COMMIT : A Scalable Approach to Mining Communication Motifs from Dynamic Networks

Saket Gurukar
Dept. of CSE
IIT Madras
Chennai, India.
saket@cse.iitm.ac.in

Sayan Ranu
Dept. of CSE
IIT Madras
Chennai, India.
sayan@cse.iitm.ac.in

Balaraman Ravindran
Dept. of CSE
IIT Madras
Chennai, India.
ravi@cse.iitm.ac.in

ABSTRACT

A fundamental problem in behavioral analysis of human interactions is to understand how communications unfold. In this paper, we study this problem by mining *Communication motifs* from dynamic interaction networks. A communication motif is a recurring subgraph that has a similar sequence of information flow. 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. Extensive experiments on three different social networks show COMMIT to be up to two orders of magnitude faster than baseline techniques. 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.

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 paper, we study the behavioral aspects of interactions within social networks by mining *communication motifs* from large dynamic networks.

To illustrate the concept of communication motifs in a dynamic network, consider Fig. 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. 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, 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(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. 4. 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(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 first explored by Zhao et al. [39]. 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

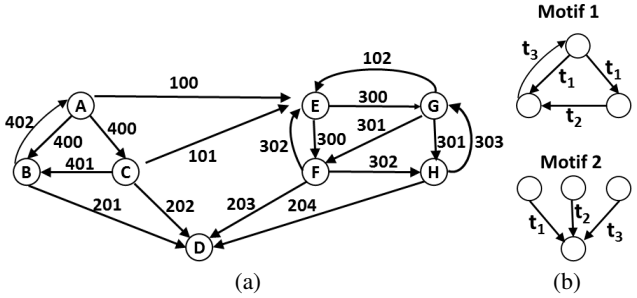


Figure 1: (a) A dynamic network denoting interactions between its users (b) The two largest communication motifs at a frequency threshold of 3 and $\Delta T = 1$. Timestamp $t_i < t_j$ if $i < j$. Motif 1 involves nodes $\{A, B, C\}$ $\{E, F, G\}$, and $\{G, H, F\}$. Motif 2 involves $\{B, C, D, F\}$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

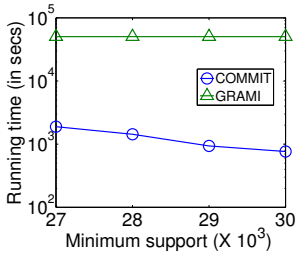


Figure 2: Running time comparison of GRAMI and COMMIT against the support threshold on the Facebook dataset [35]. Note that GRAMI was terminated after 16 hours in all cases without having completed the computation.

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. 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 [38]. Owing to its hardness, frequent subgraph mining has received significant interest in the research community with GRAMI [13] 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 subgraphs 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(a) and 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 [35]. Fig. 2 presents the results. On this network, when the support threshold is less than $30,000$, GRAMI fails to complete even after 16 hours.

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. 2, COMMIT takes around 20 minutes to complete. Fig. 3 presents the pipeline of our 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.

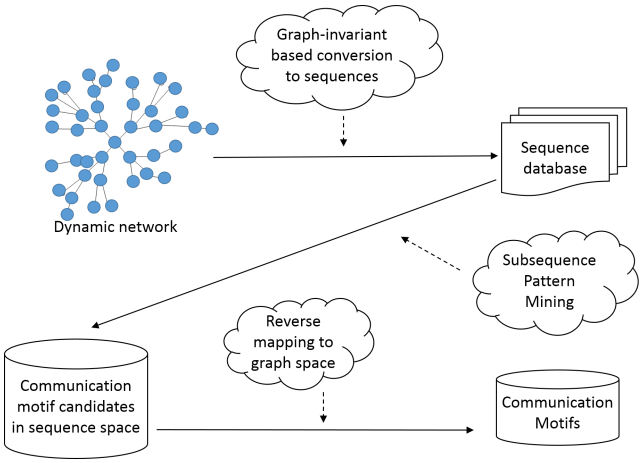


Figure 3: Pipeline of the COMMIT algorithm.

Our 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 our 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.

To summarize, the contributions of our work 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.

2. PROBLEM FORMULATION

In this section, 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.

To formalize the concept of communication motifs, first we define the idea of temporally related edges. Informally, two edges are related if they involve a common user and are close in time. An

[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.

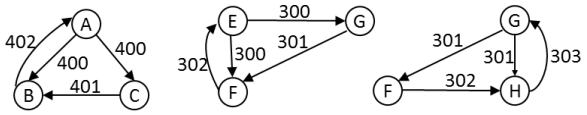


Figure 4: Embeddings of Motif 1 in Fig. 1(a)

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 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 3. 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. 4, which shows the embeddings of Motif 1 in Fig. 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 [39] and rectifies a weakness in their modeling. Interested readers are encouraged to refer to Appendix A to know more.

