As a result, the notion of edge relatedness cannot be enforced easily in such techniques. To combat this weakness of existing techniques, one could adopt the following two-stage approach. In the first stage, all frequent sub-

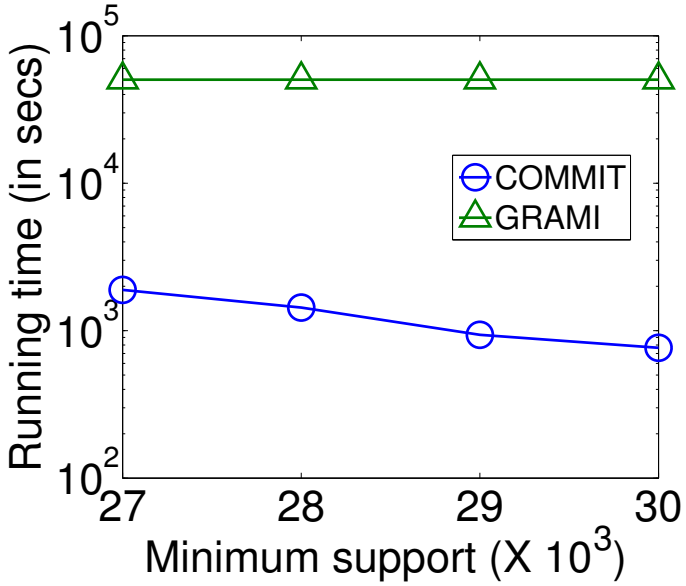


Figure 1.2: Running time comparison of GRAMI and COMMIT against the support threshold on the Facebook dataset[Viswanath *et al.*, 2009]. Note that GRAMI was terminated after 16 hours in all cases without having completed the computation.

graphs are mined. Then, in the second stage, each of the frequent subgraphs are verified whether they satisfy the temporal constraints of a communication motif. Unfortunately, this approach does not scale due to the unimportance of *node labels* in our problem.

• **Unlabeled Nodes:** As can be seen in Figs. 1.1(a) and 1.1(b), the node labels denoting user IDs do not play any role in communication motifs; only the structure and the timestamps matter. Existing frequent subgraph mining techniques rely heavily on the presence of node labels to prune the search space. Consequently, they fail to scale in our setting even if we ignore the temporal aspect.

To empirically establish the impact of unlabeled nodes, we run GRAMI on an interaction network constructed from Facebook[Viswanath *et al.*, 2009]. Fig. 1.2 presents the results. On this network, when the support threshold is less than $30,000$, GRAMI fails to complete even after 16 hours.

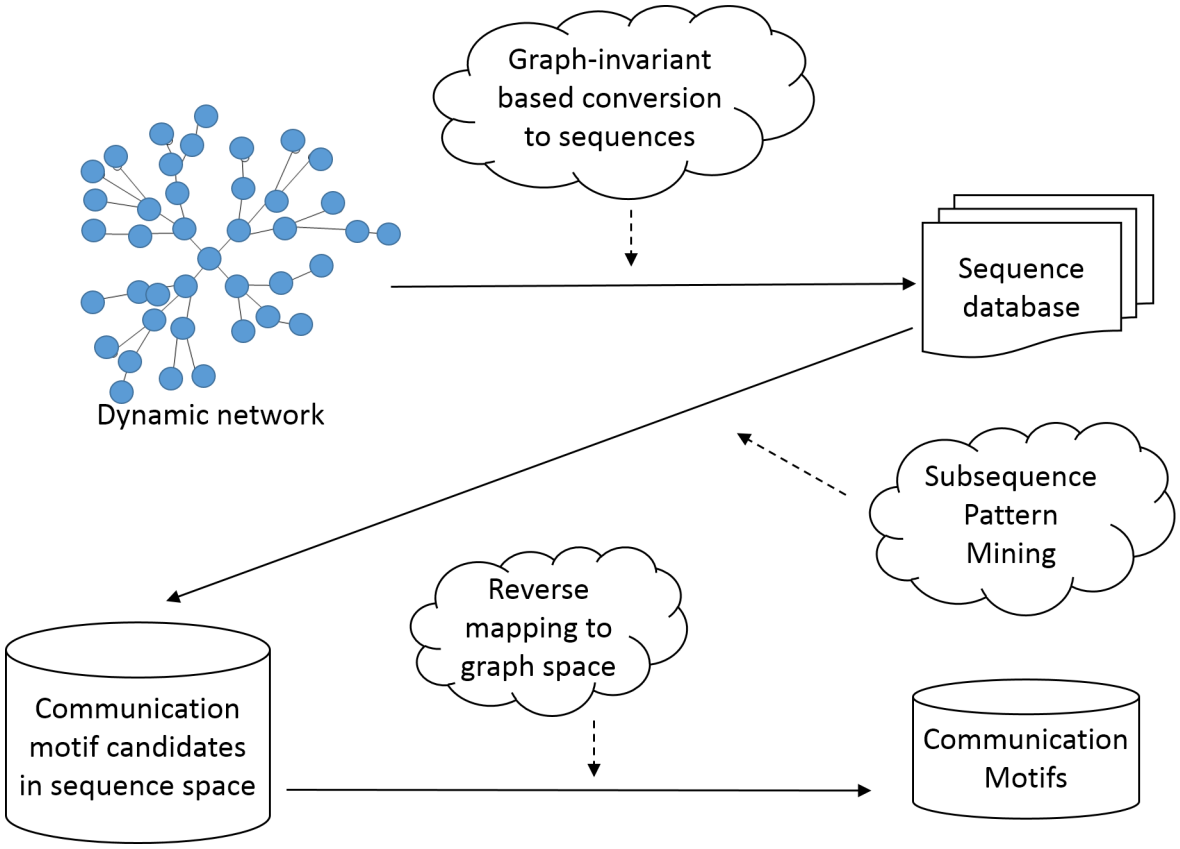


Figure 1.3: Pipeline of the COMMIT algorithm.

1.3 Outline of COMMIT

To address the challenges outlined above, we design a new algorithm called *COMMIT (COMmunication Motifs in InTeraction networks)* to mine communication motifs from large interaction networks. In contrast to GRAMI, on the Facebook dataset in Fig. 1.2, COMMIT takes around 20 minutes to complete. Fig. 1.3 presents the pipeline of proposed algorithm. In the first step, each of the connected components of the dynamic network is converted into a sequence of its interactions. This results in the dynamic network being represented as a database of interaction sequences. Through a careful analysis using graph invariants in this sequence space, we mine the frequent subsequence patterns that could potentially represent communication motifs. These patterns are then

converted to the graph space for verification and the final answer set is computed.

The proposed approach saves time in two accounts. First, COMMIT constructs a coarse-grained representation of the network by converting them to sequences. As we show later, the proposed graph-invariant based conversion scheme is a many-to-one mapping where identical subgraphs are guaranteed to have the same sequence representation. Due to coarsening of the search space, its size is drastically reduced. Second, most of the analysis happens in the sequence space. Thus, instead of enumerating subgraphs, we enumerate subsequences, which is computationally more tractable. In addition, the expensive subgraph isomorphisms are performed only on a minuscule portion that are considered candidates based on the sequence analysis.

1.4 Contributions of thesis

To summarize, the contributions of this thesis are as follows:

- We propose the idea of *communication motifs* to model the frequent human interaction patterns in social networks.
- We develop a technique called *COMMIT* to mine communication motifs in a scalable manner. COMMIT achieves scalability by mapping the interaction network into a more coarse-grained space of interaction sequences.
- Extensive experiments on three social network datasets show that COMMIT is more than an order of magnitude faster than baseline techniques. In addition, COMMIT is accurate and achieves F-scores in the range of [0.6,1] when compared to the ground truth. Finally, a qualitative analysis reveals communication motifs to be effective in characterizing the various patterns of human interactions and the crucial role that the underlying social network plays in its progression.

1.5 Outline of thesis

Rest of the thesis is organized as follows

- Chapter 2 presents the background and related work of mining communication motifs.
- Chapter 3 formalizes the problem definition of mining communication motifs.
- Chapter 4 explains the importance of moving from graph space to coarse grained sequence representation.
- Chapter 5 explains the mining of communication motif candidates in sequence space.
- Chapter 6 presents the COMMIT technique and connect the dots presented in chapters 4 and 5.
- Chapter 7 talks about the experimental validation of COMMIT on real world networks.
- Chapter 9 concludes the thesis and presents the future work.

CHAPTER 2

Background and Related Work

In this chapter, we discuss about network motifs, static and dynamic network motif detection algorithms and limitations of those existing algorithms.

2.1 Network Motifs

Generally, network motifs are statistically significant subgraphs that occur more frequently in the original network as compared to randomized networks.

The network motif detection basically consists of three steps:

1. Enumerate subgraphs in the network.
2. Detect isomorphic subgraphs and maintain their counts.
3. Calculate the subgraph significance.

Significant work has been done on algorithms to enumerate subgraphs. We have included few best algorithms in the related work. The second step of detecting isomorphic subgraphs is done with software packages such as Nauty [McKay and others, 1981], bliss [Junttila and Kaski, 2007]. The third step of calculating subgraph significance of subgraph varies based on the application. In this aspect, the proposed formulation of communication motifs and its scalability challenges have not been studied before.

For example, consider a small graph shown in Fig. 2.1 and its subgraph enumeration of size 4. For detecting network motif from a given network, one should enumerate subgraphs of all sizes 3,4,5 till the maximum specified motif size. As the size the network

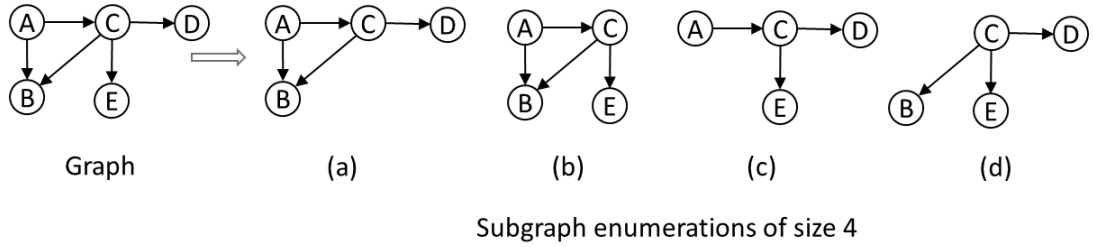


Figure 2.1: Graph and its subgraph enumeration of size 4.

increases, the number of possible subgraphs increases exponentially. Current state-of-the-art subgraph enumeration algorithms can enumerate subgraphs upto size 9 in small networks with number of nodes in order of hundreds of thousand.

Once the subgraph enumeration is done, the next step is detecting isomorphic graphs irrespective of graph node labels. For example in Fig 2.1, the subgraphs 2.1.a and 2.1.b are isomorphic to each other. Nauty [McKay and others, 1981] software compute canonical labels which is unique for a specific topology of graph. As a result, two isomorphic graphs will have same canonical labels.

The last step in detection of network motifs is determining the significance of network motifs. Quantifying the significance of motifs varies from application to application [Ciriello and Guerra, 2008]. Some research work use motif frequency as the significance factor while other work use z-score or p-value as significance factor. While using z-score as significance factor, underlying distribution of subgraphs is assumed to exhibit gaussian distribution.

Mining network motifs forms backbone of various applications such as spam detection [O'Callaghan *et al.*, 2012], protein-protein link prediction [Albert and Albert, 2004], analyzing human interactions in affiliation networks [Gallos *et al.*, 2012], network classification [Allan *et al.*, 2009, Ranu *et al.*, 2013, Ranu and Singh, 2009, Ranu *et al.*, 2011].

2.2 Static network motif detection algorithms

The problem of mining frequent subgraphs or motifs from single network is widely studied [Zhu *et al.*, 2007, Yan and Han, 2002, Borgwardt *et al.*, 2006, Elseidy *et al.*, 2014, Kuramochi and Karypis, 2001, Ketkar *et al.*, 2005, Bringmann and Nijssen, 2008].

There are number of surveys [Ciriello and Guerra, 2008, Bruno *et al.*, 2010] on static network motif detection algorithms. Static network motif detection algorithms can be roughly classified into two categories

2.2.1 Exact algorithms

Milo et al. [Milo *et al.*, 2002] studied motifs in complex networks like gene regulation,food webs,electronic circuits. The authors propose that network motifs might be the evolutionary backbone of a network[Milo *et al.*, 2002, Shen-Orr *et al.*, 2002] and can help understand basic information flow between local nodes. Milo et al. [Milo *et al.*, 2004] represent the structural and behavioral aspects of a network with significance profiles, which is a normalized vector of z-scores of motifs in the network. Zahra Razaghi et al. [Kashani *et al.*, 2009] propose a method for enumerating subgraphs called *kavosh*.

2.2.2 Approximate algorithms

- **Search algorithms based on sampling**

Kashtan et al. [Kashtan *et al.*, 2004] propose an edge sampling based algorithm, mfinder, to estimate subgraph counts. Sebastian Wernicke et al. [Wernicke, 2006] develop a sampling based motif detection algorithm, FANMOD, for estimating number of subgraphs and does not suffer from sampling bias as in mfinder [Kashtan *et al.*, 2004].

Since we have a dynamic interaction network, static motif detection techniques do not apply to our problem.

• **Apriori based motif detection algorithms**

Kuramochi et al. [Kuramochi and Karypis, 2005] detects frequent subgraphs from a single large sparse graph. The authors introduce apriori property with the help of maximum independent set on overlap graph. The other metrics like minimum image based support that are useful in introducing apriori property is reported by Bringmann et al. [Bringmann and Nijssen, 2008]. The large graphs processed by kuramochi et al. [Kuramochi and Karypis, 2005] posses number of edges in order of hundred of thousands while current large graphs posses billions of edges. Also kuramochi et al. [Kuramochi and Karypis, 2005] utilize node and edge labels to prune the search space but as stated earlier node/edge labels do not play a role in mining communication motifs. Elseidy et al. proposes GRAMI [Elseidy *et al.*, 2014] that models subgraph mining problem as constraint satisfaction problem. GRAMI finds the minimal set of instances to satisfy the frequency threshold and avoids the costly enumeration of all instances required by previous approaches. Again, GRAMI relies heavily on node labels as shown in Fig. 1.2.

2.3 Dynamic network motif detection algorithms

In the domain of dynamic graphs, a straightforward approach is to create a set of graphs H from the original dynamic network using time windows a certain length such as a month [Braha and Bar-Yam, 2009]. Each graph in set H then represents the graphs which consists of interactions between nodes occurring in a specific month time window. Motifs are then computed on each graph in H. Chechnik et al. [Chechik *et al.*, 2008] propose the idea of *activity motifs* to analyze transcription in yeast organism metabolism.

David Jurgens et al. [Jurgens and Lu, 2012] analyzed interactions of Wikipedia editors to identify significant patterns by constructing a temporal bipartite network. Kai Liu et al. [Liu *et al.*, 2012] proposed a finite mixture model to detect multiple stochastic motifs in network data but does not consider exact edge times while detecting stochastic motifs.

The closest works to our problem are proposed by Lauri Kovanen et al. [Kovanen *et al.*, 2011] and Zhao et al. [Zhao *et al.*, 2010]. The model proposed in [Kovanen *et al.*, 2011] fails on network where a person can communicate simultaneously (time overlapping edges) and hence fails to solve the proposed problem. The model in [Zhao *et al.*, 2010] is similar to ours but has the weakness of joining unrelated motifs together. More importantly, they do not propose any mining technique and the naive subgraph enumeration approach fails to scale.

CHAPTER 3

Problem Formulation

In this chapter, we formalize the problem of mining communication motifs. We represent a dynamic interaction network as a graph $G = (V, E)$ where V is a set of nodes and E is a set of edges.[1] An edge e_i is represented as (s_i, d_i, t_i) where s and d are the source and destination nodes, t is the time at which interaction happens.

3.1 Temporally Connected Graph

To formalize the concept of communication motifs, we have to first understand temporally connected graph (TCG). TCG is defined with the help of temporally related edges. Informally, two edges are related if they involve a common user and are close in time. An example of such related interactions is users A and B wishing user C on his/her birthday. Formally,

Definition 1. TEMPORALLY RELATED EDGES. *Two edges* $e_i = (s_i, d_i, t_i)$ *and* $e_j = (s_j, d_j, t_j)$ *are temporally related if they are adjacent, i.e.,* $\{s_i, d_i\} \cap \{s_j, d_j\} \neq \emptyset$*, and* $|t_i - t_j| \leq \Delta T$.

Definition 2. TEMPORALLY CONNECTED PATH. *Given a time window* ΔT *within which two adjacent interactions are considered related, a path* $\{e_1, \cdots, e_m\}$ *in a graph* G *is temporally connected, if the path is connected and* $\forall e_i, e_{i+1}$*, either* $0 < t_i - (t_j + \delta_j) < \Delta T$ *or* $0 < t_j - (t_i + \delta_i) < \Delta T$.

[1] More formally the interaction network is a multigraph since multiple interactions can take place between a pair of nodes. In order to simplify exposition, we refer to the interaction network as a graph.

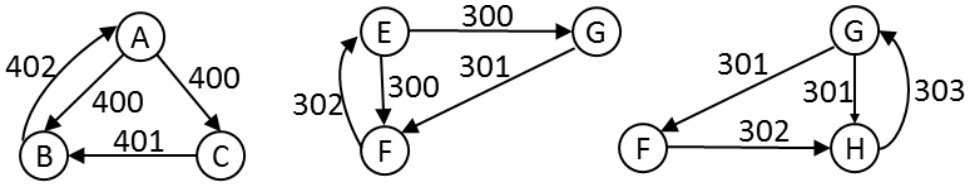


Figure 3.1: Embeddings of Motif 1 in Fig. 1.1(a)

More simply, a path is temporally connected if each pair of adjacent edges in the path are within ΔT of each other.

Definition 3. TEMPORALLY CONNECTED NODES. *Two nodes $n_i,\ n_j$ in a graph $G = (V, E)$ are temporally connected, if there exists a sequence of edges $\mathbb{P} = \{e_1, \cdots, e_m\} \in E$ such that $s_1 = n_i$, $d_m = n_j$, and $\forall e_i, e_{i+1} \in \mathbb{P}$, e_i and e_{i+1} are temporally related.*

Definition 4. TEMPORALLY CONNECTED GRAPH. *A connected interaction graph $G = (V, E)$ is temporally connected, if any pair of nodes $n_i,\ n_j$ in G is temporally connected.*

In essence, a temporally connected graph represents a group of related interactions that are connected by a common event.

Example 1. *Consider Fig. 3.1, which shows the embeddings of Motif 1 in Fig. 1.1(a). Let us focus on the first embedding involving nodes $\{A, B, C\}$. It is easy to see that at $\Delta T = 1$, all pairs of nodes are temporally connected and hence, the graph is temporally connected.*

Note that our definition of a temporally connected graph is different from the formulation of Zhao et al[Zhao *et al.*, 2010] and rectifies a weakness in their modeling.

3.1.1 Weakness of the model proposed by Zhao et al

Zhao et al.[Zhao *et al.*, 2010] use the term *communication graph* to denote a group of interactions that are related in their progression. In our work, we use the term temporally connected graph to model this same event. Informally, a communication graphs contain edges such that each edge e_i has at least one other adjacent edge e_j that is within ΔT from e_i. More formally, it is defined as follows.

Definition 5. COMMUNICATION GRAPH. *Given a time window ΔT, a* communication graph *is a collection of edges $S = \{e_1, \cdots, e_m\}$ such that $\forall e_i \in S$, there exists at least one edge $e_j \in S$, $i \neq j$, such that 1) $|\{s_i, d_i\} \cap \{s_j, d_j\}| > 0$ and 2) $|t_i - t_j| \leq \Delta T$.*

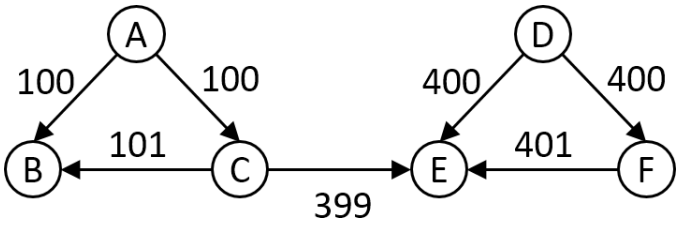


Figure 3.2: The scenario where two unrelated sets of interactions are clubbed together as related.

By the above definition, Fig. 3.2 is a communication graph since all edges are adjacent to at least one edge that is within ΔT. However, notice that the interactions involving $\{A, B, C\}$ are unrelated to those involving $\{D, E, F\}$. These two unrelated groups are clubbed together as related due to the edge between C and E.

