Flow Motifs in Interaction Networks

Chrysanthi Kosyfaki
Dept. of Computer Science and Engeeniring
University of Ioannina, Greece
xkosifaki@cs.uoi.gr

Nikos Mamoulis
Dept. of Computer Science and Engeeniring
University of Ioannina, Greece
nikos@cs.uoi.gr

Evaggelia Pitoura
Dept. of Computer Science and Engeeniring
University of Ioannina, Greece
pitoura@cs.uoi.gr

Panayiotis Tsaparas
Dept. of Computer Science and Engeeniring
University of Ioannina, Greece
tsap@cs.uoi.gr

ABSTRACT

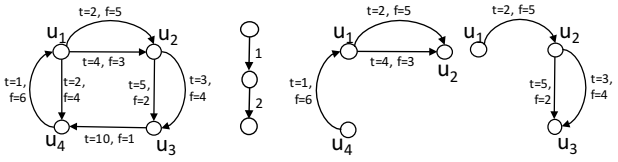
Many real-world phenomena are best represented as interaction networks with dynamic structures (e.g., transaction networks, social networks, traffic networks). Interaction networks capture flow of data which is transferred between their vertices along a timeline. Analyzing such networks is crucial toward comprehending processes in them. A typical analysis task is the finding of motifs, which are small subgraph patterns that repeat themselves in the network. In this paper, we introduce *network flow motifs*, a novel type of motifs that model significant flow transfer among a set of vertices within a constrained time window. We design an algorithm for identifying flow motif instances in a large graph. Our algorithm can be easily adapted to find the top-k instances of maximal flow. In addition, we design a dynamic programming module that finds the instance with the maximum flow. We evaluate the performance of the algorithm on three real datasets and identify flow motifs which are significant for these graphs. Our results show that our algorithm is scalable and that the real networks indeed include interesting motifs, which appear much more frequently than in randomly generated networks having similar characteristics.

1 INTRODUCTION

Interaction networks include a large number of highly connected components that dynamically exchange information. Examples of such graphs are neural networks, food webs, signal transfer pathways, the bitcoin network, social networks, and traffic networks. An interaction network captures *flow of data* (e.g., money, messages, passengers, etc.) which is transferred between its vertices along a timeline. In such a network, there could be multiple edges connecting the same pair of vertices, modeling data exchange between them at different times. Figure 1(a) shows a small example of an interaction network, where the vertices represent users who exchange money. The edges are annotated by timestamped interactions; e.g., edge u_1u_2 with label $t = 2, f = 5$ denotes that user u_1 sent 5 units of flow (money) to user u_2 at time 2.

Interaction networks are a powerful and versatile model, and as such they have been studied extensively in the literature [12, 23, 24]. In this paper, we consider the problem of finding small characteristic patterns in the networks, such as chains, triangles or cycles. These patterns are called *network motifs*. A motif is a subgraph that appears significantly more often in a real network than in a randomized network with similar characteristics [15]. Finding motifs is a method of identifying functional properties of a network. Previous work mainly focused on static motif patterns [15, 21]. Recently, there has been increasing interest in analyzing temporal networks [6, 8, 9, 17, 23, 24], where edges carry timestamps that signify the time of interaction between vertices. However, to the best of our knowledge, there is no previous work on motif search that considers the flow of data between connected nodes. Motivated by this, we define the concept of *flow motifs* in temporal interaction networks and study their identification.

Our definition of flow motifs extends a well-accepted definition of temporal motifs [17]. We define flow motifs as small graphs whose edges are ordered; the order defines how the data flows between the vertices. An instance of the motif is a subgraph of the interaction network, whose edges obey the total order specified by the edges of the motif. Moreover, the time difference between the temporally last and first edges should not exceed a pre-defined threshold δ which is a parameter of the motif. These requirements are the same as in the temporal motif definition of [17], which however disregards the data flow in interactions. The distinctive feature of our flow motifs is that, in a flow motif instance, multiple edges of the graph can instantiate a single edge of the motif, if they satisfy the order constraint with the edges that instantiate the motif's previous and next edges. The flow values in the edge-set that instantiates a motif edge are *aggregated* to a single value, which captures the *total flow* passing through the motif edge. The *minimum aggregated flow* at any motif edge defines the flow of the instance. In order for the instance to be valid, we require that its flow exceeds a threshold ϕ.