Now, we 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 4. 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. 4 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.

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(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. We point interested readers to Appendix B for further discussion.

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

DEFINITION 5. 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.

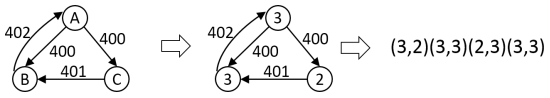


Figure 5: Sequence representation of a graph.

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.

3. MAPPING GRAPHS TO SEQUENCES.

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 [38]. To tackle this bottleneck, COMMIT first maps the dynamic network from the graph space to a sequence space.

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.

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

DEFINITION 6. 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. [21, 37]. 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. 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. 5 shows the sequence representation of the first graph in Fig. 4. It is easy to see, that since the other two graphs in Fig. 4 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.

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. 4. 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 sub-sequence 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 7. 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 8. 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 $\mathcal{M}(\mathcal{H})$, $\forall e_{h_i}, e_{h_j} \in E_H$, 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, $\mathcal{M}(\mathcal{H}) \sqsubseteq \mathcal{M}(\mathcal{G})$. □

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.

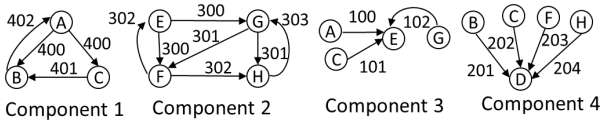


Figure 6: The temporally connected components in Fig. 1(a).

4. FREQUENT SUBSEQUENCE MINING

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

DEFINITION 9. 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 $H' \subseteq G$.*

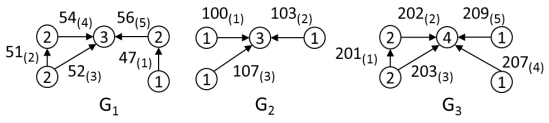
EXAMPLE 3. *The temporally connected components of the network in Fig. 1(a) at $\Delta T = 1$ are shown in Fig. 6.*

The pseudocode to identify the temporally connected components is provided in Appendix. C. 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.

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(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 well as within a sequence. Mining such frequent subsequences in a sequence database has been studied, and *CloGSgrow* [12] is the state-of-the-art technique for this purpose. CloGSgrow is an extension of PrefixSpan [28] 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.

4.1 Counting support of a subsequence

Assume the graphs in Fig. 7 are the connected components of the network we are mining. Their corresponding sequence representations are shown in the Sequence DB table in Fig. 7. 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. 7. These three instances of P in S_3 correspond to the instances with ID 3 in the $SeqDB(P)$ table of Fig. 8. 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 $(3, < 2, 3, 4 >)$,



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 7: 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 described in Sec. 3.

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 10. 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. 8 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 $sup(P) = |SeqDB(P)|$ is NP-complete.*

PROOF: See Appendix D.

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 11. 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. 7 are shown in Fig. 8. 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. 4.2.2. 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 instance $I = (x, < l_1, \cdots, l_{i-1}, l_{i+1}, \cdots, l_m >$ 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, $sup(P) \geq sup(P')$. □

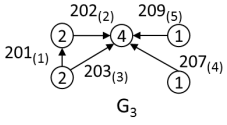
Theorem 3 establishes apriori property of the support set when only non-identically overlapping instances are recorded.

4.2 The sequence growth approach

Sequence growth is a popular strategy to search for sequences in the presence of apriori property [12, 21, 28, 37].

DEFINITION 12. 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



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

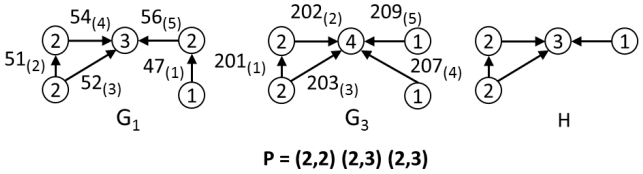


Figure 9: Illustration of the need for EXTENSIONMINER.

till the sequence becomes infrequent. The key question therefore is how do we identify the extensions?

4.2.1 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. 7. 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. 9. 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 13. 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.