In our definition, each pair of node in a temporally connected graphs needs to be temporally related. Fig .3.2 is not temporally connected since E is not temporally related to any of the nodes in $\{A, B, C\}$, which conforms with the general intuition.

3.2 Temporal Isomorphism

We first define a partial ordering among edges $e_i = (s_i, d_i, t_i)$ and $e_j = (s_j, d_j, t_j)$ as $e_i < e_j$ if and only if (*iff*) $t_i < t_j$. Based on this ordering, we next define temporal isomorphism

Definition 6. TEMPORAL ISOMORPHISM.

A dynamic graph $S_1 = (V_{s_1}, E_{s_1})$ is temporally isomorphic to $S_2 = (V_{s_2}, E_{s_2})$ if and only if there exists a bijection $f : E_{s_1} \rightarrow E_{s_2}$ satisfying

(1) if $(s, d, t) \in E_{s_1}$ then $f(s, d, t) \in E_{s_2}$

(2) if $e_i, e_j \in E_{s_1}$ and $e_i < e_j$ then $f(e_1) < f(e_2)$.

It is easy to see that the embeddings in Fig. 3.1 are all temporally isomorphic to each other. Analogous to this definition, a graph H is a *temporal subgraph* of G, if G contains a subgraph G', such that G' is temporally isomorphic to H.

3.3 Support of subgraph

The *support* $sup(H)$ of a recurring temporal subgraph H is its number of embeddings in the interaction network. As illustrated earlier, Motif 1 in Fig. 1.1(a) has a support of 3. Note that two embeddings of a motif could overlap and may not necessarily be disjoint. Due to such overlaps, the *apriori property* of a subgraph having a support at least as large as any of its supergraph is violated.

3.3.1 Violation of apriori property

The apriori property expresses a monotonic decrease of an evaluation criterion accompanying the progress of a sequential pattern. In the context of support counting for graphs, the apriori property states that the support of a graph is at least as large as the support of any of its supergraphs.

Now, consider the interaction network in Fig. 3.3. At $\Delta T = 2$, Motif 1 has a support of 1. However, Motif 2, in spite of being a supergraph of Motif 1, has a support of 3. This violation of apriori property happens since the embeddings of Motif 2 overlap with each other and share the triangular component involving nodes $\{A, B, C\}$.

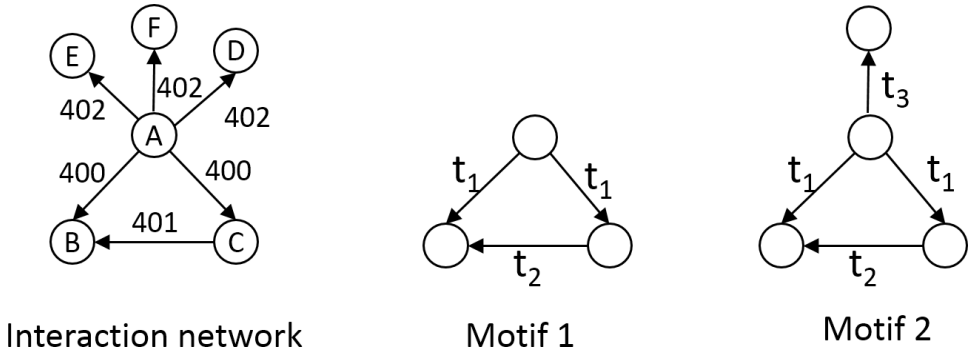


Figure 3.3: Violation of apriori property due to overlap.

3.4 Communication Motif and Queries

We now formally define *communication motif* as the following.

Definition 7. COMMUNICATION MOTIF. *Given a dynamic interaction network G, a minimum support threshold τ and a ΔT, a motif, or a recurring connected temporal subgraph of G, H, is a* communication motif *if its support $sup(H) \geq \tau$.*

Our goal is now to solve the following problems.

Problem 1. RANGE QUERY. *Mine all communication motifs in the given interaction network for a user-specified ΔT and τ.*

Problem 2. TOP-k QUERY. *Given a dynamic interaction network G, a value k and a ΔT, mine the communication motifs with the top-k highest supports.*

COMMIT solves both the mining problems in a scalable manner. For simplicity, our illustrative examples assume Problem 1.

CHAPTER 4

Mapping graphs to sequences

In this chapter, we explain the need and process of conversion of graphs to sequences in COMMIT technique.

As discussed earlier, mining communication motifs is hard since the search space is exponential. In addition, counting support of a subgraph requires us to perform subgraph isomorphism, which is NP-complete[Zeng *et al.*, 2009]. To tackle this bottleneck, COMMIT first maps the dynamic network from the graph space to a sequence space.

4.1 Conversion conditions

Let $\mathcal{M} : G \rightarrow S$ be a function to map graph G to its sequence space representation S. The goal in this conversion procedure is to map the dynamic network into a contractive space, such that the following conditions hold.

- If graph G is temporally isomorphic to graph G', then $\mathcal{M}(G) = \mathcal{M}(G')$
- If H is a temporal subgraph of G, then $\mathcal{M}(G)$ "contains" $\mathcal{M}(H)$. Indeed, we need to define "contains" more formally.

4.1.1 Temporal isomorphic condition

If we discard the temporal constraints, then the first condition can be satisfied using *graph invariants*.

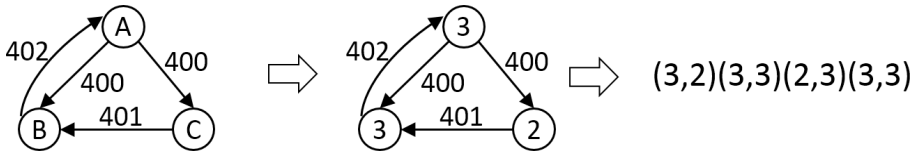


Figure 4.1: Sequence representation of a graph.

Definition 8. GRAPH INVARIANT. *A* graph invariant *is a function f, such that $f(G_1) = f(G_2)$, whenever G_1, and G_2, are isomorphic graphs.*

Graph invariants are properties of graphs that are invariant under graph isomorphisms. Examples of graph invariants are number of nodes or edges, degree sequence, diameter, canonical labeling of the adjacency matrix, etc. [Yan and Han, 2002]. To satisfy condition 1 in the presence of temporal constraints, we generate a degree sequence as our graph invariant. Specifically, we map a graph G to a sequence in the following manner. First, we assign the degree of a node as its label. Let $l(n)$ denote the label of node n. Extending the same procedure, for each edge $e = (s_i, d_i, t_i)$, we label $l(e) =$"$l(s_i)$, $l(d_i)$". Now, we extend the partial ordering defined in Sec. 3.2 to a *total ordering*. Specifically, if $t_i < t_j$, then $e_i < e_j$. Else, if $t_i = t_j$, $e_i < e_j$, if $l(e_i) < l(e_j)$, i.e., the label of e_i is lexicographically smaller (we break ties based on edge ids). Finally, the mapping $\mathcal{M}(G)$ of a graph G containing edges $\{e_1, \cdots, e_m\}$ where $e_i < e_{i+1}$, is "$l(e_1)\ l(e_2) \cdots l(e_m)$."

Example 2. *Fig. 4.1 shows the sequence representation of the first graph in Fig. 3.1. It is easy to see, that since the other two graphs in Fig. 3.1 are temporally isomorphic to the first graph, their sequence representations are also identical.*

We use the notation $S[i]$ to denote the label of the i_{th} edge in sequence S.

4.1.2 Temporal subgraph condition

After satisfying condition 1, we now focus on satisfying condition 2, which is to detect the presence of a subgraph just from a sequence space analysis. Let us revisit the first graph in Fig. 3.1. We denote this graph as G. If we remove the edge $(B, A, 402)$ from G to create graph H, then $\mathcal{M}(H) = (2,2)\ (2,2)\ (2,2)$. Clearly, $\mathcal{M}(H)$ is not a subsequence of $(3,2)\ (3,3)\ (2,3)\ (3,3)$ although H is a temporal subgraph of G. Thus, the simple sub-sequence relationship does not satisfy condition 2. The event $H \subseteq G$ does not guarantee that an edge label in H is also present in G. However, given that we use degree as node labels, for any edge label $l(e) = (a, b)$ in H, there must an edge e' in G where $l(e') = (c, d)$ and $c \geq a$ and $d \geq b$. We formalize this intuition by defining the notion of *edge containment*.

Definition 9. EDGE CONTAINMENT. *An edge e_i with label $l(e_i) = (a_i, b_i)$ is* contained *in edge e_j with $l(e_j) = (a_j, b_j)$ if $a_i \leq a_j$ and $b_i \leq b_j$. This relationship is denoted as $(a_i, b_i) \sqsubseteq (a_j, b_j)$.*

In our definition, edge-containment is only dependent on the node degrees. The semantic labels of edges and nodes, such as node type, ID, etc., do not play any role. However, if required, the proposed technique can easily be extended to incorporate such semantic labels as well. Specifically, we not only need to look for degree containment while comparing edges, but also ensure that the edges being compared, and their constituent nodes, have the same semantic labels.

Next, we define the notion of *subsequence* in the sequence space as following.

Definition 10. SUBSEQUENCE. *A sequence $\alpha =< \alpha_1, \alpha_2, ..., \alpha_m >$ is* subsequence *of sequence $\beta =< \beta_1, \beta_2, ..., \beta_n >$ iff $\exists i_1, i_2, .., i_m$ such that $1 \leq i_1 < i_2 < ... < i_m \leq n$ and $\alpha_1 \sqsubseteq \beta_{i_1}$, $\alpha_2 \sqsubseteq \beta_{i_2}$, ... $\alpha_m \sqsubseteq \beta_{i_m}$. This relationship is denoted as $\alpha \sqsubseteq \beta$.*

More simply, sequence $\alpha \sqsubseteq \beta$, if each of the edges in α is contained in some edge in β, while also maintaining the ordering of edges in α. The support of a subsequence S is defined analogously to that of a subgraph and is also denoted as $sup(S)$. A subsequence S is *frequent*, if $sup(S) \geq \tau$

Theorem 1. *If graph $H = (V_H, E_H)$ is a temporal subgraph of $G = (V_G, E_G)$, then $\mathcal{M}(H) \sqsubseteq \mathcal{M}(G)$*

PROOF: Let $E_G = \{e_{g_1}, \cdots, e_{g_n}\}$ and $E_H = \{e_{h_1}, \cdots, e_{h_m}\}$ where $m \leq n$. We know $E_H \subseteq E_G$. Let function $f : E(H) \rightarrow E(G)$ be the bijection. We have $f(e_{h_i}) = e_{g_k}$, $\forall i$ s.t. $1 \leq i \leq m$ and $1 \leq k \leq n$. Since H is also a temporal subgraph of G, from the total ordering on edges, we can claim that if $e_{h_i} < e_{h_j}$ then $f(e_{h_i}) < f(e_{h_j})$. As a result, in sequence space representation $\mathcal{M}(H)$, $\forall e_{h_i}, e_{h_j} \in E_H$, if $l(e_{h_i})$ occurs before $l(e_{h_j})$, then $l(f(e_{h_i}))$ occurs before $l(f(e_{h_j}))$ in $\mathcal{M}(G)$. Let $f(e_{h_i}) = e_{g_k}$, where $l(e_{h_i}) = (l(s_{h_i}), l(d_{h_i}))$ and $l(e_{g_k}) = (l(s_{g_k}), l(d_{g_k}))$. Now, since $H \subseteq G$, it is guaranteed that the source and destination degrees of e_{h_i} are less than or equal to that of e_{g_k} in G. Hence $l(s_{h_i}) < l(s_{g_k})$ and $l(d_{h_i}) < l(d_{g_k})$. Consequently, $e_{h_i} \sqsubseteq e_{g_j}$. Since this holds for any pair of edges in H, $\mathcal{M}(H) \sqsubseteq \mathcal{M}(G)$. □

Corollary 1. *If the support of a graph H in dynamic network G is larger than τ, then the support of $\mathcal{M}(H)$ in $\mathcal{M}(G)$ is also larger than τ.*

From Theorem 1, the problem of mining temporal subgraphs with support above τ translates to mining subsequences with support above τ. Indeed, there could be false positives where two different graphs are mapped to the same sequence. To prune out such false positives, the frequent subsequences are mapped back to the graph space to compute the true answer set. From Corollary 1, false negatives are not possible. With this insight, we next focus on frequent subsequence mining.

CHAPTER 5

Frequent subsequence mining

In this chapter, we explain the algorithm of detecting temporally connected components and propose frequent subsequence mining algorithms. The mined frequent subsequences could potentially represent the embeddings of communication motifs in the network.

5.1 Temporal connected component

Given a dynamic interaction network $G = (V, E)$ and a ΔT, we first identify *the temporally connected components* in G.

Definition 11. TEMPORALLY CONNECTED COMPONENT.
Given an interaction network G and ΔT, let H be a temporally connected subgraph of G. H is a temporally connected component if no supergraph $H' \supseteq H$ exists such that H' is temporally connected and $H' \subseteq G$.

Example 3. *The temporally connected components of the network in Fig. 1.1(a) at $\Delta T = 1$ are shown in Fig. 5.1.*

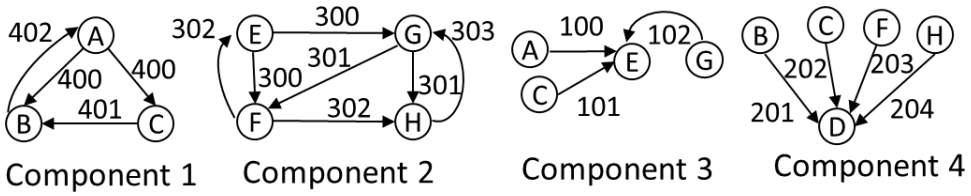


Figure 5.1: The temporally connected components in Fig. 1.1(a).

The pseudocode to identify the temporally connected components is shown in Alg. 1. Identifying the temporally connected components in a graph $G = (V, E)$ can be computed in $O(E)$ time since no edge is processed more than once.

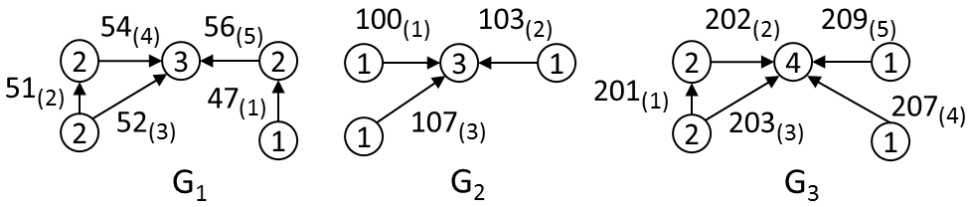
Algorithm 1 TCCDetect($N = (V, E)$, ΔT)

Require: An interaction network N, temporal threshold ΔT.

Ensure: Return all temporally connected networks in N at ΔT.

```
1: Mark all edges in E as not processed.
2: TCC ← ∅
3: while All edges are NOT processed do
4:     Create an empty graph G.
5:     Choose a random unprocessed edge e, push it on S.
6:     while S is not empty do
7:         Pop edge e from S.
8:         Add edge e in Graph G.
9:         Mark e as processed.
10:        If the time difference between e and its adjacent edge e′ is within ΔT, then
           push e′ on S.
11:    TCC ← TCC ∪ G
12: return TCC
```

From the construction of temporally connected components, it is guaranteed that a communication motif cannot span across two different components. However, it is possible for a communication motif to be contained in multiple components. For example, Motif 1 in Fig. 1.1(b) occurs once in component 1 and twice in component 2 with an overall support of 3. In COMMIT, we map each of the connected components into the sequence space. Following this transformation, our task is to mine the frequent subsequences with support of at least τ. A subsequence may repeat across sequences as



SID	Sequence
1	(1,2) (2,2) (2,3) (2,3) (2,3)
2	(1,3) (1,3) (1,3)
3	(2,2) (2,4) (2,4) (1,4) (1,4)

Figure 5.2: The connected components of an interaction network and their corresponding sequence representations. In each edge of the graph, along with the timestamp, we also show its rank (or position in the sequence representation) based on the total ordering.

well as within a sequence. Mining such frequent subsequences in a sequence database has been studied, and *CloGSgrow* [Ding *et al.*, 2009] is the state-of-the-art technique for this purpose. CloGSgrow is an extension of PrefixSpan[Pei *et al.*, 2001] and adopts a similar search space exploration strategy. To give an overview of CloGSgrow, it starts from frequent patterns of size one, and looks for *extensions* to grow one-sized patterns to two-sized frequent patterns. This process continues iteratively to build larger frequent patterns till no more extensions are possible. Unfortunately, due to altering the definition of subsequence, CloGSgrow do not work in our scenario.

5.2 Counting support of a subsequence

Assume the graphs in Fig. 5.2 are the connected components of the network we are mining. Their corresponding sequence representations are shown in the Sequence DB table in Fig. 5.2. Now, let $P = l(e_1)\ l(e_2)\ \cdots\ l(e_n)$ be a sequence over n edges. If $P \sqsubseteq S_i$ for some sequence S_i in Sequence DB, then we represent this occurrence as $(i, <$

$l_1, l_2, \cdots, l_m >$) where i is the ID of S_i (or the corresponding connected component of the network) and l_j is the position of the edge in S_i that contains the j^{th} edge of P. For example, consider the sequence $P = (1,3)(1,3)(1,3)$. P occurs thrice in S_3 of Fig. 5.2. These three instances of P in S_3 correspond to the instances with ID 3 in the $SeqDB(P)$ table of Fig. 5.3. We use the notation $SeqDB(P)$ to represent the set of all instances of P in the sequence database.The first two rows in $SeqDB(P)$ correspond to P's instances in S_1 and S_2. The instance $(3, < 2, 3, 4 >)$, denotes that the first, second and third edges of P are mapped to the second, third and fourth edges in S_3 (or G_3). Since an instance $(i, < l_1, l_2, \cdots, l_m >)$ of a subsequence uniquely identifies its mapped edges in component G_i, it is easy to derive the subgraph that is induced by this instance.

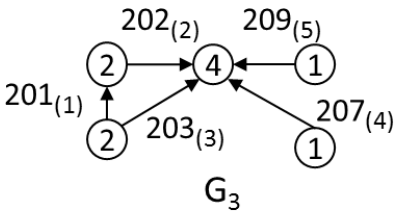
Two instances of a sequence P in S_i are called *identically overlapping* if there exists an edge in P that is mapped to the same edge in S_i in both instances. The formal definition is as follows.

Definition 12. IDENTICALLY OVERLAPPING INSTANCES. *Let two instances of a sequence $P = l(e_1) \cdots l(m)$ in $SeqDB(P)$ be $(i, < l_1, \cdots, l_m >)$ and $(i', < l'_1, \cdots, l'_m >$). These two instances are identically overlapping if (1) $i = i'$ and (2) $\exists j, 1 \leq j \leq m$ such that $l_j = l'_j$*

Example 4. *The third and fourth instances of P in $SeqDB(P)$ in Fig. 5.3 are identically overlapping since they both correspond to instances in S_3 and the first two edges of P are mapped to the second and third edge of S_3 in both instances. On the other hand, the third and the fifth instances are not identically overlapping. Note that the third and the fifth instances also overlap. However, they do not overlap in the same position and hence, they are non-identically overlapping.*

Theorem 2. *In the presence of identically overlapping instances, computing $SeqDB(P)$ is NP-hard.*

PROOF: [Ding *et al.*, 2009] proves that when identically overlapping instances are



S_3 = (2,2) (2,4) (2,4) (1,4) (1,4)

Pos: 1 2 3 4 5

P = (1,3) (1,3) (1,3)

SeqDB (P)
(1, < 3, 4, 5 >)
(2, < 1, 2, 3 >)
(3, < 2, 3, 4 >)
(3, < 2, 3, 5 >)
(3, < 3, 4, 5 >)

Support set-1 of P
(1, < 3, 4, 5 >)
(2, < 1, 2, 3 >)
(3, < 2, 3, 4 >)
(3, < 3, 4, 5 >)

Support set-2 of P
(1, < 3, 4, 5 >)
(2, < 1, 2, 3 >)
(3, < 2, 3, 5 >)

Figure 5.3: Demonstrates the instance representation of subsequence $P = (1,3)(1,3)(1,3)$ in S_3. $SeqDB(P)$ lists all instances of P in the sequence database. Furthermore, two possible support sets of P are also listed.

allowed in the "traditional" definition of subsequence, the problem is NP-hard. Now, if sequence α is a "traditional" subsequence of β, then $\alpha \sqsubseteq \beta$ by Definition 10 as well. $\square$.