(a) multigraph (b) motif (c) instance 1 (d) instance 2

Figure 1: Example of graph, motif, and instances

Consider again the interaction network of Figure 1(a). Assuming that the motif of interest is a chain of three nodes (Figure 1(b)), where the labels in edges specify the flow order and that $\delta = 5$ and $\phi = 5$, the two subgraphs of Figures 1(c) and 1(d) are instances of the motif because the sets of edges mapped to each motif edge satisfy (i) the time order constraint of the motif and (ii) thresholds δ and ϕ. For example, in Figure 1(d), both edges that connect u_2 to u_3 are temporally after the edge that connects u_1 to u_2 and their aggregated flow is 6 ($\geq \phi$); in addition, the

time difference between the temporally first and last edges in the instance is $5 - 2 = 3$ ($\leq \delta$).

Overall, a valid flow motif instance should satisfy three requirements: (a) a *structural constraint*, defined by the graph structure of the motif; (b) a *temporal constraint* defined by the temporal window size δ; (c) a *flow constraint* defined by the minimum flow value ϕ.

Flow motifs correspond to frequently occurring sub-structures with high activity that appear in short time windows. Finding instances of flow motifs is of great importance in understanding interaction networks. For instance, in networks that model money transfers, flow motifs correspond to transaction patterns involving significant flow of money that appear more frequently than expected. Flow motif search is of particular interest to Financial intelligent units (FIUs); these are organizations which identify suspicious flow patterns that may suggest criminal behavior (e.g., money laundering). Belize FIU (fiubelize.org) and Hong Kong's JFIU (www.jfiu.gov.hk) indicate as suspicious patterns which include 'smurfing' (i.e., numerous small-volume transfers which aggregate to large amounts), cyclic transactions between parties, and chains of significant money transfers within limited time (e.g., payments out which are paid in on the same or on the previous day). In addition, bitcoin theft has been associated to flow patterns in [14]. In communication and social networks, flow motifs may reveal common patterns of influence [3, 11]. For example, the strength of the relationships between two social network users is correlated with the frequency of online interactions between them [22]. This implies that groups of users with frequent communication between them within a short period have high chance to influence each other.

Given a large interaction network, we propose an algorithm that takes as input a flow motif and efficiently finds its instances in the network. Our algorithm operates in two phases. First, the structural matches of the motif (disregarding temporal and flow information) are identified. Then, for each structural match, we find the motif instances which satisfy the temporal and flow constraints. This is achieved by sliding a time window of the same length as the duration constraint of the motif and systematically finding the combinations of edges that constitute motif instances. Compared to motif search algorithms from previous work, our algorithm is novel in that it considers the aggregated flow on multiple edges that connect the same pair of nodes in the network during the construction of the motif instances. Due to the large number of possible edge combinations, the problem is harder compared to finding instances of motifs, by disregarding flows and multiple edges. Our algorithm effectively uses the duration and flow constraints to prune the space. We also suggest a variant of the algorithm that identifies the top-k instances of an input flow motif with the highest flow. Finally, we propose a dynamic programming module for the algorithm, for the problem of finding the motif instance with the maximum flow.

We evaluate the performance of the algorithm on three real datasets of different nature (bitcoin user network, facebook network, and Passenger flow network). We compare the performance of our algorithm to a baseline method which builds up motif instances by joining their components and demonstrate the superiority of our approach against this alternative method. We also show that our tested flow motifs indeed appear more frequently in real networks than in randomized networks having the same characteristics as the real ones.

In summary, this paper makes the following contributions:

- We propose the novel concept of flow motif. To our knowledge, this is the first work that defines and studies the search of flow motifs in interaction networks.
- We propose an efficient algorithm for finding flow motif instances in large interaction networks and variants of it that identify the instances of a motif with the maximum flow.
- We evaluate our approach using three real datasets, and demonstrate that it scales well for large data.
- We investigate the significance of the tested motifs in the real networks.

The rest of the paper is organized as follows. Section 2 describes work related to network flow motifs, which are then formally defined in Section 3. Our motif search algorithm is presented in Section 4. Section 5 shows how to extend our algorithm to find the k instances of a given motif with the maximum flow. In Section 6, we experimentally evaluate our algorithm and the significance of the motifs by using a randomization approach. Finally, in Section 7, we conclude our paper and give directions for future work.

2 RELATED WORK

There has been a lot of research interest in motif search and mining in interaction networks [5, 20, 21]. In this section we summarize the most representative works in static and temporal networks.