Algorithm 1 ExtensionMiner $(el, \mathbb{S}, b, \tau)$

Input: $\mathbb{S}$ is support set of edge label el, b is starting position, τ is support threshold.
Output: $\mathbb{E}$ 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)
```

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 $\forall k,\ 1 \leq k \leq sup(\alpha)$. 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. Hence proved. □

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 from these phenomenon, we define the concept of *closed* edge extensions.

DEFINITION 14. 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 define *floor* of edge labels.

DEFINITION 15. 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. 1 presents the EXTENSIONMINER algorithm. Fig. 10 presents a running example of the algorithm. EX-

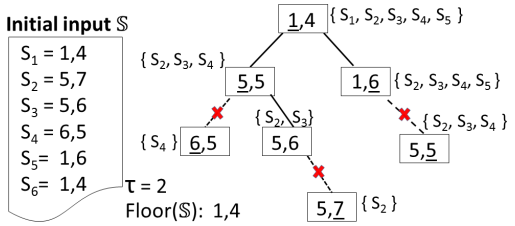


Figure 10: A running example of EXTENSIONMINER. The closed edge extensions from the given collection $\mathbb{S}$ of all edge labels correspond to the floors in the non-leaf states. Specifically, $(1,4)$, $(5,5)$, $(5,6)$, and $(1,6)$. The underlined dimension indicates the value of b in that state.

TENSIONMINER 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. 10.

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 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. 10). 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.

4.2.2 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. 4.1, 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 [12]. Here, we briefly summarize the algorithm. The correctness proofs are available in CloGSgrow [12].

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

DEFINITION 16. 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 temporally connected components in Fig. 7 (also shown in Fig. 11). Now, consider the subsequence $P = (1,3)(1,3)(1,3)$. With the sequence growth technique, the generation of this subsequence would start from the edge label $(1,3)$.

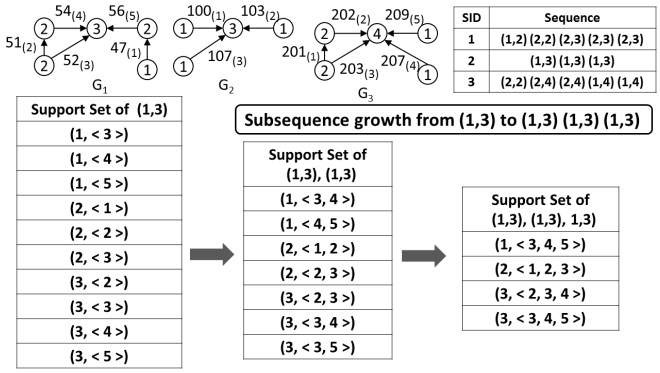


Figure 11: Illustration of sequence growth from $(1,3)$ to $(1,3)(1,3)(1,3)$. The support sets are maintained in right shift order, which allows polynomial-time support counting.

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. 11 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 E.

Revisiting the example in Fig. 11, 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 subgraph enumeration and subgraph isomorphism. This allows us to achieve scalability without compromising significantly on quality. We verify this empirically in Sec. 6.

5. COMMIT

Finally, we have all the pieces required to mine *communication motifs* scalably. We next discuss the mining pipeline of COMMIT.

Alg. 2 presents the pseudocode. Given a sequence database dynamic network G, support threshold τ and temporal threshold ΔT, first, all temporally connected components in the network are identified (line 1 in Alg. 2). 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

Algorithm 2 COMMIT $(G, \tau, \Delta T)$

Input: A dynamic interaction network G, support threshold τ, and temporal threshold ΔT.
Output: *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 each** 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 each** subsequence $P_j \in \mathbb{P}$ **do**
10: $FreqP, FreqSS \leftarrow$ **SeqGrow(** $SeqDB, P_j, SS_j, \Delta T$**)**
11: **for each** $I \in FreqSS$ **do**
12: $\mathbb{A}^+ \leftarrow$**MotifMine(** $SeqDB, SS$**)**
13: $\mathbb{A} \leftarrow$Remove false positives from $\mathbb{A}^+$.
14: **return** $\mathbb{A}$

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.

Alg. 3 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. 7). 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. 2). The procedure to check temporal subgraph isomorphism is detailed in Appendix F.

Algorithm 3 SeqGrow $(SeqDB, P, SS, \Delta T, \tau)$

Input: A sequence database $SeqDB = \{S_1, S_2, ..., S_N\}$, P is subsequence, SS the support set of P.
Output: $FreqP$ 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 each** instance $(i, < l_1, l_2, ..., l_j >) \in SS$ **do**
5: $\mathbb{S} \leftarrow \{S_i[l_k] \mid l_k > l_j$ and $| t_l - t_j | \leq \Delta T\}$
6: $f \leftarrow Floor(\mathbb{S})$