Due to Theorem 2, counting all instances of a subsequence P is not tractable. Hence, we only count those instances of P that are not identically overlapping. Hereon, any reference to an instance of a subsequence P is implicitly assumed to be a non-identically overlapping instance and the *support set* of P is defined analogously.

Definition 13. SUPPORT SET. *The support set of a subsequence P with respect to a database of sequences contains only those instances of P that are non-identically overlapping.*

The support sets of P for the components in Fig. 5.2 are shown in Fig. 5.3. Notice that for a given subsequence P, multiple support sets can be computed. To best approximate $SeqDB(P)$, we need to compute the largest support set support set SS^*, where

$$SS^* = \arg\max_{SS}\{|SS| \; |SS \subseteq SeqDB(P) \text{ is a support set of } P\}$$

The support of P is therefore $sup(P) = |SS^*|$. We discuss how to compute SS^* in Sec. 5.3.2. Regardless of whether the support set is the largest or not, it satisfies the *apriori* property.

Theorem 3. APRIORI PROPERTY OF SUPPORT. *Assume we are given a database of sequences SeqDB corresponding to each connected component of an interaction network. For any two sequences P and P', if $P \sqsubseteq P'$, then $sup(P) \geq sup(P')$.*

PROOF: We split the proof into two cases based on the different ways a sequence P can be a subsequence of P'

Case 1: $\forall j, \; P[j] \sqsubseteq P'[j]$ and $|P| = |P'|$

Let us represent

$P' = l(\overline{e_1}), l(\overline{e_2}), \cdots, l(\overline{e_m})$

$P = l(e_1), l(e_2), \cdots, l(e_m)$.

Note that each instance $I' = (x, < l_1, \cdots, l_m >)$ of P' is also an instance of P. Hence, for any given support set SS' of P', we can construct a support set SS of P containing all instances in SS'. Hence, $sup(P) \geq sup(P')$.

Case 2: $|P| \leq |P'|$

Let us assume

$P' = l(\overline{e_1}), \cdots, l(\overline{e_{i-1}}), l(\overline{e_i}), l(\overline{e_{i+1}}), \cdots, l(\overline{e_m})$

$P = l(e_1), \cdots, l(e_{i-1}), l(e_{i+1}), \cdots, l(e_m)$.

such that, $\forall j \neq i, \; l(e_j) \sqsubseteq l(\overline{e_j})$ and $|P'| - |P| = 1$. Now for any instance $I' = (x, < l_1, \cdots, l_{i-1}, l_i, l_{i+1}, \cdots, l_m >)$ of P' in its support set, we can create an in-

stance $I = (x, < l_1, \cdots, l_{i-1},\ l_{i+1}, \cdots, l_m >$ of P in P's support set. Thus, $sup(P) \geq sup(P')$.

It is easy to see that the same strategy can be generalized when $|P'| - |P| > 1$. More specifically, let $|P| = m$, $|P|' = n$, and $m < n$. Since, $P \sqsubseteq P'$, let $I'_P = (P'_{id}, < p_1, \cdots, p_m >)$ be an instance of P in P'. Recall from the definition of instance that P'_{ID} denotes the ID of P' and p_i denotes that the i^{th} edge of P is mapped to the p_i^{th} edge in P' . Therefore, for any instance, $I' = (x, < l_1, l_2, \cdots, l_n >)$ in the support set of P', we can create a support set of P containing instance $I = (x, < l_{p_1}, l_{p_2}, \cdots, l_{p_m} >)$. Thus, $sup(P) \geq sup(P')$. □

5.3 The sequence growth approach

Sequence growth is a popular strategy to search for sequences in the presence of apriori property[Yan and Han, 2002, Ding *et al.*, 2009, Pei *et al.*, 2001].

Definition 14. SEQUENCE GROWTH. *Let a subsequence $P = l(e_1), l(e_2), \cdots, l(e_m)$ be extended by the label of an edge e as $l(e_1), l(e_2), \cdots, l(e_m), l(e)$. This extension is known as sequence growth. Sequence growth is denoted by $P \circ e$. Through apriori property, if $sup(P) < \tau$, then $sup(P \circ e) < \tau$. Similarly, if $sup(l(e)) < \tau$, $sup(P \circ e) < \tau$.*

Sequence growth outlines the strategy that we can start with labels of frequent edges and keep extending them to larger sequences till the sequence becomes infrequent. The key question therefore is how do we identify the extensions?

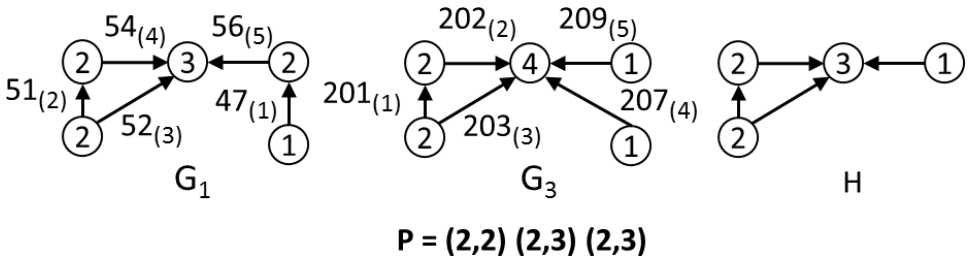


Figure 5.4: Illustration of the need for EXTENSIONMINER.

5.3.1 Identifying edge extension candidates

In traditional subsequence mining such as CloGSgrow, given a subsequence $P = l(e_1) \cdots l(e_m)$, first, the support set of P is identified. Let $\mathbb{S}$ be the sequences in the database containing P. The possible extensions of P here are those one-sized items (edge labels in our case) that occur not less than τ times after P's instances in the sequences in $\mathbb{S}$. Since we also have the temporal connectivity constraint based on ΔT, we need to look for only those edges following P that are within ΔT from e_m. In our problem, however, this strategy of CloGSgrow does not work.

To illustrate, let us revisit the components in Fig. 5.2. Let us consider the subsequence $P = (2, 2)(2, 3)(2, 3)$. P has support 3 because it occurs twice (non-identically) in SID 1 and once in SID 3. At $\Delta T = 10$, the possible extensions are $(2, 3)$ in SID 1 and two $(1, 4)$ labels in SID 3. At $\tau = 2$, only $(1, 4)$ is classified as frequent, and we would generate the subsequence $P_1 = (2, 2)(2, 3)(2, 3)(1, 4)$. P_1 has a support of 1; it has two instances in S_3, but they are identically overlapping. When any of these instances is mapped to the graph space, P_1 corresponds to graph H in Fig. 5.4. Notice that H is also a temporal subgraph of G_1, but we are unable to detect it. Now, instead of extending P with $(1, 4)$, if we extend with $(1, 3)$, we will generate $P_2 = (2, 2)(2, 3)(2, 3)(1, 3)$. The instances of P_2 in S_3 is identical to that of P_1. Furthermore, P_2 also has an instance in G_1, and all these instances correspond to H. In other words, P_2 is more accurately able to discover the common subgraph H and that is because $P_2 = M(H)$.

Clearly, we cannot overlook extensions such as $(1, 3)$, which would happen with the traditional sequence growth approach. The traditional approach fails in our problem since we need to look for edge label containment instead of edge label matching. Thus, the possible edge extensions are not only the frequent edges following P, but also any edge that is contained frequently in the edges following P. Going back to our example, edge label $(1, 3)$ is not present explicitly in any of the edges following P. However, $(1, 3)$ is contained in the edges $(1, 4)$ and $(2, 3)$, and therefore, is a valid candidate for expansion with support of 3. To formalize this intuition, we define an *edge extension candidate* as follows.

Definition 15. EDGE EXTENSION CANDIDATE. *Let $\mathbb{E}$ be the set of all edge labels that occur within ΔT from the edge e_m in subsequence $P = l(e_1), l(e_2), \cdots, l(e_m)$. An edge label $l = (s, d)$ is an edge extension candidate if $sup(l) \geq \tau$, where $sup(l) = |\{l \sqsubseteq e | e \in \mathbb{E}\}|$.*

One can immediately realize that at $\tau = 3$, along with $(1, 3)$, $(1, 2)$ and $(1, 1)$ are also valid extensions since they occur in the same edges where $(1, 3)$ occurs. As a result, extension of P with either $(1, 3)$, $(1, 2)$ or $(1, 1)$ will generate a new subsequence with the exact same support set. When support sets of two subsequences are identical, the graphs represented by the subsequences are also identical. More specifically, we claim the following.

Theorem 4. *Let $\alpha \sqsubseteq \beta$ be two subsequences with the same support sets. When this occurs, any subgraph represented by subsequence α will also be represented by subsequence β.*

PROOF: Let the support sets of α and β be $SS_\alpha = \{i_\alpha^{(k)}, < l_{1_\alpha}^{(k)}, \cdots, l_{n_\alpha}^{(k)} >\}$ and $SS_\beta = \{i_\beta^{(k)}, < l_{1_\beta}^{(k)}, \cdots, l_{n_\beta}^{(k)} >\}$ respectively where $1 \leq k \leq sup(\alpha)$. Now, $SS_\alpha = SS_\beta$ implies $i_\alpha^{(k)} = i_\beta^{(k)}$ and $l_{j_\alpha}^{(k)} = l_{j_\beta}^{(k)}$ for $\forall j,\ 1 \leq j \leq n$ and $\forall k,\ 1 \leq k \leq$

Algorithm 2 ExtensionMiner $(el, \mathbb{S}, b, \tau)$

Require: $\mathbb{S}$ is support set of edge label el, b is starting position, τ is support threshold.
Ensure: $\mathbb{E}$ is set of all frequent edges labels.

```
1: 𝔼 = 𝔼 ∪ el
2: for i = b to 1 do
3:    𝕊′ ={ y |y ∈ 𝕊, y_i > el[i] }.
4:    if |𝕊′| < τ then
5:       continue
6:    el′ = floor(𝕊′)
7:    if ∃j < i such that el′[j] > el[j] then
8:       continue
9:    ExtensionMiner(el′, 𝕊′, i, τ)
```

$sup(\alpha)$. Since both the connected component IDs and the edge positions within those components are identical for support sets of α and β, any subgraph represented by an instance of support set of α will be represented by β as well. □

From Theorem 4, it becomes critical to prune out redundant extensions that generate duplicate support sets. If we are unable to detect such redundant extensions, then we will not only be generating subsequences pointing to same subgraphs, but also further expand these subsequences using sequence growth creating an exponential explosion in search space redundancy. To guard the mining procedure from these spurious extensions, we define the concept of *closed* edge extensions.

Definition 16. CLOSED EDGE EXTENSION CANDIDATE.
An extension candidate with edge label $l = (s, d)$ is closed if $sup(l) \geq \tau$, and there does not exist another edge extension candidate with label l' such that $l \sqsubseteq l'$ and $sup(l') = sup(l)$.

It is easy to see, that when only closed edge extensions are allowed, all extensions are non-redundant. Going back to our previous example, among extension candidates $(1, 3)$, $(1, 2)$ and $(1, 1)$, only $(1, 3)$ is closed.

The above two observations significantly complicate the extension identification step. Not only do we need to search for the edges that follow P frequently, but also look for all extensions that occur within those edges. Furthermore, after finding all such possible extensions, we need to prune those that are not closed. To analyze the computational burden of this task, assume the maximum degree of a node is δ. Then, the highest possible edge label is (δ, δ). Such an edge label contains δ^2 other edge labels within it and therefore creates an explosion in the extension search space. Now among these δ^2 possible extensions, we need to prune those that are not closed, which makes the computation cost $O(\delta^4)$. Clearly, a naive algorithm to perform this task is not feasible. To overcome this bottleneck, we design the *ExtensionMiner* algorithm.

To explain EXTENSIONMINER, we first define the *floor* of edge labels.

Definition 17. FLOOR. *Floor of a set of edge labels* $\{l(e_1), \cdots, l(e_n)\}$ *is an edge label* $l(e) = (s, d)$ *such that* $s = min(l(s_1), \cdots, l(s_n))$ *and* $d = min(l(d_1), \cdots, l(d_n))$*, where* s_i *and* d_i *are the source and destination of edge* e_i*.*

For an edge label $l(e) = (s, d)$, we use $l(e)[0]$ to denote s and $l(e)[1]$ to denote d. Alg. 2 presents the EXTENSIONMINER algorithm. Fig. 5.5 presents a running example of the algorithm. EXTENSIONMINER identifies *closed* edge-label extensions in a bottom-up, depth-first manner. For a subsequence $P = l(e_1) \cdots l(e_m)$, we compute the collection $\mathbb{S}$ of all edge labels occurring after the last edge $e_m \in P$, but within ΔT from e_m. Note that $\mathbb{S}$ may contain the same edge label multiple times. Such a scenario is shown in the illustration of EXTENSIONMINER in Fig. 5.5, where the labels of S_1 and S_6 are identical.

At the start, floor of all edge labels $e = floor(\mathbb{S})$ is calculated and EXTENSIONMINER (e, $\mathbb{S}, 0, \tau$) is called. e represents the edge label contained in all labels in $\mathbb{S}$ and has support $|\mathbb{S}|$. If $|\mathbb{S}| \geq \tau$, we store e as an extension candidate (line 1). In each of the subsequent steps, EXTENSIONMINER moves on to a state with a smaller $\mathbb{S}$ and a larger

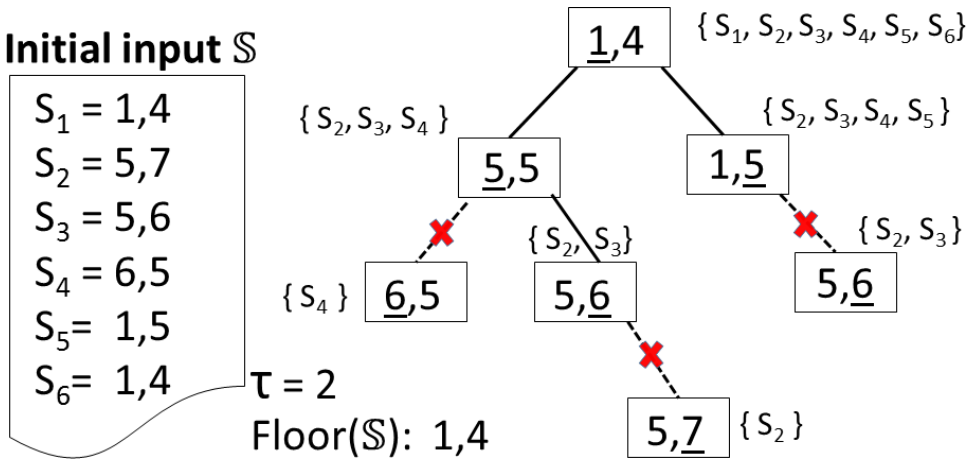


Figure 5.5: A running example of EXTENSIONMINER. The closed edge extensions from the given collection $\mathbb{S}$ of all edge labels correspond to the floors in the non-leaf states. Specifically, $(1, 4)$, $(5, 5)$, $(5, 6)$, and $(1, 5)$. The underlined dimension indicates the value of b in that state.

floor e. Specifically, for all indices of e (line 2), a new set $\mathbb{S}'$ is created (line 3) containing all values greater than $e[i]$. This process continues till we reach a state with $|\mathbb{S}| < \tau$ (lines 4-5). In addition, we do not expand on states that have already been visited in an earlier branch of the search tree (lines 7-8, Ex. child $(5, 5)$ on right branch of root in Fig. 5.5). Thus, EXTENSIONMINER is *correct* in identifying all possible closed edge extensions, *non-redundant* in pruning out all duplicate states at the earliest stage, and *efficient* since it performs the minimum number of computations required to identify all closed extensions.

5.3.2 Computing the largest support set

ExtensionMiner allows us to employ the sequence growth approach. More specifically, given a frequent subsequence P, we identify the possible extensions using EXTENSIONMINER and generate new subsequences of a larger size. The supports of these new subsequences are then computed to verify if they are above τ. Now, recall that in Sec. 5.2,

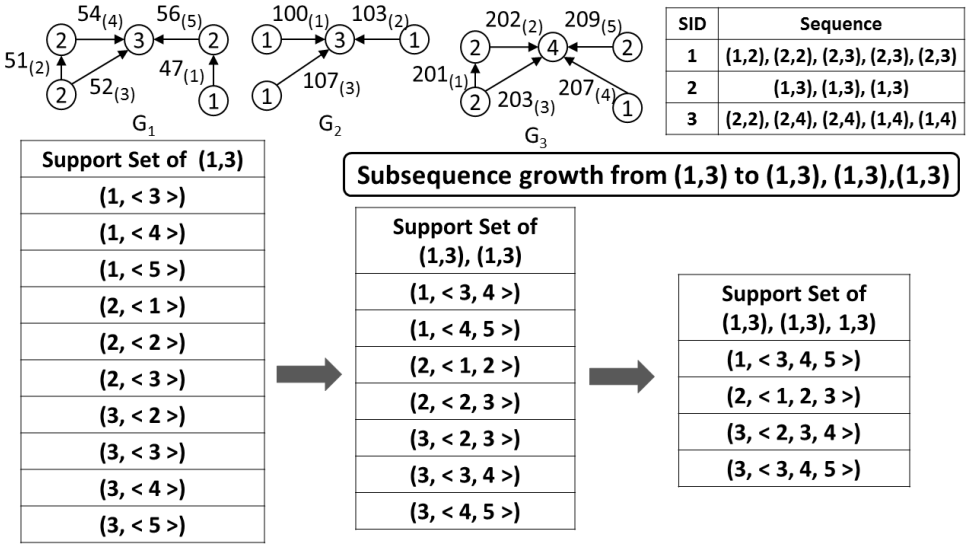


Figure 5.6: Illustration of sequence growth from $(1,3)$ to $(1,3)(1,3)(1,3)$. The support sets are maintained in right shift order, which allows polynomial-time support counting.

we realized that a subsequence can have multiple support sets, and we need to identify the largest one. Towards that goal, we use the greedy polynomial-time support counting strategy outlined in CloGSgrow[Ding *et al.*, 2009]. Here, we briefly summarize the algorithm. The correctness proofs are available in CloGSgrow[Ding *et al.*, 2009].

First, we define the concept of *right shift order*.

Definition 18. RIGHT-SHIFT ORDER. *Instances of a sequence P in its support set are said to be in right-shift order, if one instance $(i, < l_1, \cdots, l_m >)$ is ordered before another instance $(i', < l_1^{'}, \cdots, l_m^{'} >)$ when (1) $i < i'$ or (2) $i = i'$ and $l_m < l_m^{'}$.*

For example, instance $(3, < 2, 3, 4 >)$ followed by $(3, < 3, 4, 5 >)$ is in right-shift order.

To illustrate the greedy support counting algorithm, let us revisit the sequences generated out of the connected components in Fig. 5.2 (also shown in Fig. 5.6). Consider the subsequence $P = (1,3)(1,3)(1,3)$. With the sequence growth technique, the gen-

eration of this subsequence would start from the edge label $(1, 3)$.

1. Find all instances of subsequence $(1, 3)$ in the sequence database and store them in right-shift order. Since there is no scope of overlap for single edge subsequences, the computation is trivial. The first table in Fig. 5.6 denotes all instances of $(1, 3)$. Next, EXTENSIONMINER identifies all possible extensions and let us assume that $(1, 3)$ is one such extension.

2. Our goal is now to compute the largest support set of $(1, 3)$ $(1, 3)$ from $(1, 3)$. To locate the first instance of $(1, 3)(1, 3)$, the search starts from the first entry in support set of $(1, 3)$. More specifically, we extend instance $(1, < 3 >)$ of subsequence $(1, 3)$ to instance $(1, < 3, 4 >)$ of subsequence $(1, 3)(1, 3)$. Next, we move to the second instance of $(1, 3)$, which is $(1, < 4 >)$, to identify the next instance of $(1, 3)(1, 3)$. Now, note that due to right shift order, we need to search for the extension $(1, 3)$ only from position 5 onwards in SID 1. This follows from the fact that the preceding instance of $(1, 3)(1, 3)$ ends at position 4 in SID 1. As a result, any non-identically overlap instance can occur only beyond position 4. This observation lies at the core of obtaining a polynomial time algorithm in identifying the largest support set.

3. From instance set of subsequence $(1, 3)(1, 3)$, we follow the same strategy to find non-identically overlapping instances of $(1, 3)$ $(1, 3)(1, 3)$ in right-shift order.