Static Networks. Milo et al. [15] introduced the concept of motifs and studied their identification in large graphs. They defined a network motif as a *"pattern of interconnections occurring in complex networks at numbers that are significantly higher than those in randomized networks"*. They investigated motif discovery in directed networks, which do not carry temporal information (i.e., the motifs do not consider the time when the interactions took place).

FANMOD [21] is an efficient tool for finding network motifs in static networks, up to a size of eight vertices. Given a subgraph size, the tool either enumerates all subgraphs of that size or samples them uniformly. The identified subgraphs are grouped into classes based on their isomorphism. The significance of each class is finally measured by counting their frequencies in a number of random graphs (generated by swapping edges between vertices in the original network).

Temporal Networks. In *temporal networks*, the interactions between vertices are labeled by the time when they happen. Fundamental definitions, concepts, and problems on temporal networks are given in [6]. For instance, the concept of *time-respecting path* and its relation to network flows are defined and studied here.

Paranjape et al. [17] define motifs in temporal networks as small connected graphs, whose edges are temporally ordered. Instances of a motif are subgraphs that structurally match the motif and their edges obey the order. In addition, the time-difference between the temporally last and the first edges should not exceed a motif *duration* constraint δ. They propose a general algorithmic framework for computing the number of motif instances in a graph and fast algorithms that count certain classes of temporal motifs. Our network flow motifs are similar to the temporal motifs of [17], however, in our case (i) a motif edge can be instantiated by multiple edges of the graph and (ii) we introduce and consider a minimum flow requirement.

Another work that defines and studies the enumeration of temporal motifs is [8]. In the context of this work, the interactions between vertices are not instantaneous but they carry a duration interval. Motifs are again subgraphs whose edges are temporally ordered. As opposed to [17], there is no δ threshold between the last and the first edge in a motif instance. Instead, a maximum time-difference Δt between consecutive edges in a motif instance is allowed.

Rocha et al. [2] also define motifs that model the information spread in temporal networks. They study the impact of time ordering information by comparing the instances of the motifs by considering or not the temporal order. The flow motifs defined and used in [2] are different to ours, because in our case (i) we consider the flow on edges (ii) we define the flow in a motif differently and (iii) our input graph and the motif instances are multigraphs.

Communication motifs are suggested as a model for capturing the structure of human interaction in networks over time. Zhao et al. [23] studied the evolution of such behavioral patterns in social networks. For any two adjacent interactions, the term *maximum flow* is used to characterize those interactions that are the most probable to belong to the same information propagation path among any such adjacent interactions. On the other hand, in our context, flow refers to the data (e.g., money, messages, etc.) being transferred from one node along network paths. Another work that studies behavioral patterns in social networks by defining and mining communication motifs between people in social networks is [4]. A scalable mining technique (called COMMIT) for such communication motifs is proposed.

A recent work that studies the structure of social networks and the temporal relations between entities in them is [24]. Temporal pattern search is proposed as a tool in this direction. In order to facilitate the efficient retrieval of pattern instances, occurrences of small patterns are precomputed and indexed.

Flow can also be used to describe other concepts. In [9], the authors study the information propagation problem. They try to identify all time-respecting paths in temporal networks to model potential pathways for information spread. Our work is differnt in that (i) we are interested in specific motifs and (ii) we consider the flow on edges. The identification of time-respecting paths (as defined in [9]) that form cycles is studied in [10], where an efficient algorithm (2SCENT) is proposed.

Motif discovery in Heterogeneous Information Networks (HiNs) which carry temporal information was also recently studied [12]. In such graphs, some nodes are associated to events (which happened at a specific time). A motif is then defined by a graph and a maximum temporal difference between the events that instantiate its event nodes. As in the rest of previous work, any data flow on the edges of the network is disregarded in the definition and search of motifs.

3 DEFINITIONS

In this section, we formally define flow motifs and the graph wherein they are identified. Table 1 shows the notations used frequently in the paper.

The input to our problem is a directed multigraph $G(V, E)$, where each pair of nodes $u, v \in V$ can be connected by any number of edges in E. We denote by $E(u, v)$ the edge-set from $u \in V$ to $v \in V$. Each edge $e \in E$ is annotated by a unique *timestamp* $t(e)$ in a continuous time domain $\mathcal{T}$ and a positive real number $f(e)$, called *flow*.