7: $\mathbb{E} \leftarrow$ **ExtensionMiner(** $f, \mathbb{S}, 0, \tau$**)**
8: **for each** edge $e \in \mathbb{E}$ **do**
9: $P_i^+ \leftarrow P \circ e$
10: $SS_i^+ \leftarrow$ **GetSup**$(SeqDB, P, SS, e)$ \\See Appendix E for **GetSup()**
11: $\mathbb{P} \leftarrow P_i^+, \mathbb{SS} \leftarrow SS_i^+$
12: **for each** $P_i^+, SS_i^+ \in \mathbb{P}, \mathbb{SS}$ **do**
13: **SeqGrow(** $SeqDB, P_i^+, SS_i^+, \Delta T, \tau$**)**

Algorithm 4 MotifMine $(SeqDB, SS)$

Input: A sequence database $SeqDB = \{S_1, S_2, ..., S_N\}$, support set SS.
Output: Motif with their frequencies
1: **for each** 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

6. EXPERIMENTS

In this section, 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.

6.1 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. Proposed algorithms are written in C++ with -O3 flag.

6.1.1 Datasets

We evaluate our COMMIT on the three social network datasets in Table 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.

6.1.2 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* [13]. 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. 2 remain the same. To ensure that the top-k set is not overloaded with small motifs, we consider motifs of size at least 3.

ΔT is an important parameter in our model and controls whether

Dataset	Number of Nodes	Number of Edges	Duration (days)
Twitter [3]	4,978,421	26,526,180	30
Facebook [35]	45,813	855,539	1540
Enron Email Network [1]	10833	77050	349

Table 1: Summary of the datasets.

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. 12(a)-12(b) present the average growth rate in the subgraph sizes as ΔT is varied in a range $[t_{min}, t_{max}]$, where

$$Coverage(\Delta T) = \frac{\text{subgraph size at } \Delta T}{\text{subgraph size at } t_{max}} \quad (1)$$

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 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 Appendix G.

6.2 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 main memory. We represent the size of communication motifs as the number of interactions involved in it. 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. 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 $m - 1$. 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 [2]. 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. 12(c)-12(e) 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 support 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 the Facebook dataset, the accuracy is invariant with ΔT, specifically at higher ks where the ranking stabilizes. On Facebook, the accuracies are slightly lower at $\Delta T = 15$ mins and $\Delta T = 30$ mins 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 [34] 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. 12(f)-12(h) presents the results against k on multiple $\Delta T s$. 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 $\Delta T = 15$ mins and $\Delta T = 30$ mins. As discussed earlier, 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 increase with k.

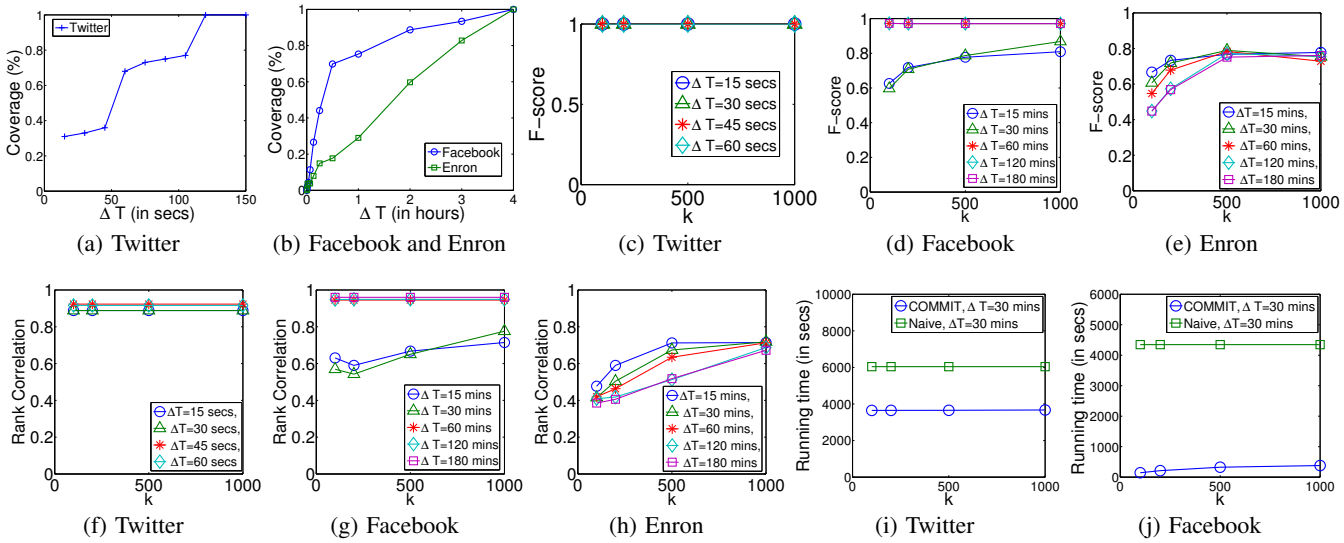
To summarize, both the F-score and rank correlation improve with k and saturate around 0.80 on Facebook and Enron. On Twitter, Naïve fails to scale.

6.3 Scalability of COMMIT

After establishing the accuracy of COMMIT, we next focus on its scalability. 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 setting, 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 they analyzes 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.

6.3.1 *Top-k queries*

First, we benchmark the performance of COMMIT against k. Figs. 12(i), 12(j),13(a) 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



(a) Twitter (b) Facebook and Enron (c) Twitter (d) Facebook (e) Enron

(f) Twitter (g) Facebook (h) Enron (i) Twitter (j) Facebook

Figure 12: Growth rate of coverage with ΔT in (a) Twitter and (b) Facebook and Enron. Analysis of F-score with k on (c) Twitter (d) Facebook and (e) Enron. k vs Spearman's rank correlation on (f) Twitter (g) Facebook and (h) Enron. Growth rate of running time with k in (i) Twitter, (j) Facebook.