The above example illustrates the intuition behind the greedy strategy. Since this algorithm is not a core contribution of our work, we present the formal pseudocode in Appendix 6.2.

Revisiting the example in Fig. 5.6, one can see that the 4-node graph G_2 is described by $(1, 3)(1, 3)(1, 3)$ in the sequence space. In the graph space, $sup(G_2) = 5$. The approximated support in the sequence space is 4. In other words, we are able to achieve a good approximation without performing any of the costly steps such as sub-

graph enumeration and subgraph isomorphism. This allows us to achieve scalability without compromising significantly on quality. We verify this empirically in Chapter 7.

CHAPTER 6

COMMIT

In this chapter, we merge all the pieces required to mine *communication motifs* in scalable manner. We next discuss the mining pipeline of COMMIT.

Algorithm 3 COMMIT ($G, \tau, \Delta T$)

Require: A dynamic interaction network G, support threshold τ, and temporal threshold ΔT.

Ensure: *Communication motifs* with support no less than τ.

1: $\mathbb{C} \leftarrow$ All temporally connected components in G.
2: $SeqDB \leftarrow \{\mathcal{M}(c)|\forall c \in \mathbb{C}\}$
3: $\mathbb{S} \leftarrow$ All edge labels in SeqDB.
4: $f \leftarrow Floor(\mathbb{S})$
5: $\mathbb{E} \leftarrow$ **ExtensionMiner(** $f, \mathbb{S}, 0, \tau$**)**
6: **for all** edge label $e \in \mathbb{E}$ **do**
7: $P_j \leftarrow e$; $SS_j \leftarrow \{(i, < l >)$ for some $i,\ S_i[l] \sqsubseteq e\}$
8: $\mathbb{P} \leftarrow P_j, \mathbb{SS} \leftarrow SS_j$
9: **for all** subsequence $P_j \in \mathbb{P}$ **do**
10: $FreqP, FreqSS \leftarrow$ **SeqGrow(** $SeqDB, P_j, SS_j, \Delta T, \tau$)
11: **for all** $I \in FreqSS$ **do**
12: $\mathbb{A}^+ \leftarrow$**MotifMine(** $SeqDB, SS$)
13: $\mathbb{A} \leftarrow$Remove false positives from $\mathbb{A}^+$.
14: **return** $\mathbb{A}$

Alg. 3 presents the pseudocode. Given a dynamic network G, support threshold τ

and temporal threshold ΔT, first, all temporally connected components in the network are identified (line 1 in Alg. 3). These connected components are then mapped to the sequence space for frequent subsequence mining (line 2). In sequence space, the mining starts by collecting all edge labels in $\mathbb{S}$ (line 3). Note that $\mathbb{S}$ is a collection and not a set since an edge label e is present $sup(e)$ times in $\mathbb{S}$. EXTENSIONMINER is next executed on $\mathbb{S}$, which returns all closed edge labels $\mathbb{E}$ with supports no less than τ (line 5). For each edge label in $\mathbb{E}$, we calculate its largest support set using right-shit order. Next, we extend each edge $P \in \mathbb{E}$ with the help of SEQGROW algorithm to larger subsequences.

Algorithm 4 SeqGrow $(SeqDB, P, SS, \Delta T, \tau)$

Require: A sequence database $SeqDB = \{S_1, S_2, ..., S_N\}$, P is subsequence, SS the support set of P.

Ensure: $FreqP$ is a set of frequent sequences and $FreqSS$ contains the associated support sets.

1: **if** $|SS| < \tau$ **then**
2: **return**
3: $FreqP \leftarrow P$; $FreqSS \leftarrow SS$
4: **for all** instance $(i, < l_1, l_2, ..., l_j >) \in SS$ **do**
5: $\mathbb{S} \leftarrow \{S_i[l_k] \mid l_k > l_j \text{ and } \mid t_l - t_j \mid \leq \Delta T\}$
6: $f \leftarrow Floor(\mathbb{S})$
7: $\mathbb{E} \leftarrow$ **ExtensionMiner(** $f, \mathbb{S}, 0, \tau$**)**
8: **for all** edge $e \in \mathbb{E}$ **do**
9: $P_i^+ \leftarrow P \circ e$
10: $SS_i^+ \leftarrow$ **GetSup**$(SeqDB, P, SS, e)$ \\See section 6.2 for **GetSup()**
11: $\mathbb{P} \leftarrow P_i^+, \mathbb{SS} \leftarrow SS_i^+$
12: **for all** $P_i^+, SS_i^+ \in \mathbb{P}, \mathbb{SS}$ **do**
13: **SeqGrow(** $SeqDB, P_i^+, SS_i^+, \Delta T, \tau$)

Alg. 4 presents the SEQGROW algorithm. SEQGROW employs the sequence growth approach and aggressively leverages the apriori property. More specifically, given a subsequence P, SEQGROW extends P to a larger subsequence only if $sup(P) \geq \tau$ (line 1) (recall Theorem 3). To extend a sequence P with edge e, first all edges within ΔT from P are identified, and then filtered using EXTENSIONMINER. The extensions provided by EXTENSIONMINER are used to grow P till we reach a state where no extension e exists such that $sup(P \circ e) \geq \tau$. When SEQGROW terminates, it returns the frequent subsequences.

The last step in COMMIT is to map the frequent subsequences to the graph space. This is achieved using the MOTIFMINE algorithm. In COMMIT, an instance $I = (i, < l_1, \cdots, l_n >)$ of a sequence stores sequence id i, which corresponds to the i^{th} component of the network. In addition, each l_j in I maps the location of the j^{th} edge in I to its location in component i. As a result, each instance of subsequence uniquely identifies the subgraph it represents (lines 2-4 in Alg. 5). After identifying the corresponding subgraphs of all instances of a frequent subsequence, we compute their supports in the graph space using *temporal subgraph isomorphism* and check if they are actually frequent (lines 11-13 in Alg. 3).

6.1 MotifMine Algorithm

In this section, we design the MotifMine algorithm in Alg. 5 to return to graphs from the space of sequences.

An instance $I = (i, < l_1, \cdots, l_n >)$ of a sequence stores sequence id i, which corresponds to the i^{th} component of the network. In addition, each l_j in I maps the location of the j^{th} edge in I to its location in component i. Recall that in Sec. 4.1.1 we devised a mechanism to impose a total ordering on the edges of a given graph. Here, l_j

maps to the edge ranked l_j in component i. As a result, each instance of subsequence uniquely identifies the subgraph it represents (lines 2-4 in Alg. 5).

Example 5. *Consider one instance* $(3, < 2, 3, 4 >)$ *of pattern* $P_3 = (1,3)(1,3)(1,3)$ *from Fig. 5.6. The induced graph and its corresponding matched edges are shown in Fig. 6.1.*

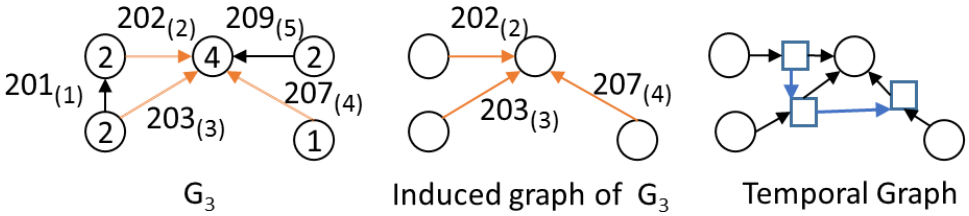


Figure 6.1: Instance $I = (3, < 2, 3, 4 >)$ of subsequence (1,3)(1,3)(1,3) corresponds to temporal component G_3 in Fig. 5.6. I represents an induced subgraph of G_3 (shown using the orange edges). For checking temporal isomorphism, induced graphs are converted into temporal graphs and the frequencies of temporal graphs are computed for final verification.

Algorithm 5 MotifMine $(SeqDB, SS)$

Require: A sequence database $SeqDB = \{S_1, S_2, ..., S_N\}$, support set SS.
Ensure: Motif with their frequencies

1: **for all** instance $(i, < l_1, l_2, ..., l_j >) \in SS$ **do**
2: Find G_i associated with S_i.
3: Find edge ids $< e_a, e_b, ..., e_j >$ associated with $< l_1, l_2, ..., l_j >$.
4: Form induced graph IG from graph G_i and edges $< e_a, e_b, ..., e_j >$.
5: **if** IG is *temporally connected graph* **then**
6: Create a temporal node for each edge.
7: Create link from temporal node A to B, if $time_A < time_B$ and $\nexists C$ such that $time_A < time_C < time_B$.
8: Count frequency of each temporal graph

In a traditional setting, to compute the frequency of each unique subgraph, we need to perform graph isomorphisms. In our problem, however, we need to check for temporal isomorphism (Definition 3.2). Towards that goal, given an interaction graph, we

convert it into a "temporal" graph such that the interaction graphs are temporally isomorphic to each other if an only if their corresponding temporal graphs are also isomorphic. The temporal graph is constructed in the following manner. On each edge $e = (s, d)$ of the interaction graph, we partition it into two edges $(s, t), (t, d)$ by introducing a new temporal node t. We refer to this temporal node as $e's$ *temporal node*. Let e' be the edge in interaction graph that is ordered immediately after e and t' the temporal node in e'. To impose the temporal constraints, we now add one more edge from t to t'. This process is repeated for each edge in the original interaction graph (lines 4-7). Fig. 6.1 illustrates the correspondence between an interaction graph and its temporal graph.

Theorem 5. *Let G_1 and G_2 be two interaction graphs and C_1, C_2 their corresponding temporal graphs respectively. If G_1 is temporally isomorphic to G_2, then C_1 is isomorphic to C_2.*

PROOF: Let us assume that the edges between the temporal nodes in C_1, C_2 are absent. In this scenario, it is trivial to see that if G_1 and G_2 are isomorphic, then C_1 and C_2 are isomorphic as well. Now, because G_1 and G_2 are temporally isomorphic, for any two edges $e_i, e_j \in G_1$ such that e_j is ordered immediately after e_i, for edges $f(e_i), f(e_j) \in G_2$, $f(e_j)$ is also ordered immediately after $f(e_i)$, where f is the bijection from edges in G_1 to G_2. Now, if we consider the edges between the temporal nodes in C_1, C_2, due to the edge ordering preservation, whenever there is an edge from the temporal node in e_i to e_j, there is also an edge from the temporal node in $f(e_i)$ to $f(e_j)$. Thus, a bijection exists from edges in C_1 to edges in C_2. □

Based on Theorem 5, the final temporal graph frequencies are computed and returned.

6.2 Pseudocode of the GetSup Algorithm

Alg. 6 presents the pseudocode of the GETSUP algorithm. We implement this following the algorithm proposed in [Ding *et al.*, 2009].

Algorithm 6 GetSup $(SeqDB, P, SS, e)$

Require: A sequence database $SeqDB = \{S_1, S_2, ..., S_N\}$, subsequence P, support set SS and edge e.

Ensure: A support set SS^+ of subsequence $P \circ e$,

1: **for all** $S_i \in SeqDB$ s.t. $SS_i = I \cap S_i(P) \neq \phi$ (P has instances in S_i, I is instance) in the ascending order of i **do**
2: $last\_pos \leftarrow 0$, $SS_i^+ \leftarrow \phi$.
3: **for all** $(i, < l_1, l_2, ..., l_{j-1} >) \in SS_i = I \cap S_i(P)$ in right-shift order **do**
4: $pos \leftarrow max\{last\_pos, l_{j-1}\}$
5: $l_j \leftarrow min\{l | S_i[l] \sqsubseteq e$ and $l > pos\}$
6: **if** $l_j = \infty$ **then**
7: **break**
8: $last\_pos \leftarrow l_j$
9: $SS_i^+ \leftarrow SS_i^+ \cup \{(i, < l_1, l_2, ..., l_{j-1}, l_j >)\}$
10: **return** $SS^+ = \cup_{1 \leq i \leq N} SS_i^+$

6.3 Computational Complexity of COMMIT

The worst case complexity of COMMIT is exponential which is similar to that of all frequent subgraph mining techniques, such as gSpan, Gaston, GRAMI. When the support threshold is close to 0, the answer set is exponential. However, the existing frequent subgraph mining techniques utilize the sparsity of the search space and in real world networks, the running times are less than exponential. COMMIT falls in this category. Theoretically, the worst case complexity of COMMIT is exponential, but in real world networks, COMMIT mines communication motifs in a more tractable manner.

CHAPTER 7

Experiments

In this chapter, we show that COMMIT is close to optimal in terms of accuracy, up to two orders of magnitude faster than existing techniques, and effective in characterizing the recurring interaction patterns.

- **Quality:** The accuracy of the communication motifs mined by COMMIT is close to optimal.
- **Scalability:** COMMIT is up to 2 orders of magnitude faster than existing techniques.
- **Impact:** Communication motifs are effective in characterizing the evolution of interactions.

7.1 Experimental setup

All experiments are performed on a 64-bit Intel i7-2600 CPU @ 3.40GHz machine with 32 GB RAM running on Ubuntu 14.04. All our algorithms are written and compiled in C++ with -O3 flag.

7.1.1 Datasets

We evaluate our COMMIT on the three social network datasets in Table 7.1. The Twitter dataset, which is the largest of the three, contains all tweets in December, 2009. If a tweet for person A "mentions" a set of persons P using the "@" operator, then it creates

an edge from A to each of the person in P. In Facebook, if user X posts a message on wall of another user Y, then a directed edge from node X is created to node Y. Finally, the Enron email network contains around half a million emails. Edges from an email are created in the same manner as in Twitter mentions.

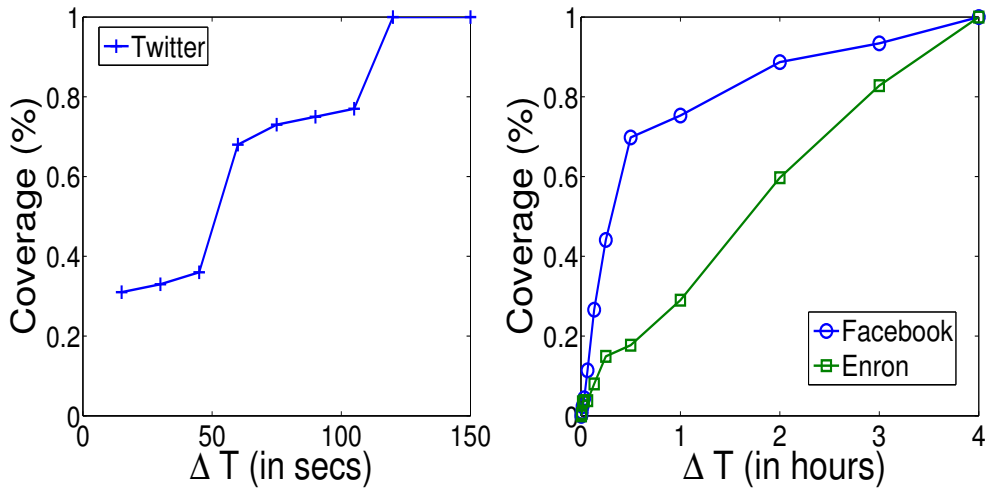
7.1.2 Benchmarking Setup:

The baseline approach is to enumerate all possible subgraphs in the given network, and verify if each of these subgraphs are frequent and temporally connected. We call this approach the *Naïve* approach. An alternative approach is to directly mine the frequent subgraphs and then verify if they are communication motifs. The state-of-the-art technique to mine frequent subgraphs is *GRAMI*[Elseidy *et al.*, 2014]. Thus, these two form constitute our baseline techniques.

In our experiments, we evaluate both top-k and range queries for scalability. The range version is compared with GRAMI, and the top-k version is benchmarked against Naïve since GRAMI does not support top-k frequent subgraphs. To evaluate accuracy of COMMIT, we use the top-k version since the intuitive interpretation of top-k is simpler. For top-k queries, we use the best-first search algorithm, where the support threshold changes as more patterns are mined. Specifically, at any stage, the support threshold τ is the support of the k^{th} most frequent subsequence till that point. All other aspects of the COMMIT algorithm in Alg. 3 remain the same. To ensure that the top-k set is not

Dataset	**Number of Nodes**	**Number of Edges**	**Duration (days)**
Twitter[sna,]	4,978,421	26,526,180	30
Facebook[Viswanath *et al.*, 2009]	45,813	855,539	1540
Enron Email Network[enr,]	10833	77050	349

Table 7.1: Summary of the datasets.



(a) Twitter (b) Facebook and Enron

Figure 7.1: Growth rate of coverage with ΔT in (a) Twitter and (b) Facebook and Enron.

overloaded with small motifs, we consider motifs of size at least 3.

ΔT is an important parameter in our model and controls whether two interactions are related. To learn the appropriate ΔT, we pick a sample of 1000 nodes proportional to their frequencies of interactions. This is necessary since a large portion of the users are dormant. For each selected node, we extract the subgraph of radius ΔT around it. Figs. 7.1(a)-7.1(b) present the average growth rate in the subgraph sizes as ΔT is varied in a range $[t_{min}, t_{max}]$, where

$$Coverage(\Delta T) = \frac{\text{subgraph size at } \Delta T}{\text{subgraph size at } t_{max}} \tag{7.1}$$

In Facebook, the growth rate saturates at 30 minutes indicating the lifeline of related interactions. In Enron, no clear pattern is visible as the growth rate is linear. In Twitter, the coverage shows two jumps at $\Delta T = 60$ seconds and $\Delta T = 120$ seconds. Thus, a threshold between 60 to 120 secs is a reasonable value for ΔT. In Twitter, we look at a much smaller time range, since the spread of information flow is extremely high,

but is limited within a short time-window. This behavior stems from a combination of two of its properties. First, twitter has a large number of celebrities with followers in millions. When such celebrities tweet, they generate a high volume of responses from the followers. At the same time, this bursty behavior exists for a small duration since a tweet is visible on the timeline only for a short period till it gets pushed down by more recent tweets. Due to this property of twitter, we vary the time window in the range of 15 seconds to 2 minutes. On the other hand, interactions in Facebook and Emails remain active for a much longer while since a Facebook wall or email inbox do not receive content at the same express rate.

This analysis guides our choice of default parameter values. Unless explicitly specified, we set $k = 500$, and $\Delta T = 30$ minutes for Enron and Facebook and 30 seconds for Twitter. A detailed analysis on the impact of ΔT on temporally connected components in interaction networks is provided in below section.

7.1.3 Impact of temporally connected components

In this section, we discuss how the properties of the temporally connected components affect the running time. Figs. 7.2(a)-7.2(c) show how the number of temporally connected components vary with ΔT. As can be seen, majority of the components contain less than three interactions, while the remaining interactions in the network are distributed among a minority of large connected components. The size distribution of temporally connected components in Twitter, which follows a power-law, is shown in Fig. 7.2(d).