Table 1: Table of notations

Notations	Description
$G_M(V_M, E_M)$	graph structure of motif M
δ	duration constraint of a motif
ϕ	flow constraint of a motif
$\ell(e)$	order of edge e in a motif M
SP_M	spanning path of motif M
e_i or $SP_M[i]$	i-th edge of motif M
$SP_M[i:j]$	subpath $e_i \ldots e_j$ of SP_M
$G(V, E)$	input graph
$E(u, v)$	set of edges in G from u to v
$f(e)$	flow on edge e
$t(e)$	timestamp of edge e
$f(G_I)$	flow of motif instance G_I
$G_T(V, E_T)$	time-series graph equivalent to $G(V, E)$
(t, f)	flow interaction element on an edge of E_T
$R(u, v)$	time series on edge $(u, v) \in E_T$
$R(e_i)$	time series on edge of E_T mapped to e_i
S	set of structural matches of a motif
G_s	structural match of a motif

Figure 2 shows an example of an input graph G from a real application, where vertices correspond to users (addresses) of the bitcoin network and edges correspond to transactions between them. Each edge is annotated by the timestamp of the transaction followed by the transaction amount. For example, user u_1 at timestamps 13 and 15 sent 5 and 7 bitcoins, respectively, to u_2.

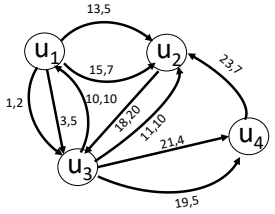


Figure 2: Example of an interaction graph (bitcoin user graph)

Definition 3.1 (Flow Motif). A network flow motif M is a triplet (G_M, δ, ϕ) consisting of (i) a directed graph $G_M(V_M, E_M)$ with $m = |E_M|$ edges, where each edge e is labeled by a unique number $\ell(e)$ in $[1, m]$; (ii) a value δ, which defines an upper-bound on the duration of the motif; and (iii) a value ϕ, which defines a lower bound on the flow of the motif.

The labels of the edges in the motif graph G_M define a total order of the edges that models the direction of the flow in G_M. For example, if G_M consists of two edges (u, v) and (v, w) and we have $\ell(u, v) = 1$ and $\ell(v, w) = 2$, this means that the flow in the graph originates from node u, it is first transferred to v, and then from v to w.

Figure 3 shows some examples of motifs (we only show the motif graphs G_M, but not the thresholds δ and ϕ). The numbers in the parentheses denote the number of nodes and edges in the motifs. For example, the motif labeled $M(3, 3)$ models a cyclic flow between three nodes.

We assume that the ordering of the edges according to their labels defines a *path* in the graph G_M. We refer to this path as the *spanning path* of the motif, and we denote it as SP_M. The spanning path is not necessarily a simple path, i.e., there may be repeated vertices in the path. We sometimes refer to a motif graph

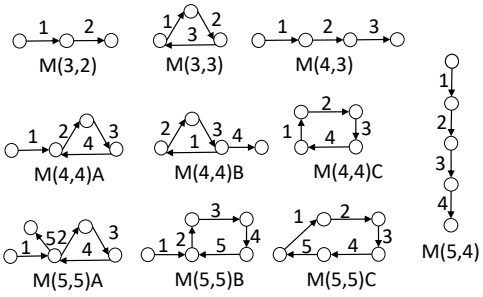


Figure 3: Examples of motifs.

G_M by its spanning path $SP_M = e_1 e_2 \dots e_m$, i.e., the total order of its edges, where e_i denotes the edge with label i. For example, we may refer to motif $M(3,3)$ in Figure 3 by the sequence $SP_{M(3,3)} = e_1 e_2 e_3$ of its three edges. In addition, we use e_i or $SP_M[i]$ to denote the i-th edge of the motif, and $SP_M[i:j]$ to denote the subsequence of edges $e_i \dots e_j$ along the path. We now define motif instances as follows.

Definition 3.2 (Flow Motif Instance). An instance of a motif $M = (G_M, \delta, \phi)$ in the graph $G(V,E)$ is a subgraph, $G_I(V_I, E_I)$, $V_I \subseteq V$, $E_I \subseteq E$ of G with the following properties:

- There is a bijection $\mu : V_M \rightarrow V_I$ from the vertex set of the motif graph V_M to the instance vertex set V_I.
- For every edge $(u,v) \in E_M$ there is a non-empty set of edges $E_I(\mu(u),\mu(v))$ in G_I, such that $E_I(\mu(u),\mu(v)) \subseteq E(\mu(u),\mu(v))$. In addition, $E_I = \bigcup_{(u,v)\in E_M} E_I(\mu(u),\mu(v))$.
- The edge-sets in G_I are *time-respecting*: For every pair of edges (u,v) and (v,w) in E_M, if $l(u,v) < l(v,w)$, then for every pair of edges $e_i \in E_I(\mu(u),\mu(v))$, $e_j \in E_I(\mu(v),\mu(w))$, $t(e_i) < t(e_j)$.
- The maximum time difference between any two edges in E_I is at most δ.
- The sum of flows of any edge-set in E_I is at least ϕ.