to mine all communications motifs of size 3 in Twitter and size 4 in Facebook. In other words, the experiments only provides 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 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. 13(b), the running time is a straight line. Fig. 13(c) demonstrates the performance in Facebook. As can be seen, there is an exponential growth in the running time of Naïve. At 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 due to the bulk of the processing happening 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

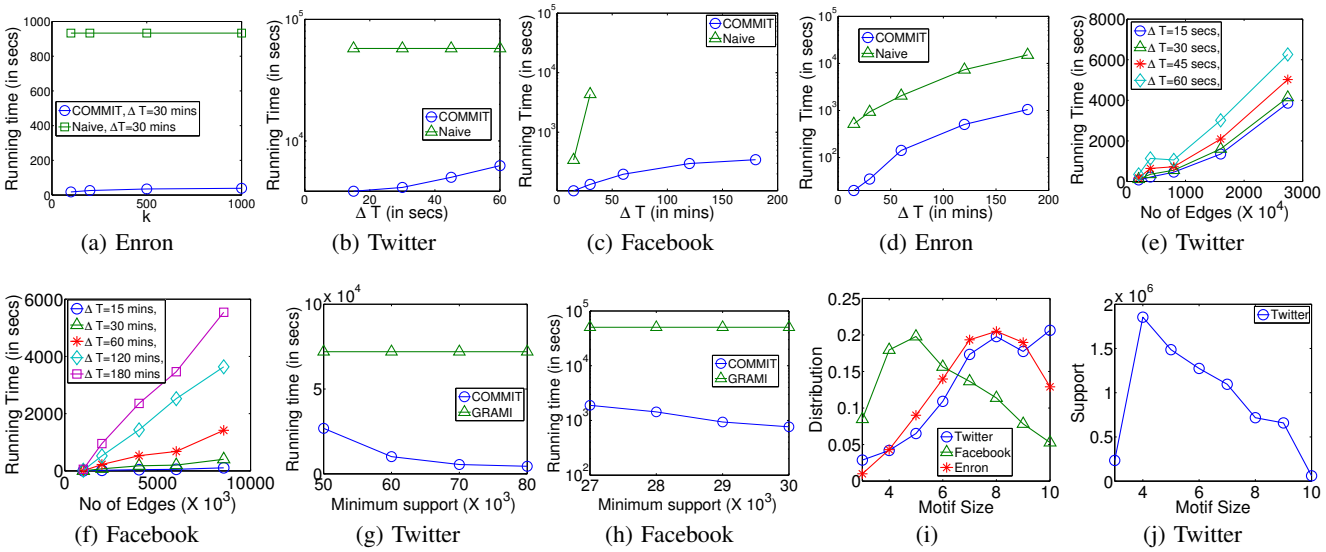
Finally, we look at the growth rate of running time against the size of the interaction network. Figs 13(e)-13(f) presents the results on a series of ΔTs. On both datasets, the growth rates resemble a linear curve. On twitter the growth rate is higher since it is more dense. We ignore the ENRON dataset for this experiment since it is the smallest.

6.3.2 *Range query*

We now concentrate on the range query performance of COMMIT. For benchmarking purposes, we compare the running time of COMMIT with GRAMI [13]. Note that the answer sets of GRAMI and COMMIT are different. GRAMI mines frequent subgraphs. However, as illustrated earlier, these frequent subgraphs can subsequently 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 [13], 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 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. 13(g) 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 an 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. 13(h) demonstrates the results. In contrast, COMMIT finishes within 30 minutes across all values of τ in Fig. 13(h). Overall, COMMIT is more than 70 times faster than GRAMI.

6.3.3 *Distribution of motif sizes*

Secs. 6.3.1 and 6.3.2 show that Naïve can somewhat scale when the motif sizes are small; specifically, motifs of size 3 in Twitter



(a) Enron (b) Twitter (c) Facebook (d) Enron (e) Twitter

(f) Facebook (g) Twitter (h) Facebook (i) (j) Twitter

Figure 13: (a) Growth rate of running time with k in Enron. Growth rate of running time with ΔT in (b) Twitter, (c) Facebook and (d) Enron. Growth rate of the running time against the size of the interaction network in (e) Twitter and (f) Facebook. Growth rate of running time against the support threshold in the range query setting. Running time comparison of GRAMI and COMMIT against the support threshold on (g) Twitter and (h) Facebook. Distribution of motif sizes (i) and their supports (j).

and size 4 in Facebook and Enron. In this section, we investigate whether motifs of larger sizes occur in interaction networks. Fig. 13(i) demonstrates the distribution of motifs with respect to their sizes in the top-10000 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. 13(j) shows the result in Twitter, which is the largest interaction network among the three. The support distributions in Facebook and Enron are shown in Appendix H. In Twitter, the total support 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. 13(i) and 13(j) 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. 14, are of size 3. On the other hand, the number of larger motifs in the top-10000 list, such as those of size 10, is higher since combinatorially, the space of size-10 motifs is larger than size-4 motifs.

6.4 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. 14. We restrict the discussion to motifs of size 3 just for the sake of simplicity.

6.4.1 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 denotes 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.

6.4.2 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).

6.4.3 Enron

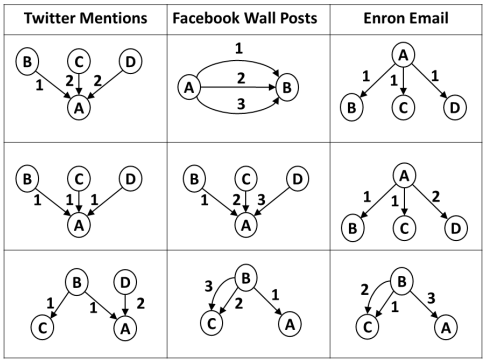


Figure 14: Top-3 communication motifs.

The fact that Enron is an email network is clearly evident from the top-3 patterns in Fig. 14. 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 [1]. 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).

6.4.4 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, predicting the nature of communication taking place such as birthday wish, group discussion etc. Furthermore, communication motifs reveal that the underlying social network has a strong influence on how people interact. In a previous study, Kovanen et al. [20], showed 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-complete. 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. 6.2 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 however, 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.