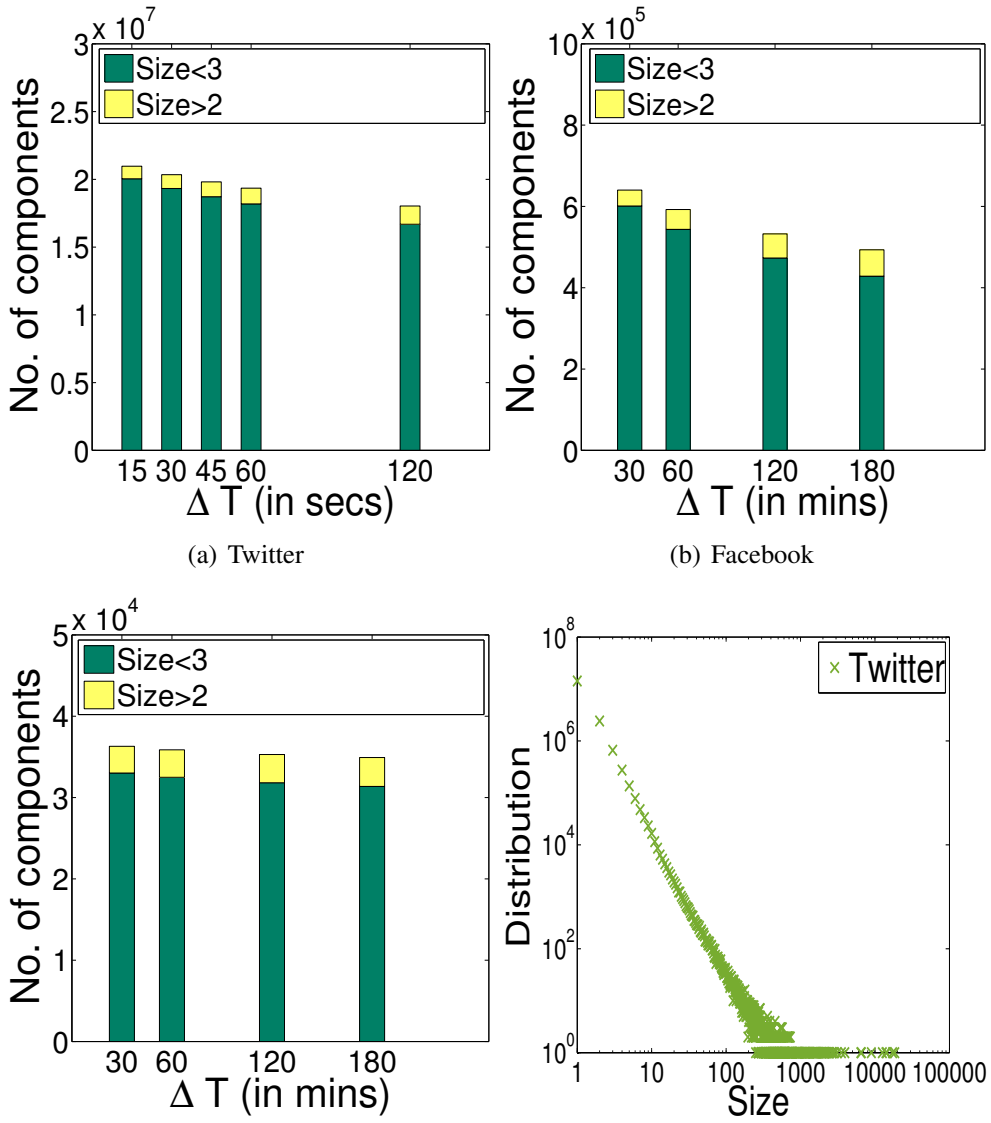
The running time is affected by two aspects: the number of temporally connected components, and the sizes of the temporally connected components. For example, a network with 20 million edges can split into 1 million components with 20 edges each,

or, in the extreme case, a single component containing 19 million edges and remaining components containing an edge each. Although both cases have equal number of components, the running times would vary significantly. As shown in Figs. 7.2(a)-7.2(c), the real results tend to be more like the second case. This phenomenon stems from the well documented scale-free property of social networks.

To illustrate the impact on running time, when the temporally connected components are large in size, there is more scope of overlap among motifs and thus higher supports for motifs of larger sizes. This drives up the running time since enumeration of larger motifs is more expensive. On the other hand, when there are more temporally connected components, the number of sequences is higher and consequently, small motifs become extremely frequent. In summary, both these factors are important. Both the size and the number of connected components are dictated by ΔT. While the size is directly proportional to ΔT, the number of components is inversely proportional to ΔT. Generally, with higher ΔT, the running time goes up (Figs. 7.6(a)-7.6(c)), which indicates that larger components have more impact on the running time.

7.2 Accuracy of COMMIT

In this section, we measure the accuracy of communication motifs mined by COMMIT. To construct the ground truth dataset, we identify the top-k communication motifs using the Naïve algorithm, which enumerates all possible subgraphs. Since, the sizes of the datasets are too large for the naive approach, it invariably runs out of main memory and crashes even on a machine with 32GB of main memory. This happens since Naïve needs to enumerate all possible subgraphs and store them in memory for support counting. Due to this weakness of Naïve, we build the ground truth dataset in a breadth-first manner. More specifically, we first generate all communication motifs of size 2 edges.



(a) Twitter (b) Facebook

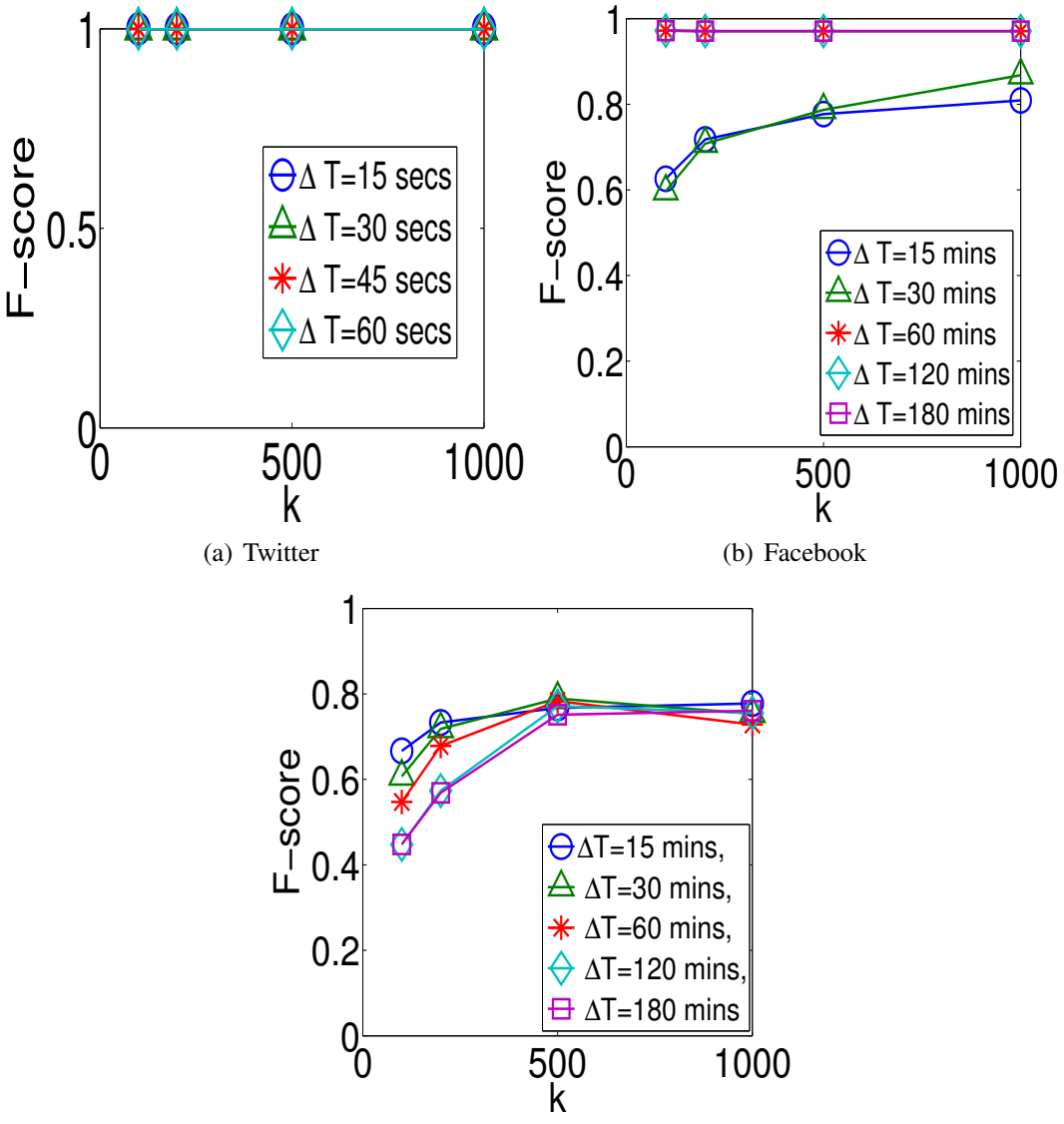
(c) Enron (d) Twitter

Figure 7.2: (a-c) Number of temporally connected components in the three interaction networks. (d) The distribution of the sizes of temporally connected components in Twitter at $\Delta T = 120$ seconds.

Then, we proceed to motifs of size 3 edges and so on. Thus, if Naïve crashes while mining motifs of size m, then we know that we have the ground truth for all motifs till size $m-1$. The size of a communication motif is the number of interactions in it. To benchmark the accuracy of COMMIT, we compute the top-k list on only those communication motifs that are within the size of $m-1$. The accuracy of COMMIT is quantified using the *F-score* measure. F-score can be visualized as a weighted average of the *precision* and *recall*. An F-score of 1 corresponds to the best performance, and 0 corresponds to the worst.

Figs. 7.3(a)-7.3(c) demonstrate the results on a range of ΔTs as k is varied. On Twitter, which is the largest network with more than 26 million edges, the naive algorithm quickly runs out of memory. Naïve is able to generate motifs only up to size 3, all of which also occur in the top-k lists of COMMIT. Thus, Twitter is not a good dataset for verification of accuracy. On Facebook and Enron, Naïve scales better and its top-k lists contain motifs of larger sizes. On Facebook, COMMIT generally achieves an F-score exceeding 0.8. As k grows, the F-scores almost touch 1. A similar improvement with k is also seen in Enron. This improvement of accuracy with k is natural. At small ks, the difference in the supports of the top motif and the k^{th} motif is normally very small. Here, if COMMIT underestimates the frequency of a motif by a small amount δ, then its impact on the top-k list is high. When k grows, a wider error range is available for a communication motif to remain within the top-k list and hence, the increase in accuracy. To put the value of k in context, even Enron, the smallest network, contains millions of subgraphs. Thus, $k = 50$ represents a very small portion of the subgraph space, and even in this small region, COMMIT has an accuracy 0.6, which improves to 0.8 at $k > 500$.

Except on Facebook, the accuracy is invariant with ΔT, specifically at higher ks where the ranking stabilizes. On Facebook, the accuracies are slightly lower at $\Delta T = 15$

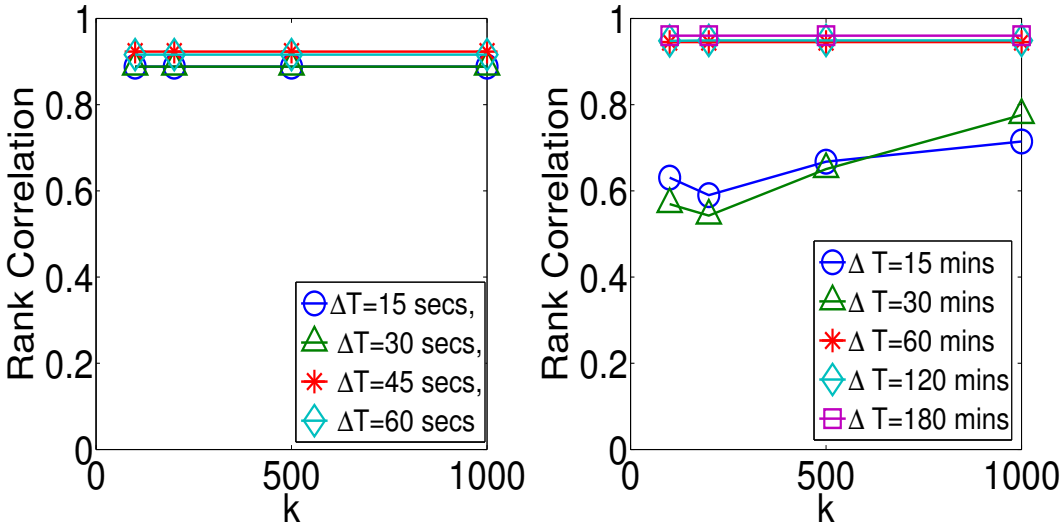


(a) Twitter

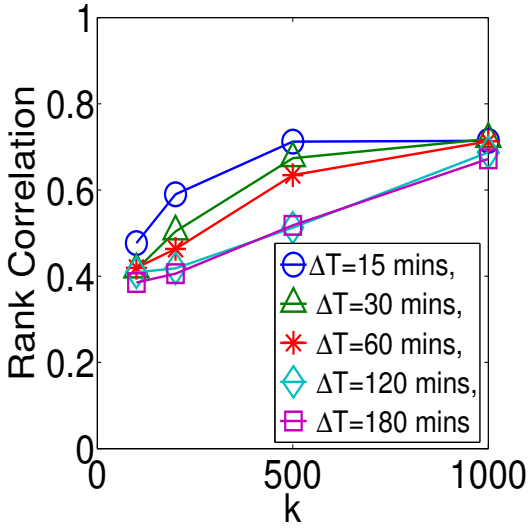
(b) Facebook

(c) Enron

Figure 7.3: Analysis of F-score with k on (a) Twitter, (b) Facebook and (c) Enron.



(a) Twitter (b) Facebook



(c) Enron

Figure 7.4: k vs Spearman's rank correlation on (a) Twitter (b) Facebook and, (c) Enron.

minutes and $\Delta T = 30$ minutes because at higher ΔTs, Naïve once again fails to scale and mines a limited number of motifs, all of which are part of COMMIT's top-k answer set. Since ENRON is a much smaller network, Naïve finishes within a manageable time limit at all ΔTs.

In addition to the F-Score of the top-k list, we also verify how well the ranking within the top-k lists are preserved. To assess the similarity of the top-k rankings, we compute the Spearman's rank correlation coefficient[Spearman, 1904] between the ground truth and COMMIT's top-k lists. Spearman's rank correlation takes as input a list of items and their ranks on each of the methods. In our case, the top-k lists from Naïve and COMMIT may not overlap completely. In such a situation, we create a list by taking the union of the two top-k lists and their corresponding ranks. Figs. 7.4(a), 7.4(b), 7.4(c) presents the results against k on multiple ΔTs. The trends are similar to that of F-score. In Twitter, the correlation is above 0.9. However, this is largely due to Naïve running out of memory and generating only a small set of motifs. On Facebook, the correlation improves from 0.6 to 0.8 as k grows at $\Delta T = 15$ minutes and $\Delta T = 30$ minutes. At higher ΔT, Naïve crashes before generating all k patterns. Similar to Facebook, Enron and the correlation saturates at 0.75 for k beyond 500. The reason behind this improvement is the same as with k; the permissible error range increases with k.

To summarize, both the F-score and rank correlation improve with k and saturate around 0.80 on Facebook and Enron. On Twitter, Naïve fails to scale.

7.3 Scalability of COMMIT

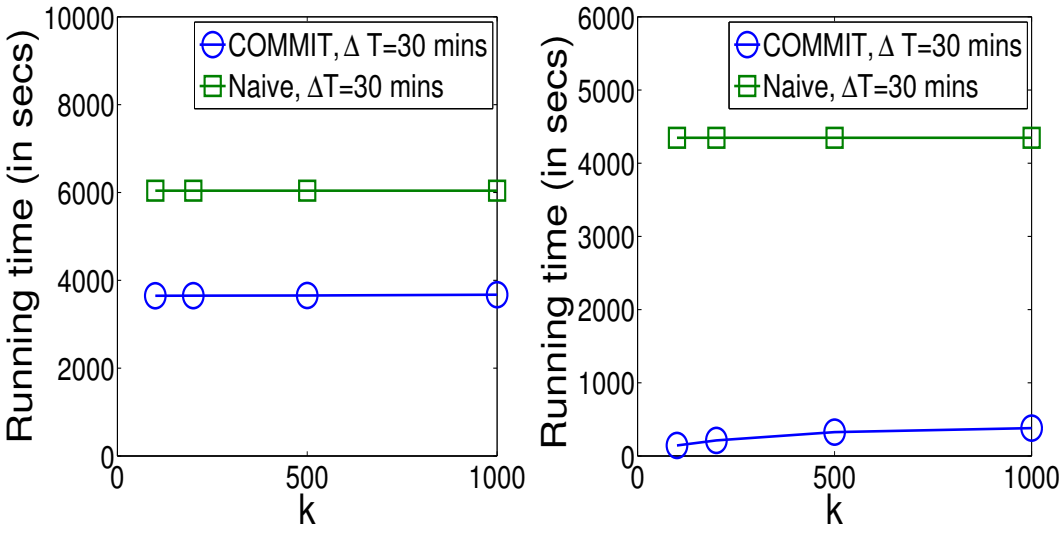
We evaluate COMMIT on both top-k and range queries. In the top-k setting, we benchmark COMMIT against Naïve since no other technique exists. In the range query set-

ting, where the input is a support threshold, we benchmark COMMIT against GRAMI. While GRAMI does not solve the problem of communication motif, it forms a part of the alternative pipeline where frequent subgraphs are first mined, and then analyzed to check if they satisfy the constraints of communication motifs. In other words, GRAMI provides a lower bound on the running times of the alternative communication motif discovery route. In the following experiments, unless specifically mentioned, we set $k = 500$, and $\Delta T = 30$ minutes for Facebook and Enron, and $\Delta T = 30$ seconds for Twitter.

7.3.1 Top-k queries

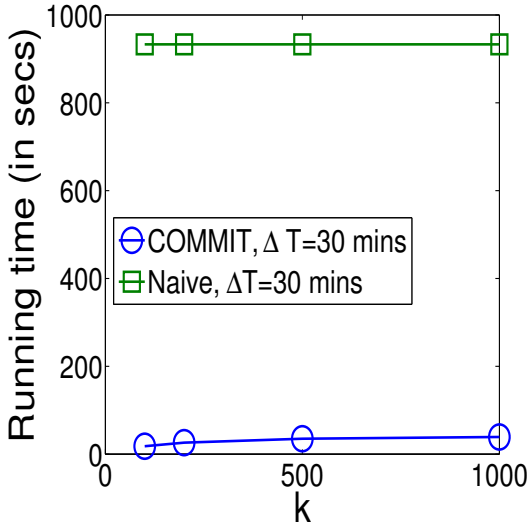
First, we benchmark the performance of COMMIT against k. Figs. 7.5(a), 7.5(b),7.5(c) present the results. As we saw in the previous section, Naïve inevitably runs out of memory on Twitter and Facebook, and thus it is not possible to compute its actual running time. Thus, in these experiments we report the time Naïve takes to mine all communications motifs of size 3 in Twitter and size 4 in Facebook. In other words, the experiments only provide loose lower bounds on the actual running times of Naïve. Since Naïve needs to enumerate all subgraphs regardless of the value of k, its running time is constant with k. In COMMIT, there is a minor increase in the running time with k. For top-k queries, we use the best-first search algorithm, where the support threshold changes as more patterns are mined. At any stage, the support threshold is the support of the k^{th} most frequent pattern till that point. When k is large, this threshold is smaller and hence an increase in the running time. Notice that the running times of Naïve on Twitter and Facebook are similar although Twitter is significant larger. This results from that fact that regardless of the dataset size, Naïve runs out of memory around the same time.

We further study the scalability of top-k queries against ΔT, which controls when



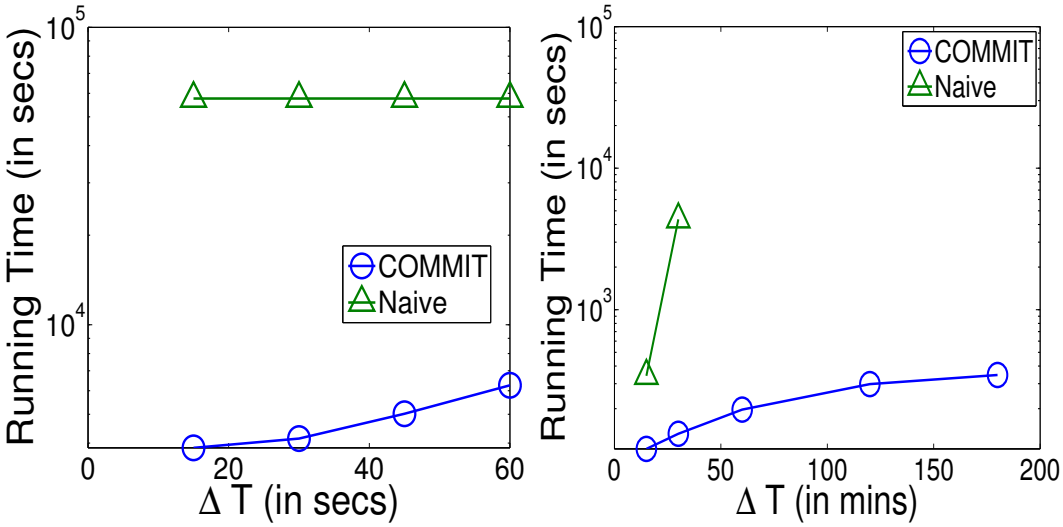
(a) Twitter

(b) Facebook



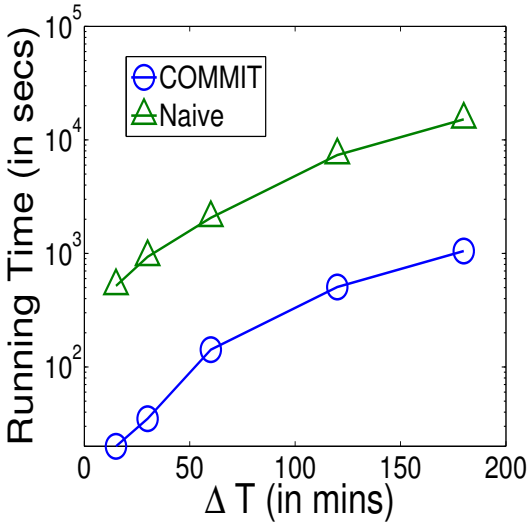
(c) Enron

Figure 7.5: Growth rate of running time with k in (a) Twitter and (b) Facebook and (c) Enron.