The first two conditions express a structural requirement on the matching subgraph, the third and fourth conditions temporal constraints, and the last condition a minimum flow constraint. Figure 4(a) shows an instance of $M(3,3)$ in the graph of Figure 2, assuming that $\delta = 10$ and $\phi = 7$. u_3, u_1, and u_2 are mapped to the first, second, and third node of $M(3,3)$ according to the order of its edges. u_1 and u_2 in the instance are linked by two edges which are both temporally after the edge(s) that link u_3 to u_1 and before the edge(s) that link u_2 to u_3. The maximum time difference between any two edges is 8 ($\leq \delta$) and the aggregate flows on $E_I(u_3,u_1)$, $E_I(u_1,u_2)$, and $E_I(u_2,u_3)$ are 10, 12, and 20, respectively (i.e., each of them is at least ϕ). If we denote $M(3,3)$ by its spanning path $SP_{M(3,3)} = e_1 e_2 e_3$, we can express the instance of Figure 4(a) by $[e_1 \leftarrow \{(10,10)\}, e_2 \leftarrow \{(13,5),(15,7)\}, e_1 \leftarrow \{(18,20)\}]$.

For the ease of exposition, we define the flow $f(G_I)$ of an instance G_I of motif M as the minimum total flow among all edge-sets $E_I(\mu(u),\mu(v))$ which instantiate the edges (u,v) of M. Formally:

$$f(G_I) = \min_{(u,v)\in E_M} \sum_{e\in E_I(\mu(u),\mu(v))} f(e) \quad (1)$$

We now define the concept of motif instance maximality.

Definition 3.3 (Instance Maximality). An instance $G_I(V_I, E_I)$ of a motif $M = (G_M, \delta, \phi)$ is maximal iff, the addition of one more edge to any edge-set $E_I(\mu(u),\mu(v))$ of G_I from the corresponding edge-set $E(\mu(u),\mu(v))$ of G violates the duration or flow constraints of the motif.

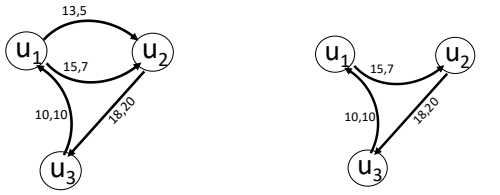


Figure 4: Examples of motif instances

For example, assuming that $\delta = 10$ and $\phi = 7$, Figure 4(b) shows an instance of $M(3,3)$ in the graph of Figure 2, which is not maximal. This is because the addition of edge (13,5) to $E_I(u_1,u_2)$ results in the valid instance of Figure 4(a). In this paper, we focus on finding maximal instances of motifs only, because non-maximal ones are redundant and considering them can mislead towards the importance of a motif. For example, if $\phi = 0$, all combinations of subsets of the edge-sets that form a valid motif instance are also valid (but not maximal) instances. Considering them would exponentially increase the total number of motif instances, potentially over-estimating its importance.

4 FINDING FLOW MOTIF INSTANCES

We now present an efficient algorithm for enumerating the maximal instances of a given motif $M(V_M, E_M)$ in an input graph $G(V,E)$. For the ease of presentation, we consider the input graph G not as a temporal multi-graph, but as a graph where all original edges from a vertex $u \in V$ to a vertex $v \in V$ are *merged* to a single edge. The single edge (u,v) is associated with an *interaction time-series* $R(u,v) = \{(t_1,f_1),(t_2,f_2),\dots\}$. Each pair (t_i,f_i) represents a *flow interaction* occurring at time t_i with flow transfer f_i from u to v. The interaction time series is ordered in time. Figure 5 shows an example of how the edges of a multigraph G are merged to time series. For example, the two edges from u_1 to u_2 are considered as a single edge; the two edges with timestamps 13 and 15 are now considered as a time series on a single edge (u_1,u_2). The conversion of the multigraph to a graph does not have to be explicitly performed; for each connected pair of vertices, it suffices to consider their multiple edges ordered by timestamp. We will use $G_T(V,E_T)$ to denote this graph and we will refer to it as the *time series graph*.