7. RELATED WORK

Generally, network motifs are statistically significant subgraphs that occur more frequently in the original network as compared to randomized networks. Quantifying the significance of motifs varies from application to application [11, 31]. Nonetheless, mining network motifs forms the backbone of various applications such as spam detection [27], protein-protein link prediction [4], analyzing human interactions [14], network classification [5, 29, 30, 32].

Mining frequent subgraphs from single network is widely studied [6,8,13,18,21,22,37,40]. Milo et al. [26] proposed that network motifs might be the evolutionary backbone of a network [26, 33]. Milo et al. [25] 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.

The network motif detection basically consists of three steps: 1) Enumerate subgraphs of a given size 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 [24]. 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.

In literature, there are a number of surveys [9, 11] on static network motif detection algorithms. Kashtan et al. [17] propose an edge sampling based algorithm, mfinder, to estimate subgraph counts. Sebastian Wernicke et al. [36] develop a sampling based motif detection algorithm, FANMOD, for estimating number of subgraphs and does not suffer from sampling bias as in mfinder [17]. Zahra Razaghi et al. [16] propose a method for enumerating subgraphs called *kavosh*. Since we have a dynamic interaction network, static motif detection techniques do not apply to our problem.

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 [7]. 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. [10] propose the idea of *activity motifs* to analyze transcription in yeast organism metabolism. David Jurgens et al. [15] analyzed interactions of Wikipedia editors to identify significant patterns by constructing a temporal bipartite network. Kai Liu et al. [23] 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. [19] and Zhao et al. [39]. The model proposed in [19] fails on network where a person can communicate simultaneously (time overlapping edges) and hence fails to solve the proposed problem. The model in [39] 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.

8. CONCLUSION

In this paper, 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. 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. 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. Overall, COMMIT opens up a new direction in motif mining by unleashing the potential of communication motifs.

9. REFERENCES

[1] ENRON, http://www.cs.cmu.edu/~enron/.

[2] http://en.wikipedia.org/wiki/f1_score.

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

[4] I. Albert and R. Albert. Conserved network motifs allow protein-protein interaction prediction. *Bioinformatics*, 20(18):3346–3352, 2004.

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

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

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

[8] B. Bringmann and S. 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.

[9] F. Bruno, L. Palopoli, and S. 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.

[10] G. Chechik, E. Oh, O. Rando, J. Weissman, A. Regev, and D. Koller. Activity motifs reveal principles of timing in transcriptional control of the yeast metabolic network. *Nature biotechnology*, 26(11):1251–1259, 2008.

[11] G. Ciriello and C. 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.

[12] B. Ding, D. Lo, J. Han, and S.-C. 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.

[13] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis. Grami: Frequent subgraph and pattern mining in a single large graph. *Proceedings of the VLDB Endowment*, 7(7), 2014.

[14] L. K. Gallos, D. Rybski, F. Liljeros, S. Havlin, and H. A. Makse. How people interact in evolving online affiliation networks. *Phys. Rev. X*, 2:031014, Aug 2012.

[15] D. Jurgens and T.-C. Lu. Temporal motifs reveal the dynamics of editor interactions in wikipedia. In *ICWSM*, 2012.