(a) Twitter

(b) Facebook



(c) Enron

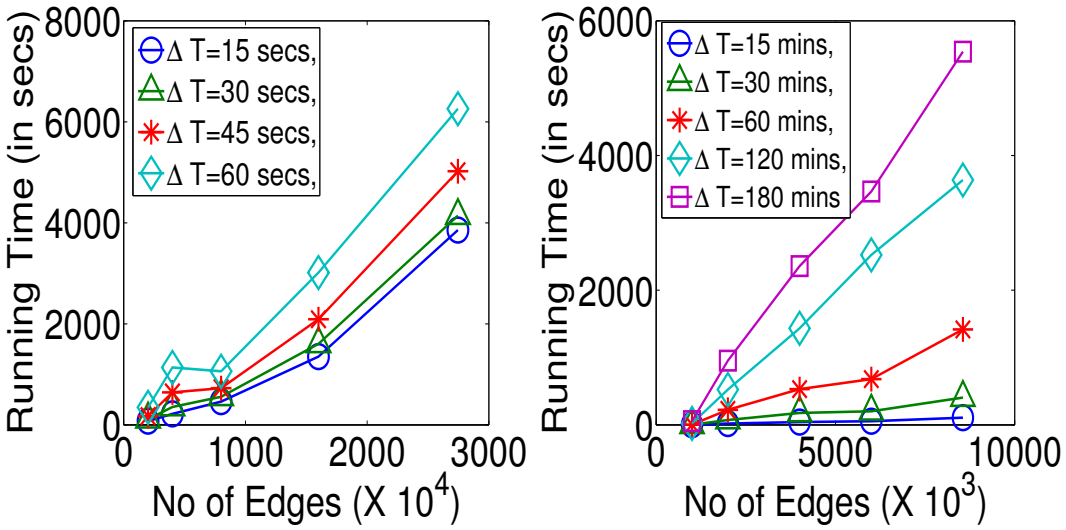
Figure 7.6: Growth rate of running time with ΔT in (a) Twitter, (b) Facebook and (c) Enron.

two interactions are classified as related. In addition, we adopt a different strategy to estimate the running time of Naïve on Twitter since Naïve is unable to scale beyond patterns of size 3. To mine patterns of larger sizes on Twitter, we partition Twitter into multiple smaller chunks of 50,000 edges each. Then we let Naïve run on each of these partition in parallel. While Naïve finished on some of the partitions, it could not finish mining all chunks even after 20 hours across all values of ΔT. Thus, as visible in Fig. 7.6(a), the running time is a straight line. Fig. 7.6(b) demonstrates the performance in Facebook. As can be seen, there is an exponential growth in the running time of Naïve. At a larger ΔT, the sizes of the communication motifs and their corresponding subsequence representations are larger. Thus the sequence growth algorithm runs longer, the support counting is more expensive, and in the graph space, verification cost is higher. In addition, the sizes of the temporally connected components grow with ΔT as well. The impact on Naïve is much more drastic since the cost of subgraph isomorphism goes up. On the other hand, COMMIT is insulated from such a drastic impact since the bulk of the processing happens in sequence space. A similar trend to Facebook is also visible in Enron. Overall, COMMIT is up to two orders of magnitude faster than the Naïve algorithm.

Finally, we look at the growth rate of running time against the size of the interaction network. Figs 7.7(a)-7.7(b) presents the results on a series of ΔTs. On both datasets, the growth rates resemble a linear curve. On twitter, the growth rate is higher since it is more dense. We ignore the ENRON dataset for this experiment since it is the smallest.

7.3.2 Range query

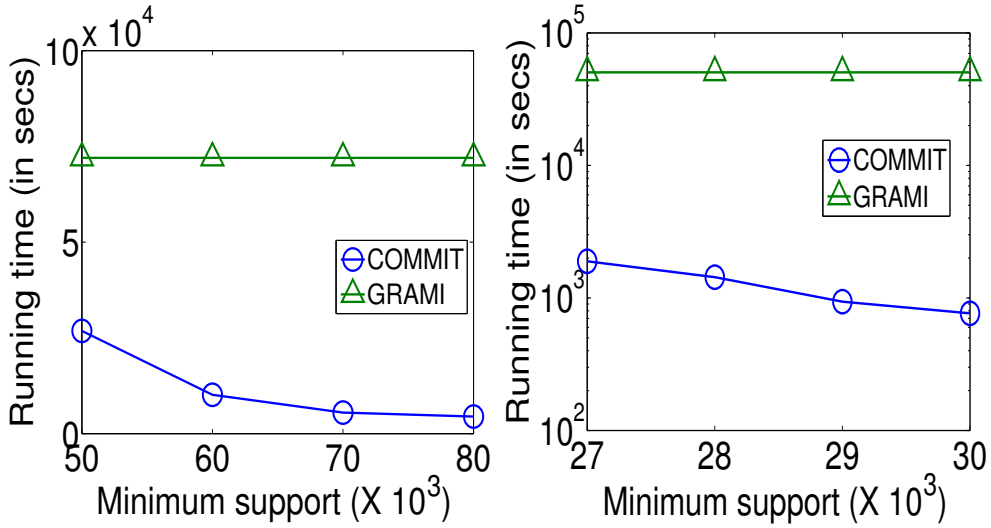
We compare the running time of COMMIT with GRAMI[Elseidy *et al.*, 2014]. Note that the answer sets of GRAMI and COMMIT are different. GRAMI mines frequent subgraphs. However, as illustrated earlier, these frequent subgraphs can subsequently



(a) Twitter (b) Facebook

Figure 7.7: Growth rate of the running time against the size of the interaction network in (a) Twitter and (b) Facebook.

be analyzed to extract the communication motifs. As discussed earlier, without any metadata, it is non-intuitive to know what an appropriate support threshold is since the number of subgraphs in the networks itself is unknown. We therefore follow the strategy of GRAMI[Elseidy *et al.*, 2014], where the threshold is set in proportion to the number of nodes. In Twitter, we vary the support threshold from $\tau = 1\%$ of total number of nodes to higher values. In this support range, GRAMI fails to complete even after 16 hours. Thus, the running time of GRAMI is shown as a straight line in Fig. 7.8(a) and only indicates a lower bound of the actual. GRAMI fails to scale since it relies heavily on node labels to prune the search space. COMMIT, on the other hand, uses node degrees as labels, which are subsequently used to mine communication motifs. As expected, the running time goes down with increase in the minimum support threshold. To ease the setting and check the performance at higher values of τ, in Facebook, we start growing τ from 5% of the node set size. However, we again see a similar result and GRAMI fails to complete within 16 hours. Fig. 7.8(b) demonstrates the results.



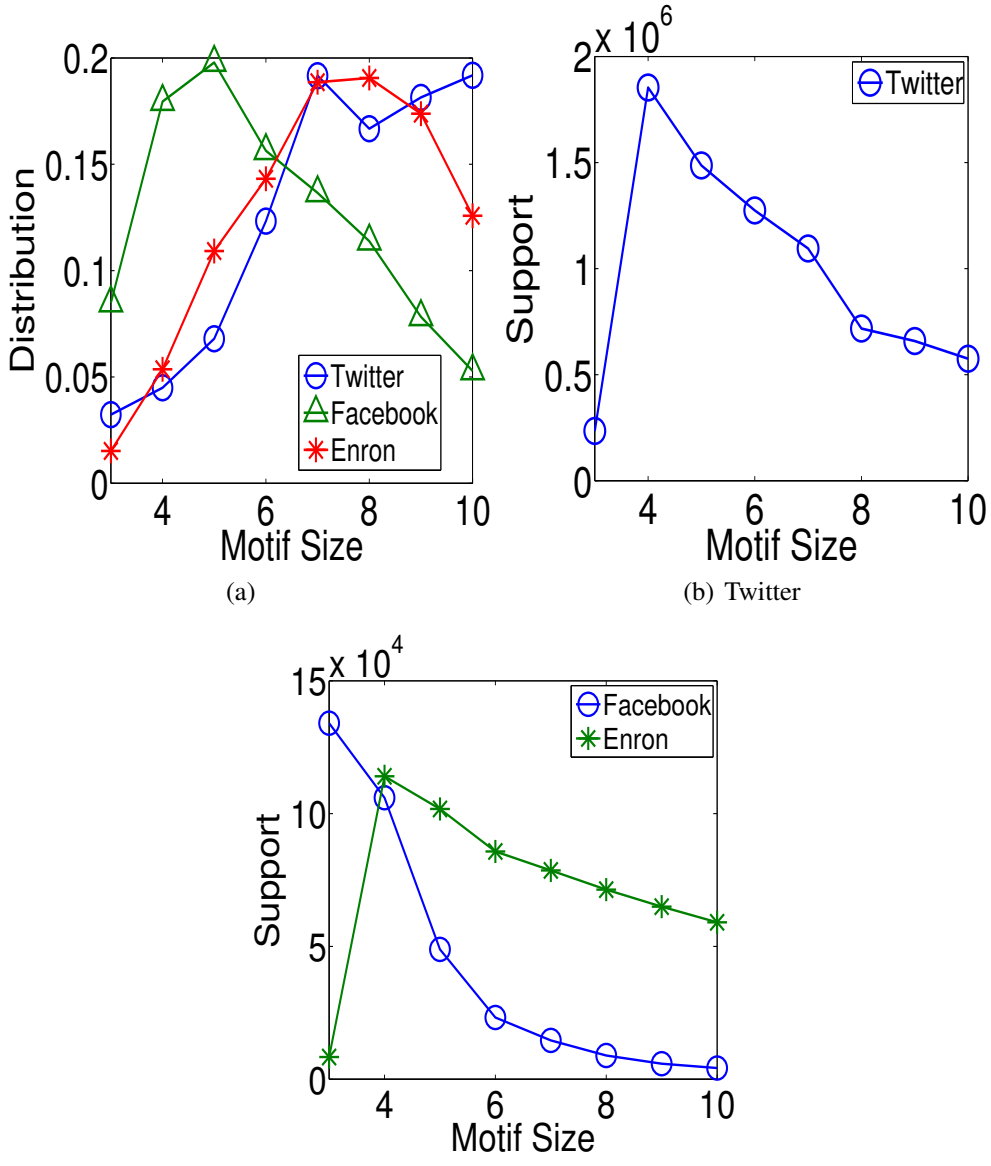
(a) Twitter (b) Facebook

Figure 7.8: Growth rate of running time against the support threshold in the range query setting on (a) Twitter and (b) Facebook.

In contrast, COMMIT finishes within 30 minutes across all values of τ in Fig. 7.8(b). Overall, COMMIT is more than 70 times faster than GRAMI.

7.3.3 Distribution of motif sizes

Secs. 7.3.1 and 7.3.2 show that Naïve can somewhat scale when the motif sizes are small; specifically, motifs of size 3 in Twitter and size 4 in Facebook and Enron. In this section, we investigate whether motifs of larger sizes occur in interaction networks. Fig. 7.9(a) demonstrates the distribution of motifs with respect to their sizes in the top-5000 set. Across all three networks, majority of the motif sizes are above 4. This result highlights the need for COMMIT. Next, we further study the size of communication motifs with respect to their support levels. More specifically, we plot the summation of supports of all motifs of a particular size. Fig. 7.9(b) shows the result in Twitter, which is the largest interaction network among the three. In Twitter, the total support



(a)

(b) Twitter

(c) Twitter

Figure 7.9: Distribution of motif sizes (a) and their supports on (b) Twitter and (c) Facebook and Enron datasets.

from size-4 motifs is the highest. Motifs of sizes between 5 to 7 are also very frequent. An important observation that comes out from the results in Figs. 7.9(a) and 7.9(b) is that although the number of size-10 motifs is much higher than size-4 motifs, size-4 motifs are more frequent. This is natural since it is possible to merge two or more size-4 motifs into a single larger motif. Due to this same reason, the top-3 most frequent motifs across all three datasets, shown in Fig. 7.10, are of size 3. On the other hand, the number of larger motifs in the top-5000 list, such as those of size 10, is higher since combinatorially, the space of size-10 motifs is larger than size-4 motifs.

Fig. 7.9(c) demonstrates the distribution of supports of communication motifs with respect to their sizes in the Facebook and Enron datasets. Generally, the overall support decreases with motif size. However, Facebook shows a different behavior in one aspect. While size-3 motifs are rare in Twitter and Enron, they are extremely frequent in Facebook. As visible in Fig. 7.9(a), the number of size-3 motifs in the top-k set is also relatively higher in Facebook than in Twitter or Enron. This behavior indicates that people tend to interact in smaller groups in Facebook than in Twitter or emails in a corporate setting, such as Enron.

7.3.4 Approximation factor

The mined communication motifs are the approximation of the true communication motifs present in the network. But, as shown in section 1.2, no state-of-the-art frequent subgraph mining can be utilized to mine communication motifs because of scalability bottleneck. To solve this scalability bottleneck, we provide an approximation. As shown in section 7.2, the accuracy measured with metrics F -score and rank correlation is around 0.80 on the real world networks. The approximation factor will depend on the underlying network and will improve with the value of k.

7.4 Implications of communication motifs

In this section, we analyze the top-3 communication motifs of size 3 from Twitter, Facebook, and Enron and discuss how they reveal the patterns of communications in a social network. The motifs are shown in Fig. 7.10.

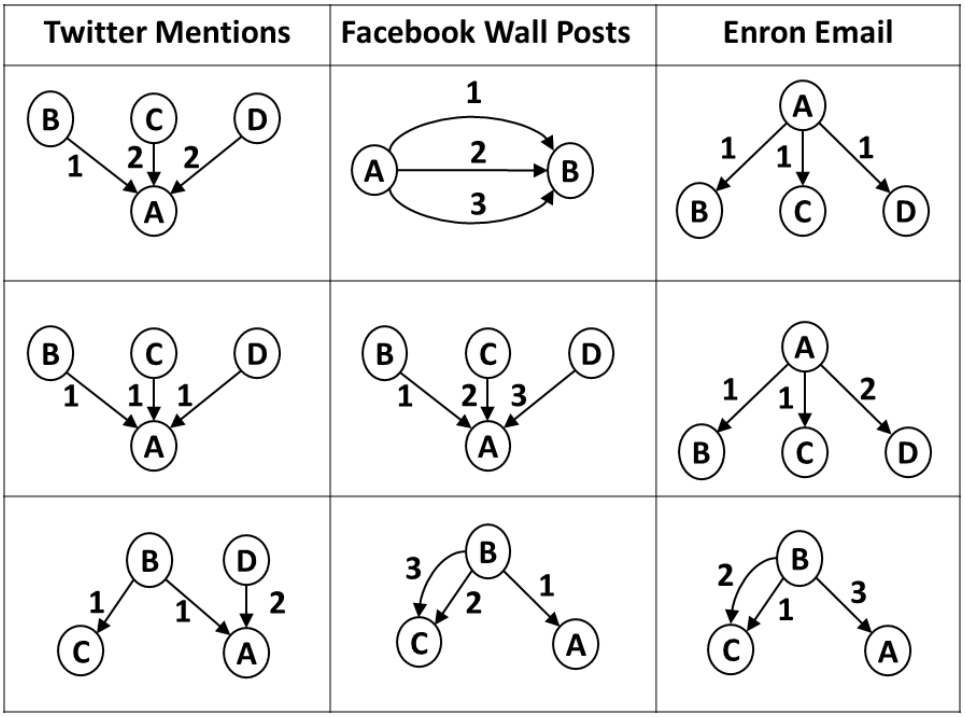


Figure 7.10: Top-3 communication motifs.

7.4.1 Twitter mentions dataset

A distinct pattern in Twitter that is revealed through communication motifs is that people tend to communicate more with celebrities or twitter handles of prominent events that are in news. For example, the news that "Tiger Woods announcing that he will not be attending his own charity golf tournament" lead to lot of tweets in which "@TigerWoods" is mentioned. An identical pattern is again observed during the "movie release

of Avatar" generating to bursts of tweets to "@officialavatar", which is the official account for Avatar movie.

The first communication motif in Twitter shows that node "A" is related to some celebrity and the edge labels denote the *temporal sequence* of communication links. We observe that often there is a sudden peak in the number of tweets to a specific person within a short duration of time. This pattern is evident in the overlapping times stamps of the first motif and even more prominently in the second communication motif in which all three people mention the celebrity (or prominent event representative) node A at the same time. The third communication motif shows people (node B) tend to mention both the famous person A and second person (node C) in the same tweet. Overall, we observe that people use Twitter as a medium to communicate with famous persons (or organizations like a soccer club, or upcoming movie, etc.). Furthermore, the tweets are often bursty in nature as evident from the first and second communication motifs. The burstiness is explained from the design of Twitter where a tweet is continuously pushed down from the timeline by more recent tweets and is therefore visible only for a limited period.

7.4.2 Facebook wall-posts dataset

In Facebook, the patterns are distinctly different from Twitter. We observe with the help of communication motifs that people tend to interact more with their friends. As evident from the first communication motif, people (B) tend to post a message on the wall of same friend (A) again and again. Another distinct pattern in Facebook shows that when a person (A) has a birthday or anniversary, A's friends wish him/her by writing on the wall of A . This pattern is the second most frequent behavior as evident in the second *communication motifs*. The third common behavior is people (B) interacting frequently with multiple friends (A and C), with a distinct preference towards one of them (C).

7.4.3 Enron

The fact that Enron is an email network is clearly evident from the top-3 patterns in Fig. 7.10. The communication motifs reveal emails being used as a broadcasting mechanism. This is expected since the Enron dataset contains data from about 150 users, most of whom are senior managers in the company hierarchy[enr,]. A manager routinely needs to distribute information to employees working under him/her. Hence, it is not surprising to see the top-2 motifs depicting this pattern. The third communication motif is of similar nature as well, but shows multiple emails to the same user (C).

7.5 Applications of communication motifs

As clearly evident from our analysis on the three interaction networks, communication motifs are effective in characterizing the common mode of interactions happening in a network. These motifs can be used for a myriad of applications such as predicting trends by mining the patterns that commonly precede the trend, and predicting the nature of the communication taking place such as birthday wishes, group discussions, etc. Furthermore, communication motifs reveal that the underlying social network has a strong influence on how people interact. Similar observations are made in a previous study by Kovanen et al. [Kovanen *et al.*, 2013], who show the difference in communication patterns in dense and sparse regions of electronic communication records. All in all, these motifs can be used as *features* to characterize social networks itself.

Indeed, COMMIT is a heuristic and optimality cannot be guaranteed. We resort to a heuristic since computing the optimal answer set is NP-hard. Therefore, an important question arises: *If communication motifs are used to characterize social networks, what is the impact of a non-optimal answer set?* The analysis in Sec. 7.2 shows that the F-

score and rank-correlation of COMMIT is generally around 0.8. Thus, the answer set is close to optimal. More importantly, the non-optimal motifs in COMMIT's answer sets are also highly frequent; only, they are not in the top-k list. Thus, these small minority of non-optimal motifs may not be the best k motifs to characterize, but they are still informative and unlikely to lead to any inaccurate conclusions.

CHAPTER 8

Temporal Analysis of Telecom Call Graphs

Analysis of dynamic networks can lead to new insights such as densification laws and shrinking diameters. We analyze temporal properties of Call Detail Records containing more than 1 billion calls based on sliding windows at various time windows such as day-night windows, weekday-weekend windows, etc.

8.1 Introduction

In most network analysis, the nodes and edges are considered static which implies graph topology will not change with time. In interaction networks, the graph topology changes with time since interaction between entities last for some time. Static analysis of such network might lead to erroneous inferences. Graph generation models assumed that average degree of nodes remains constant and diameter of graph increases as network grows. This assumption was proved wrong by Leskovec et al. [Leskovec *et al.*, 2005].

[Leskovec *et al.*, 2005] proposes densification of graph i.e. increase in average degree of nodes over time and shrinking diameter over time as graph grows based on study of two temporal properties density and diameter. This result have implications in graph sampling, prediction of next state of graph, graph generation models and also abnormality detection. Analysis of temporal properties like degree distribution, neighborhood distribution, cliques and strongly connected component over time of call and SMS graphs is done in [Nanavati *et al.*, 2008] which also proposes treasure hunt model for mobile call graphs. The time window specified is Uniform day time window

for two operators, while we have explored other time windows. Gautier Krings *et al.* [Krings *et al.*, 2012] analyzed the effect of time window on telecom networks with focus on link dynamics. Although they analyzed the effect of time windows, their focus is different from ours. The temporal properties which we analyzed are completely different with the focus on patterns of those temporal properties on different time windows.

With 1 billion edges in call graph, performing temporal analysis on short time period is computationally expensive. Hence we address this problem by proposing to study graph generated by aggregating data over different time windows. The goal of the study is to identify differences in calling patterns when windows range over different time periods. We chose a day-night split, i.e., calls made during a single day were aggregated together and calls made during a single night were aggregated together, a weekday-weekend split, i.e., calls made during a given week and the subsequent weekend were aggregated separately, a uniform time window, i.e., calls made during successive n days were aggregated together; and cumulative weeks, i.e., calls made till the end of a certain week starting from week 0 were accumulated.

8.2 Dataset

A Call Detail Record (CDR) of mobile telecom operator contains information related to calls like caller number, called number, time at which call is initiated, duration of call and many other details. CDR analysis is done by treating people as nodes and calls between them as edges.

Properties	Values
No of nodes	1771134
No of edges	20510811
No of weakly connected components (WCC)	18
Size of Maximum WCC	1766905
No of edges in Maximum WCC	20508042
Largest bi-connected component nodes	1465195
Largest bi-connected component edges	20199413
Clustering Coefficient	0.063308
Diameter	12
Reciprocity	0.4394683
Density	6.538532e-06
Transitivity	0.01247928

Table 8.1: Static properties of Call graph

8.2.1 DataSet Preparation

All calls are stored in structured text files and various details of a single call are recorded in this files. Caller No, Called No , Time of Call and Duration of all calls were extracted from more than 1 billion calls using Apache Pig Script. For a specific time window say uniform day time window, multiple calls between two persons in a day were considered as single call since considering multiple calls requires more computational power. We performed our analysis on two 2.4GHz Quad-Core Intel Xeon processor, 6144kb L2 cache and 24Gb running main memory. The analysis of properties was done using Stanford SNAP tool and igraph tool in R.

8.3 Static Properties

In static graph analysis, multiple calls from one person to other are considered as single directed edge. Weights can be added to the edges [Onnela *et al.*, 2007] depending upon the duration of call between two persons but the properties we analyzed does not differ

for edge weighted graphs.

The values of properties of static call graph are shown in Table 8.1. The number of nodes indicates the actual number of customers subscribed to our customer for a specific region. The number of edges indicates the total number of unique calls in span of 90 days. Note that actual number of calls is more than 1 billion but number of unique calls is near to 20 million. The ratio between number of unique calls to the total number of calls is 0.02 signifies the existence of large number of calls between same two persons.

The number of weakly connected components are 18 of which the number of nodes in maximum WCC is 99% of total number of nodes signifying the global connectivity between persons. The diameter of the graph is 12 whereas the diameter reported in [Nanavati *et al.*, 2008] is 20 but the location of both operators varies continent wise. The low value of density signifies the sparseness of the call graph.

8.4 Temporal Properties

The call detail records of our operator consists of 90 days of call records. The various temporal time windows on which we performed our analysis, we call them as

8.4.1 Day Night Time Window

In this time window, all calls made in day light from 6 am to 5:59 pm are aggregated to form a day time graph while calls made in night from 6:00 pm to 5:59 am are aggregated to form a night time graph. So for 90 days, a total of 180 alternate day and night graphs were created. This time window helps in analyzing call patterns and properties of call graph in day and night time and also helps in anomaly detection.

8.4.2 Uniform Day Time Window

In this time window, for each day a graph is created by aggregating all calls from that day. Some people may not even initiate a single call in a specific day, hence the number of nodes across graphs varies from day to day. Since 90 days of CDR data is available, total 90 such graphs were created. This time window analyzes call patterns and properties of graph on daily basis and may even help in anomaly detection.

8.4.3 Weekday and Weekend Time Window

In this time window, calls in weekday of a specific week are aggregated to form a graph and same procedure is followed for weekend of that specific week. For 90 days, a total of 25 weekday and weekend graphs were created. This time window helps in analyzing patterns and properties of calls across weekdays and weekends since calling patterns might be different because of holidays in weekends.

8.4.4 Cumulative Week Time Window

In this time window, all calls in each week are aggregated from day 0 till that week. For instance, time snapshot of graph for 3^{th} week would contain all calls made from week 0 to week 3. So for 90 days, total of 13 such cumulative week graphs were created. This time window represents the network growth phase and helps in inference about network growth.

The temporal properties analyzed for mentioned time windows are number of nodes, number of edges, number of bidirectional edges, number of closed triads, number of open triads, clustering coefficient, effective and full diameter. The number of nodes across a time window say uniform day time window signifies the number of people

initiating atleast a single call on that day. The number of edges across a time window signifies the total number of unique calls between people in that time window. The number of bidirectional edges signifies reciprocity in the call graph for a time window. The number of open triads signifies number of people who are at a distance of one hop while number of closed triads signifies effect of triadic closure. The full diameter is maximum distance between two nodes while effective diameter of graph is the 90^{th} percentile distance between two nodes. The clustering coefficient of graph is $3 *$ number of closed triads/ number of open triads.

8.5 Results and Discussion

The results of experiments on the four time window as mentioned in section 8.4 is discussed here . The call details records contains calls from 31st Jan 2010, 4 pm onwards to 30th April 2010 till 11:59 pm (90 days).

8.5.1 Uniform Day time window

The temporal properties results of uniform day time window is shown in Figure 8.1. For the first day, since CDR contains calls from 4 pm, the total number of calls on first day is very less and hence the properties of graph on first day varies significantly from the other day graphs. The diameter and effective diameter of first day graph is significantly high due to less number of calls but the number of people initiating the call are comparatively high resulting in increase of diameter. The clustering coefficient of first day is significantly low compared to other days. In Figure 8.2, all the five properties have a significant drop in values in first day. The number of edges for latter days seems to regular interval peaks. So, to verify if any day say Sunday dominates all other days in

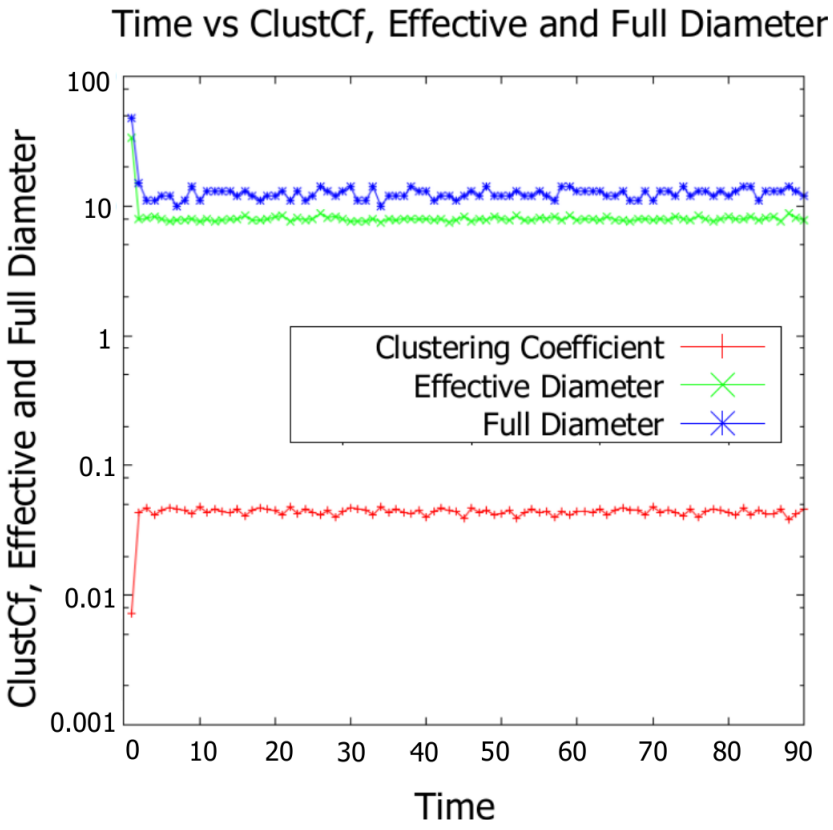


Figure 8.1: The temporal properties of call graph on uniform day time window. For each day, a call graph is created by aggregating all calls on that day and various properties of that call graph are analyzed.

terms of number of unique calls, we found the number of calls with respect to specific day. As shown in Figure 8.3 , we found that no particular day dominates other with respect to unique number of calls. A simple check can be done by verifying different color points at the top of each day. For fairness, we removed week one data since Sunday of week 1 was not completely recorded.

8.5.2 Day Night time window

The temporal properties results of day and night time window are shown in Figure 8.4. Since for first day, calls are recorded from 4 pm, we discard calls from 4 pm to 5:59 pm

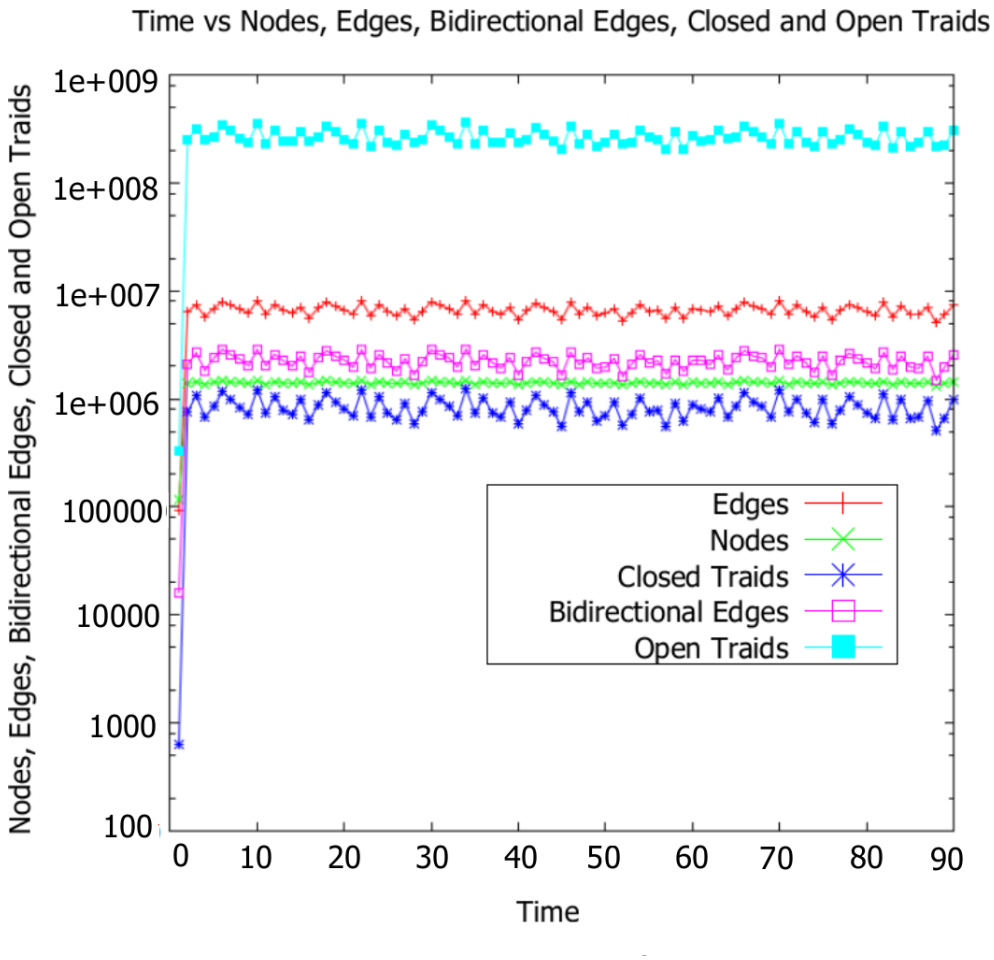


Figure 8.2: The temporal properties of Call graph on uniform day time window.

and hence the first data point is night point. The full diameter and effective diameter of this time window graphs increases in night graph compared to day time graph while clustering coefficient decreases.

8.5.3 Weekday and Weekend time window

The temporal properties results of weekday and weekend time window is shown in Figure 8.6 . Since first day was Sunday and calls from 4 pm were recorded, we removed first weekend datapoint from Figure 8.6 for good visualization. The full diameter of

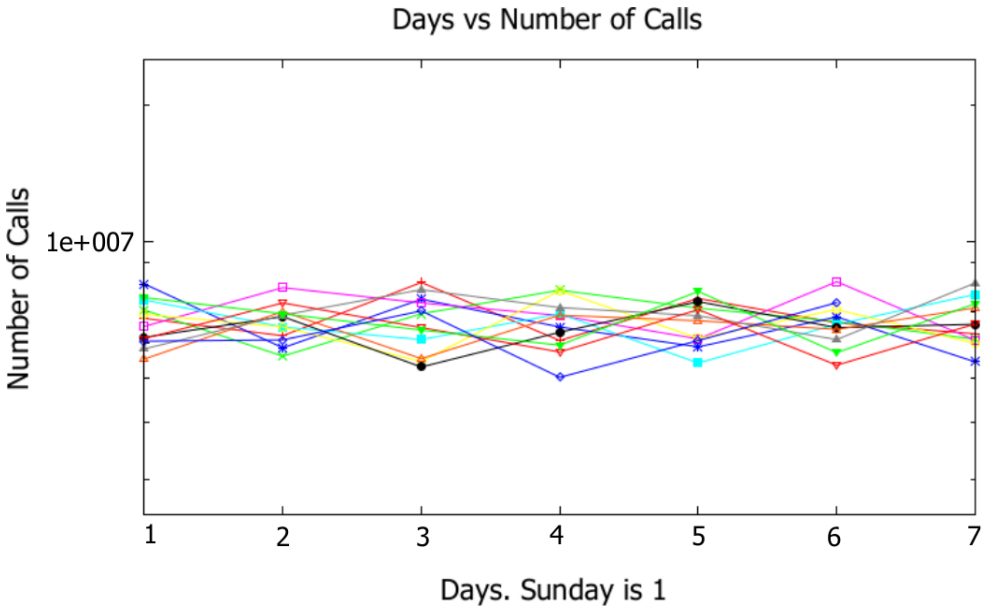


Figure 8.3: The number of unique calls with respect to days.In particular no day dominates other days in terms of unique calls, as can be seen by different top color for each days. Day 1 represents Sunday.

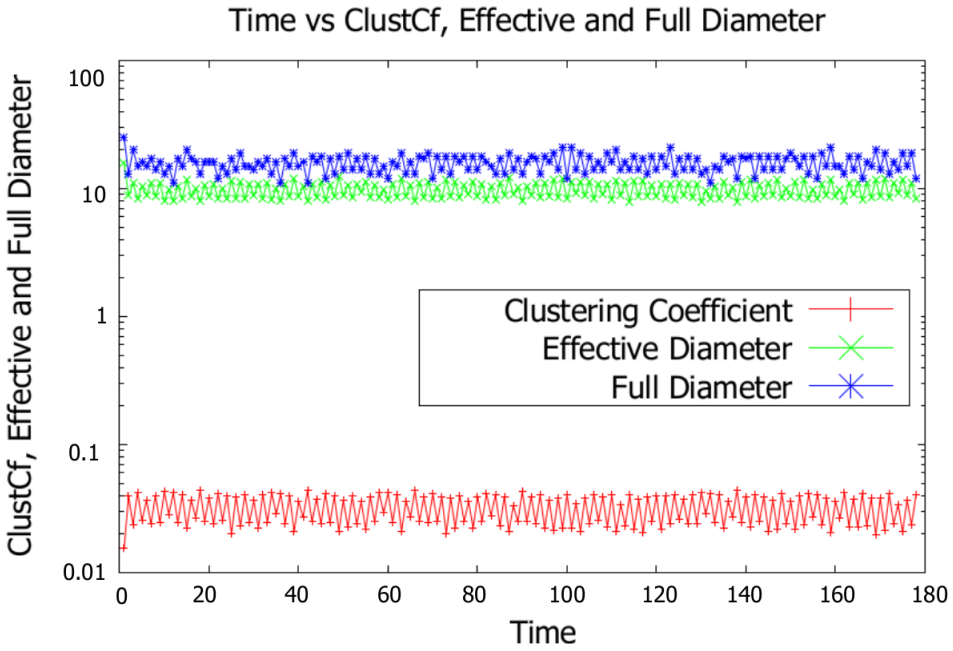


Figure 8.4: The temporal properties of Call graph on day and night time window.The first data point represents night graph.

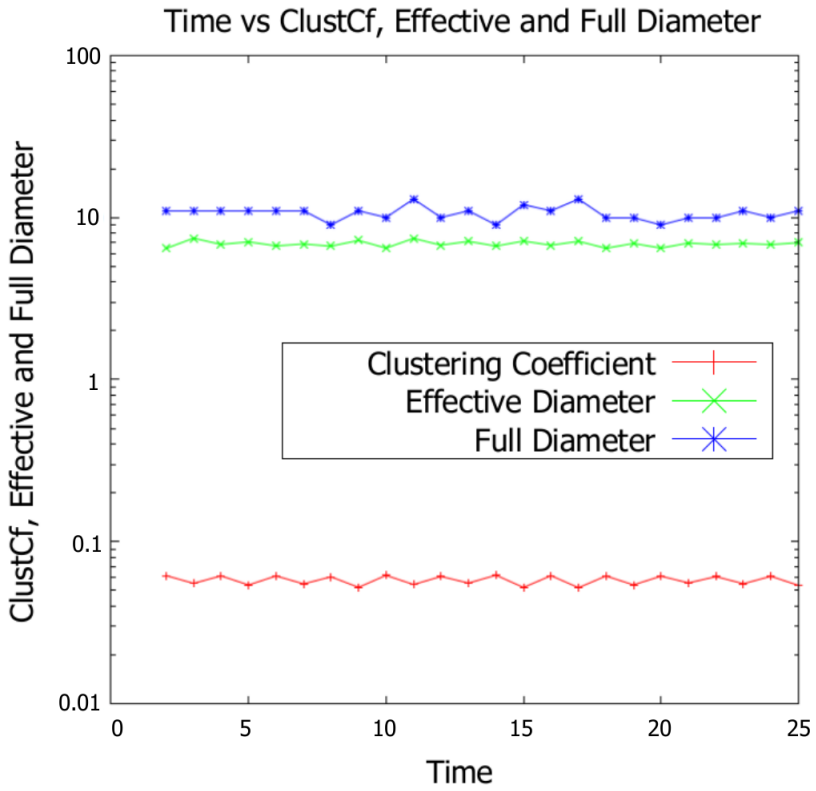


Figure 8.5: The temporal properties of Call graph on weekday and weekend time window. For all weekdays in a specific week, a call graph is created by aggregating all calls on that weekdays and various properties of that call graph are analyzed.

first 6 datapoints are same which is 11. The full diameter is calculated on the sampled graph in Stanford SNAP tool. The clustering coefficient also varies minutely across each weekdays and also for weekends. As with Uniform Day time window, the number of open triads are in orders of magnitude higher than closed triads. The values of number of edges, nodes and bidirectional are also roughly equal across each weekdays and for weekends signifying same level of macroscopic interactions occurring among people for each weekdays and each weekends.