[16] Z. R. Kashani, H. Ahrabian, E. Elahi, A. Nowzari-Dalini, E. S. Ansari, S. Asadi, S. Mohammadi, F. Schreiber, and A. Masoudi-Nejad. Kavosh: a new algorithm for finding network motifs. *BMC bioinformatics*, 10(1):318, 2009.

[17] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20(11):1746–1758, 2004.

[18] N. S. Ketkar, L. B. Holder, and D. 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.

[19] L. Kovanen, M. Karsai, K. Kaski, J. Kertész, and J. Saramäki. Temporal motifs in time-dependent networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2011(11):P11005, 2011.

[20] L. Kovanen, K. Kaski, J. KertÃľsz, and J. SaramÃd'ki. 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.

[21] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *ICDM*, pages 313–320, 2001.

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

[23] K. Liu, W. K. Cheung, and J. Liu. Detecting multiple stochastic network motifs in network data. In *Advances in Knowledge Discovery and Data Mining*, pages 205–217. Springer, 2012.

[24] B. D. McKay et al. *Practical graph isomorphism*. Department of Computer Science, Vanderbilt University, 1981.

[25] R. Milo, S. Itzkovitz, N. Kashtan, R. Levitt, S. Shen-Orr, I. Ayzenshtat, M. Sheffer, and U. Alon. Superfamilies of evolved and designed networks. *Science*, 303(5663):1538–1542, 2004.

[26] 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.

[27] D. O'Callaghan, M. Harrigan, J. Carthy, and P. Cunningham. Network analysis of recurring youtube spam campaigns. *CoRR*, abs/1201.3783, 2012.

[28] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. 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.

[29] S. Ranu, B. T. Calhoun, A. K. Singh, and S. J. Swamidass. Probabilistic substructure mining from small-molecule screens. *Molecular Informatics*, 30(9):809–815, 2011.

[30] S. Ranu, M. Hoang, and A. Singh. Mining discriminative subgraphs from global-state networks. In *SIGKDD*, pages 509–517, 2013.

[31] S. Ranu and A. K. Singh. Graphsig: A scalable approach to mining significant subgraphs in large graph databases. In *ICDE*, pages 844–855, 2009.

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

[33] S. S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs in the transcriptional regulation network of escherichia coli. *Nature genetics*, 31(1):64–68, 2002.

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

[35] B. Viswanath, A. Mislove, M. Cha, and K. 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.

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

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

[38] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: On approximating graph edit distance. *PVLDB*, 2(1), 2009.

[39] Q. Zhao, Y. Tian, Q. He, N. Oliver, R. Jin, and W.-C. Lee. Communication motifs: a tool to characterize social communications. In *CIKM*, pages 1645–1648. ACM, 2010.

[40] F. Zhu, X. Yan, J. Han, and P. 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.

APPENDIX

A Weakness of the model proposed by Zhao et al [39]

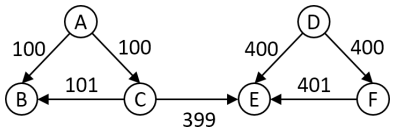


Figure 15: The scenario where two unrelated sets of interactions are clubbed together as related.

Zhao et al. [39] 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 17. 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) $|t_i - t_j| \leq \Delta T$.*

By the above definition, Fig. 15 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 .15 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.

B Violation of apriori property

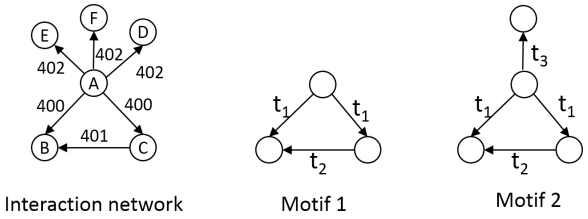


Figure 16: Violation of apriori property due to overlap.

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. 16. 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 $\{A, B, C\}$.

C Identifying temporally connected components

Algorithm 5 TCCDetect($N = (V, E)$, ΔT)

Input: An interaction network N, temporal threshold ΔT.
Output: Return all temporally connected networks in N at ΔT.
1: Mark all edges in E as not processed.
2: $TCC \leftarrow \emptyset$
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: $TCC \leftarrow TCC \cup G$
12: **return** TCC

D Computing support of a subsequence is NP-complete

THEOREM 5. *In the presence of identically overlapping instances, computing $sup(P) = |SeqDB(P)|$ is NP-complete.*

PROOF: [12] proves that when identically overlapping instances are allowed in the "traditional" definition of subsequence, the problem is NP-complete. Now, if sequence α is a "traditional" subsequence of β, then $\alpha \sqsubseteq \beta$ by Definition 8 as well. □.

E Pseudocode of the GetSup Algorithm

Alg. 6 presents the pseudocode of the GETSUP algorithm. We implement this following the algorithm proposed in [12].

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

Input: A sequence database $SeqDB = \{S_1, S_2, ..., S_N\}$, subsequence P, support set SS and edge e.
Output: A support set SS^+ of subsequence $P \circ e$,
1: **for each** $S_i \in SeqDB$ 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 each** $(i, < l_1, l_2, ..., l_{j-1} >) \in SS_i = I \cap S_i(P)$ 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** $SS^+ = \cup_{1 \leq i \leq N} SS_i^+$

F Returning to graph space

Algorithm 7 MotifMine ($SeqDB, SS$)

Input: A sequence database $SeqDB = \{S_1, S_2, ..., S_N\}$, support set SS.
Output: Motif with their frequencies
1: **for each** 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

With the formalization of the frequent subsequence mining framework under the altered definition of "subsequence" in Definition 8, the last remaining piece in the COMMIT algorithm is the procedure to return to graphs from the space of sequences. In this section, we design the MOTIFMINE algorithm in Alg. 7 to fill this gap.

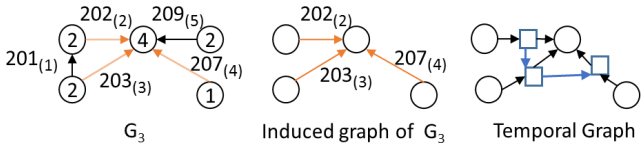


Figure 17: Instance $I = (3, < 2, 3, 4 >)$ of subsequence (1,3)(1,3)(1,3) corresponds to temporal component G_3 in Fig. 11. I represents an induced subgraph of G_3 (shown using the orange edges). For checking temporal isomorphism, induced graphs are converted into temporal graphs and the frequencies of temporal graphs are computed for final verification.

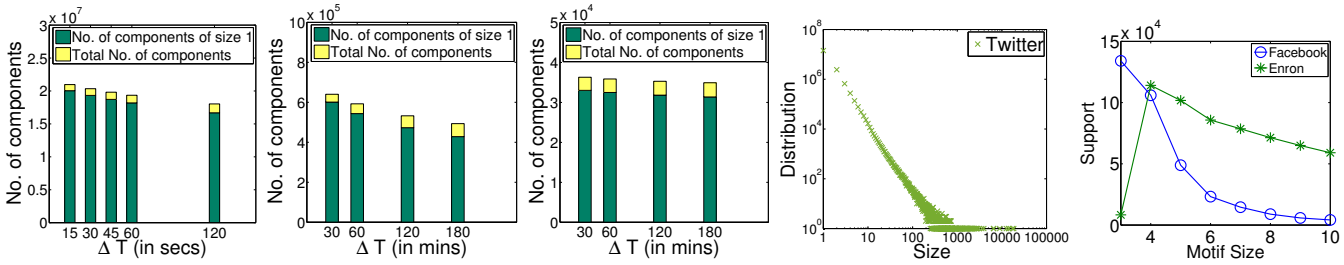
An instance $I = (i, < l_1, \cdots, l_n >)$ of a sequence stores sequence id i, which corresponds to the i^{th} component of the network. In addition, each l_j in I maps the location of the j^{th} edge in I to its location in component i. Recall that in Sec. 3 we devised a mechanism to impose a total 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. 7).

EXAMPLE 5. *Consider one instance $(3, < 2, 3, 4 >)$ of pattern $P_3 = (1, 3)(1, 3)(1, 3)$ from Fig. 11. The induced graph and its corresponding matched edges are shown in Fig. 17.*

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 4). 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. 17 illustrates the correspondence between an interaction graph and its temporal graph.

THEOREM 6. *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. □



(a) Twitter (b) Facebook (c) Enron (d) Twitter (e) Facebook and Enron

Figure 18: (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 (e) Distribution of communication motif sizes against their supports in Facebook and Enron datasets.

G Impact of temporally connected components on running time

Based on Theorem 6, the final temporal graph frequencies are computed using subgraph isomorphism tests and returned. In this section, we discuss how the properties of the temporally connected components affect the running time. Figs. 18(a)-18(c) show how the number of temporally connected components vary with ΔT. As can be seen, majority of the components contain just one interaction, 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. 18(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. 18(a)-18(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 is 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. 13(b)-13(d)), which indicates that larger components have more impact on the running time.

H Support and size distribution of communication motifs

Fig. 18(e) demonstrates the distribution of supports of communication motifs with respect to their sizes in the Facebook and Enron datasets. The behaviors are similar to that of Twitter (Fig. 13(j)). 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. 13(i), 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.

I Acknowledgments

The work was supported by Ericsson Research India PO 9201813513.