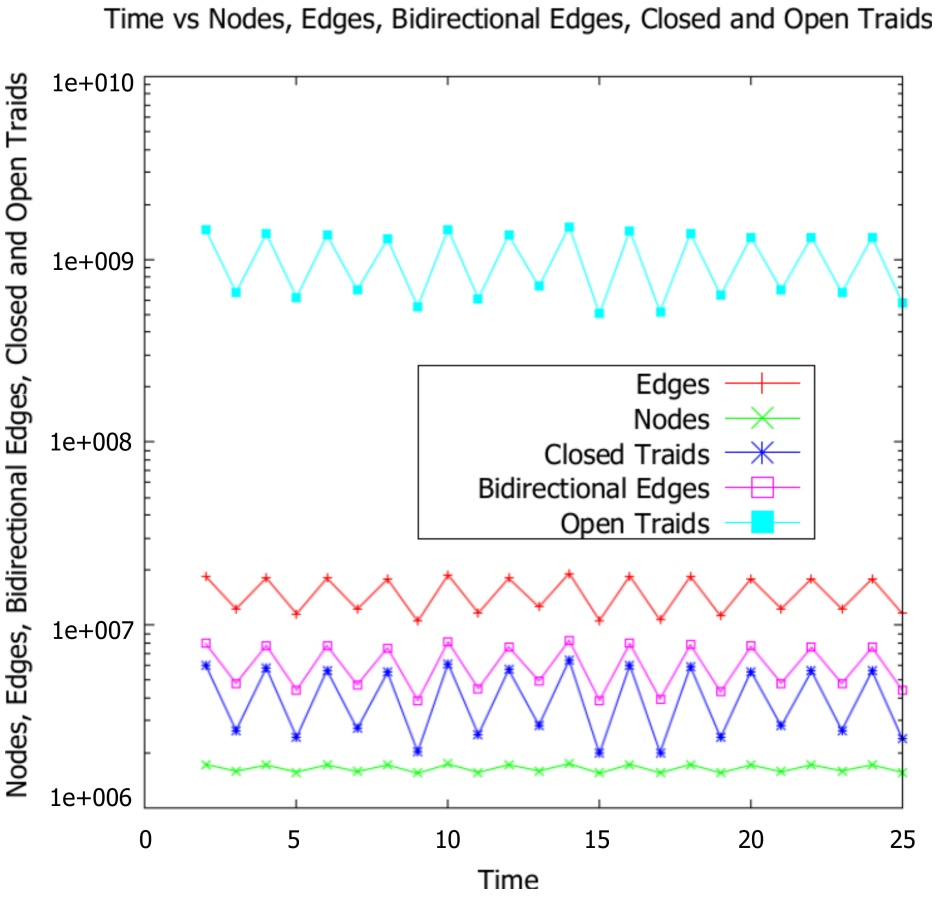


Figure 8.6: The temporal properties of Call graph on weekday and weekend time window. For all weekdays in a specific week, a call graph is created by aggregating all calls on that weekdays and various properties of that call graph are analyzed.

8.5.4 Cumulative Week time window

The temporal properties results of weekday and weekend time window is shown in Figure 8.8. The graph shows that calls gets saturated over two weeks period of time which implies people tend to call same group of people again and again. One such study [Onnela *et al.*, 2007] also reports saturation but over a period of two months.The change in full diameter and effective diameter is due to calculation of those values in sampled graph.

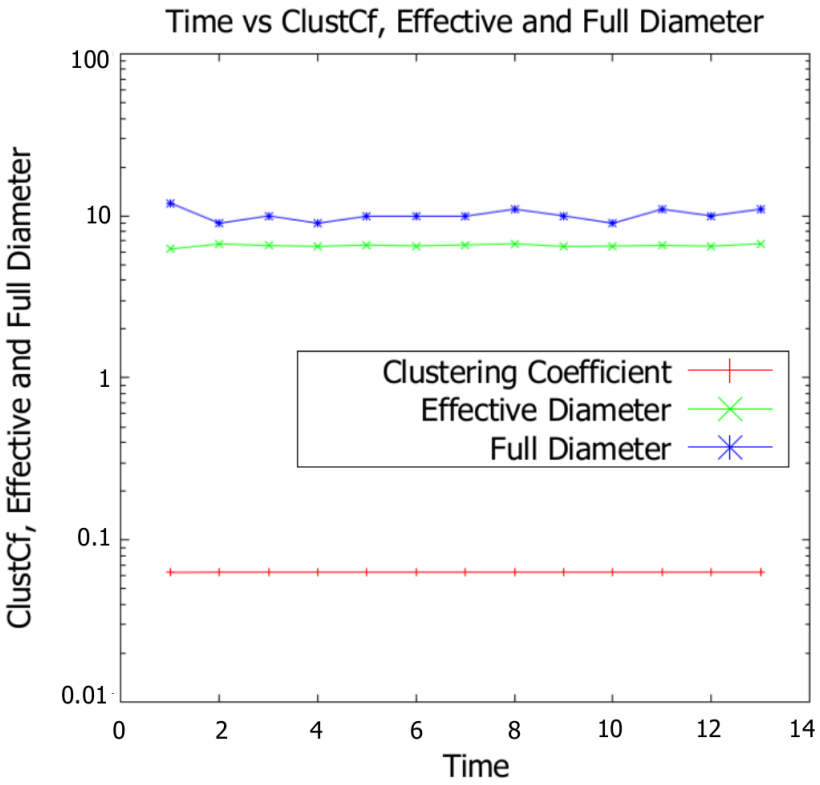


Figure 8.7: The temporal properties of Call graph on Consecutive week time window. For all calls initiated from week 0 to specific week are aggregated and graph is created for that week. This graph shows saturation of calls, implying people call same group of people again and again.

8.6 Choice of time window

The properties like clustering coefficient, diameter of call graph changes with the choice of time window. This leads to the question what is the right choice of time window for a graph? Our analysis showed in short size time window Day Night time window, anomalies can be easily detected but then weekly patterns and weak links between communities cannot be effectively captured. When size of time window is large, the anomalies seen in Day Night time window cannot be easily detected in Weekday Weekend time window. Hence the appropriate choice of size of time window depends on the study.

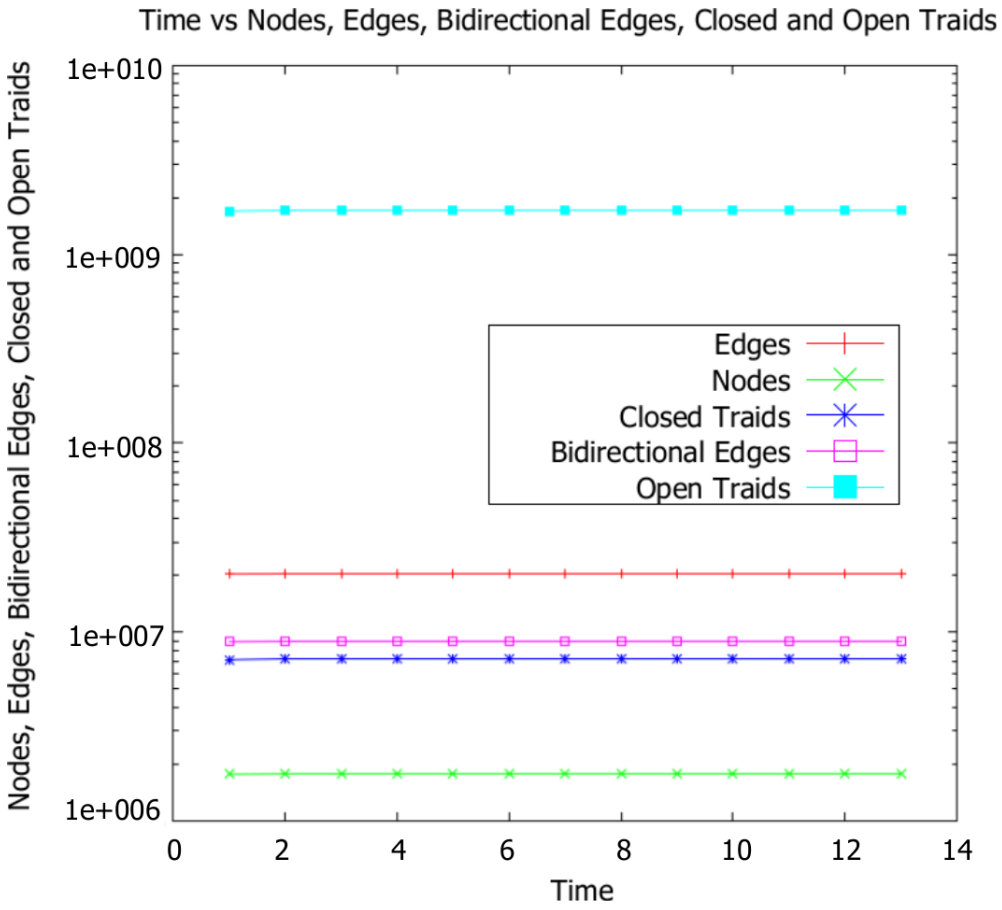


Figure 8.8: The temporal properties of Call graph on Consecutive week time window. For all calls initiated from week 0 to specific week are aggregated and graph is created for that week.

CHAPTER 9

Conclusions and Future Work

In this thesis, we studied an increasingly important problem of mining *communication motifs* from large dynamic interaction networks. Since each communication motif corresponds to a recurring subgraph with a similar sequence of information flow, it required us to venture into the exponential subgraph search space of the interaction network. To scale the mining framework, we proposed an algorithm called *COMMIT (COMmunication Motifs in InTeraction networks)*. COMMIT derives its pruning power from mapping the interaction network to a contractive sequence space. Following analysis in the sequence space, only a small set of subsequences are identified as likely candidates to represent communication motifs in the graphs. Thus, the expensive subgraphs enumeration and subgraph isomorphism tasks are performed only on a small set of likely candidates.

Extensive experiments on three social networks demonstrated COMMIT to be accurate and efficient. COMMIT is up to 2 orders of magnitude faster than existing techniques. Also, COMMIT can mine communication motifs of larger size with large time threshold whereas Naïve fails to mine large size motifs. In addition, a qualitative analysis of the communicative patterns reveal their unmatched power in distinguishing between social network through the role they play in the progression of interactions of their users.

The few areas in which COMMIT technique can be enhanced are as follows

• **Similar communication motifs:**

The COMMIT technique calculates the frequency of temporally connected subgraphs based on temporal isomorphism. So any two temporally connected subgraphs with a slight contrast in structure or temporal edge sequence will be considered as different subgraphs. A better way to capture interaction dynamics would be to mine representative communication motif. For this purpose, one has to come up with a similarity measure among temporally connected subgraphs.

• **Dynamic ΔT update**

In the proposed COMMIT technique, we require a time threshold parameter ΔT that determines the relatedness of two linked interactions. The ΔT is considered fixed for all the interactions. But, the relatedness of interactions depend heavily on the event, time period, people involved in interactions and many other factors. Also, in COMMIT, for determination of ΔT we define coverage metric but after choosing a specific value of ΔT we have to start the mining process from the initial stage. Hence, a dynamic way of updating ΔT would be good addition.

We also studied call detail records containing more than 1 billion calls using four different time window. The summary of inferences for specified four time windows are

- Day and Night time window study lead to anomaly detection in the number of calls on a specific day which might be due to holiday on that day.
- Uniform Day time window study lead to inference that no day dominates other day in terms of number of calls, number of nodes.
- Weekday and Weekend time window study lead to inference that macroscopic level properties recur with respect to weekdays and weekends.
- Cumulative Week time window study lead to inference about the saturation of graph over consecutive weeks.

We addressed the problem of mining recurring patterns of interactions which we call *communication motifs* and propose COMMIT technique to mine them. With no other scalable technique present in literature for mining communication motifs in large networks, COMMIT opens up a new direction in motif mining.

REFERENCES

[Albert and Albert, 2004] IstvÃąn Albert and RÃl'ka Albert. Conserved network motifs allow protein-protein interaction prediction. *Bioinformatics*, 20(18):3346–3352, 2004.

[Allan *et al.*, 2009] E.G. Allan, W.H. Turkett, and E.W. Fulp. Using network motifs to identify application protocols. In *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pages 1–7, Nov 2009.

[Borgwardt *et al.*, 2006] K.M. Borgwardt, H.-P. Kriegel, and P. Wackersreuther. Pattern mining in frequent dynamic subgraphs. In *ICDM*, pages 818–822, 2006.

[Braha and Bar-Yam, 2009] Dan Braha and Yaneer Bar-Yam. Time-dependent complex networks: Dynamic centrality, dynamic motifs, and cycles of social interactions. In *Adaptive Networks*, pages 39–50. Springer, 2009.

[Bringmann and Nijssen, 2008] Björn Bringmann and Siegfried Nijssen. What is frequent in a single graph? In *Proceedings of the 12th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, PAKDD'08, pages 858–863, Berlin, Heidelberg, 2008. Springer-Verlag.

[Bruno *et al.*, 2010] Francesco Bruno, Luigi Palopoli, and Simona E Rombo. New trends in graph mining: Structural and node-colored network motifs. *International Journal of Knowledge Discovery in Bioinformatics (IJKDB)*, 1(1):81–99, 2010.

[Chechik *et al.*, 2008] Gal Chechik, Eugene Oh, Oliver Rando, Jonathan Weissman, Aviv Regev, and Daphne Koller. Activity motifs reveal principles of timing in

transcriptional control of the yeast metabolic network. *Nature biotechnology*, 26(11):1251–1259, 2008.

[Ciriello and Guerra, 2008] Giovanni Ciriello and Concettina Guerra. A review on models and algorithms for motif discovery in protein–protein interaction networks. *Briefings in functional genomics & proteomics*, 7(2):147–156, 2008.

[Ding *et al.*, 2009] Bolin Ding, David Lo, Jiawei Han, and Siau-Cheng Khoo. Efficient mining of closed repetitive gapped subsequences from a sequence database. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 1024–1035. IEEE, 2009.

[Elseidy *et al.*, 2014] Mohammed Elseidy, Ehab Abdelhamid, Spiros Skiadopoulos, and Panos Kalnis. Grami: Frequent subgraph and pattern mining in a single large graph. *Proceedings of the VLDB Endowment*, 7(7), 2014.

[enr,] ENRON, http://www.cs.cmu.edu/˜enron/.

[Gallos *et al.*, 2012] Lazaros K. Gallos, Diego Rybski, Fredrik Liljeros, Shlomo Havlin, and Hernán A. Makse. How people interact in evolving online affiliation networks. *Phys. Rev. X*, 2:031014, Aug 2012.

[Junttila and Kaski, 2007] Tommi Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In David Applegate, Gerth Stølting Brodal, Daniel Panario, and Robert Sedgewick, editors, *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithms and Combinatorics*, pages 135–149. SIAM, 2007.

[Jurgens and Lu, 2012] David Jurgens and Tsai-Ching Lu. Temporal motifs reveal the dynamics of editor interactions in wikipedia. In *ICWSM*, 2012.

[Kashani *et al.*, 2009] Zahra RM Kashani, Hayedeh Ahrabian, Elahe Elahi, Abbas Nowzari-Dalini, Elnaz S Ansari, Sahar Asadi, Shahin Mohammadi, Falk Schreiber, and Ali Masoudi-Nejad. Kavosh: a new algorithm for finding network motifs. *BMC bioinformatics*, 10(1):318, 2009.

[Kashtan *et al.*, 2004] Nadav Kashtan, Shalev Itzkovitz, Ron Milo, and Uri Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20(11):1746–1758, 2004.

[Ketkar *et al.*, 2005] Nikhil S. Ketkar, Lawrence B. Holder, and Diane J. Cook. Subdue: Compression-based frequent pattern discovery in graph data. In *Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*, OSDM '05, pages 71–76, New York, NY, USA, 2005. ACM.

[Kovanen *et al.*, 2011] Lauri Kovanen, Márton Karsai, Kimmo Kaski, János Kertész, and Jari Saramäki. Temporal motifs in time-dependent networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2011(11):P11005, 2011.

[Kovanen *et al.*, 2013] Lauri Kovanen, Kimmo Kaski, János Kertész, and Jari Saramäki. Temporal motifs reveal homophily, gender-specific patterns, and group talk in call sequences. *Proceedings of the National Academy of Sciences*, 110(45):18070–18075, 2013.

[Krings *et al.*, 2012] Gautier Krings, Márton Karsai, Sebastian Bernhardsson, Vincent D Blondel, and Jari Saramäki. Effects of time window size and placement on the structure of an aggregated communication network. *EPJ Data Science*, 1(1):4, May 2012.

[Kuramochi and Karypis, 2001] Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *ICDM*, pages 313–320, 2001.

[Kuramochi and Karypis, 2005] Michihiro Kuramochi and George Karypis. Finding frequent patterns in a large sparse graph. *Data mining and knowledge discovery*, 11(3):243–271, 2005.

[Leskovec *et al.*, 2005] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *Proc. of KDD'05*, 2005.

[Liu *et al.*, 2012] Kai Liu, William K Cheung, and Jiming Liu. Detecting multiple stochastic network motifs in network data. In *Advances in Knowledge Discovery and Data Mining*, pages 205–217. Springer, 2012.

[McKay and others, 1981] Brendan D McKay et al. *Practical graph isomorphism*. Department of Computer Science, Vanderbilt University, 1981.

[Milo *et al.*, 2002] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

[Milo *et al.*, 2004] Ron Milo, Shalev Itzkovitz, Nadav Kashtan, Reuven Levitt, Shai Shen-Orr, Inbal Ayzenshtat, Michal Sheffer, and Uri Alon. Superfamilies of evolved and designed networks. *Science*, 303(5663):1538–1542, 2004.

[Nanavati *et al.*, 2008] Amit Anil Nanavati, Rahul Singh, Dipanjan Chakraborty, Koustuv Dasgupta, Sougata Mukherjea, Gautam Das, Siva Gurumurthy, and Anupam Joshi. Analyzing the structure and evolution of massive telecom graphs. *IEEE Trans. Knowl. Data Eng.*, 20(5):703–718, 2008.

[O'Callaghan *et al.*, 2012] Derek O'Callaghan, Martin Harrigan, Joe Carthy, and PÃądraig Cunningham. Network analysis of recurring youtube spam campaigns. *CoRR*, abs/1201.3783, 2012.

[Onnela *et al.*, 2007] J.-P. Onnela, J. SaramÃd'ki, J. HyvÃűnen, G. SzabÃş, D. Lazer, K. Kaski, J. KertÃl'sz, and A.-L. BarabÃąsi. Structure and tie strengths in mobile communication networks. *Proceedings of the National Academy of Sciences*, 104(18):7332–7336, 2007.

[Pei *et al.*, 2001] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 0215–0215. IEEE Computer Society, 2001.

[Ranu and Singh, 2009] Sayan Ranu and Ambuj K Singh. Mining statistically significant molecular substructures for efficient molecular classification. *Journal of chemical information and modeling*, 49(11):2537–2550, 2009.

[Ranu *et al.*, 2011] Sayan Ranu, Bradley T Calhoun, Ambuj K Singh, and S Joshua Swamidass. Probabilistic substructure mining from small-molecule screens. *Molecular Informatics*, 30(9):809–815, 2011.

[Ranu *et al.*, 2013] Sayan Ranu, Minh Hoang, and Ambuj Singh. Mining discriminative subgraphs from global-state networks. In *SIGKDD*, pages 509–517, 2013.

[Shen-Orr *et al.*, 2002] Shai S Shen-Orr, Ron Milo, Shmoolik Mangan, and Uri Alon. Network motifs in the transcriptional regulation network of escherichia coli. *Nature genetics*, 31(1):64–68, 2002.

[sna,] SNAP, http://snap.stanford.edu/.

[Spearman, 1904] Charles Spearman. The proof and measurement of association between two things. *The American journal of psychology*, 15(1):72–101, 1904.

[Viswanath *et al.*, 2009] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. On the evolution of user interaction in facebook. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (WOSN'09)*, August 2009.

[Wernicke, 2006] Sebastian Wernicke. Efficient detection of network motifs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 3(4):347–359, 2006.

[Yan and Han, 2002] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002.

[Zeng *et al.*, 2009] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *PVLDB*, 2(1), 2009.

[Zhao *et al.*, 2010] Qiankun Zhao, Yuan Tian, Qi He, Nuria Oliver, Ruoming Jin, and Wang-Chien Lee. Communication motifs: a tool to characterize social communications. In *CIKM*, pages 1645–1648. ACM, 2010.

[Zhu *et al.*, 2007] Feida Zhu, Xifeng Yan, Jiawei Han, and Philip S. Yu. gprune: A constraint pushing framework for graph pattern mining. In *Proceedings of the 11th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, PAKDD'07, pages 388–400, Berlin, Heidelberg, 2007. Springer-Verlag.

Publications and Patents

Publication

1. S. Gurukar, S. Ranu and B. Ravindran, “COMMIT : A Scalable Approach to Mining Communication Motifs from Dynamic Networks”, SIGMOD, 2015.

2. S. Gurukar and B. Ravindran, “Temporal Analysis of Telecom Graphs”, Proceedings of the Social Networking Workshop at COMSNETS, 2014.

Patent

1. S. Gurukar, S. Ranu, B. Ravindran, S. Subramanian and A. Dauneria. "Temporal Motif Based Approach To Analyze Devices Reconnection Patterns". *United States PCT/SE2014/051303 (Application Number)*, 2014