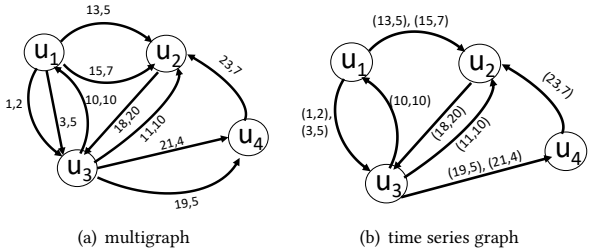


Figure 5: From a multigraph to a time series graph

Our algorithm takes as input the multigraph $G(V,E)$ and a motif $M = (G_M, \delta, \phi)$, and finds all instances of M in G. The

algorithm operates on the time series graph G_T and works in two phases P1 and P2:

- P1 Find the set S of all *structural matches* of graph G_M in graph G_T, disregarding the labels on the edges and constraints δ and ϕ.
- P2 For each $G_s \in S$, using the time series of the edges in G_s, find all instances of M in G_s (which should satisfy the duration and flow constraints defined by δ and ϕ).

We now elaborate on the two phases.

Phase P1: To illustrate phase P1, as an example, consider the graph G_T of Figure 5(b) and the motif $M(3,3)$ shown in Figure 3. Figure 6 shows all six structural matches of $M(3,3)$ in G_T found in phase P1. The labels $\{e_1, e_2, e_3\}$ on the edges of the matches indicate the edges of the motif on which they are mapped. For example, edge (u_1, u_2) of the first match is mapped to the first edge e_1 of the motif.

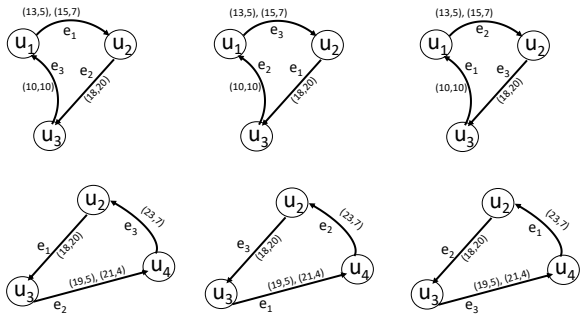


Figure 6: Structural matches of $M(3,3)$ (phase P1)

Algorithmically, for phase P1, any graph pattern matching algorithm for static graphs can be used (e.g., [21]). In our implementation, we exploit the fact that the ordering of the edges defines a path. Using a modified depth-first search algorithm on G_T, we can extract all paths of length $|E_M|$ that are structural matches of G_M in G_T. Specifically, in a loop, we map every node in G_T to the first node in G_M (i.e., the origin node of the first edge in G_M) and recursively find all paths that originate from that node and map them to the spanning path SP_M of G_M. For example, for motif $M(3,3)$, the depth-first search algorithm should make sure that the last vertex of the traversed path is the same as the first vertex of the path. Hence, the algorithm on the graph G of our running example would identify path $u_1u_2u_3u_1$ as a match of $M(3,3)$.

Phase P2: In phase P2, given the set of structural matches S, for each $G_s \in S$, we process the time series on the edges of G_s in order to find valid flow motif instances. In a nutshell, we slide a time window of length δ along the set of all (t_i, f_i) interactions on the edges of G_s; for all sets of interactions within δ time difference, we find all combinations thereof which constitute valid motif instances. Note that each structural match G_s from phase P1 may produce an arbitrary number of flow motif instances, as each time window position can generate different instances depending on the combinations of edge flows we use.

To illustrate, consider again $M(3,3)$ (for $\delta = 10$) and a possible structural match, shown in Figure 7. We will get different flow motif instances depending on whether we consider window $[10, 20]$ or $[15, 25]$. Furthermore, even for the specific time-window $[10, 20]$, we can get different flow motif instances depending on how we combine the edges in this window. For example, one possible flow motif instance is $[e_1 \leftarrow \{(10,5)\}, e_2 \leftarrow \{(11,3), (16,3)\}, e_3 \leftarrow \{(19,6)\}]$, while another flow motif instance is $[e_1 \leftarrow \{(10,5)\}, e_2 \leftarrow \{(11,3)\}, e_3 \leftarrow \{(14,4), (19,6)\}]$. Note that the flow in the former case is 5, while in the latter is 3, meaning that the latter instance would be rejected for $\phi = 5$.

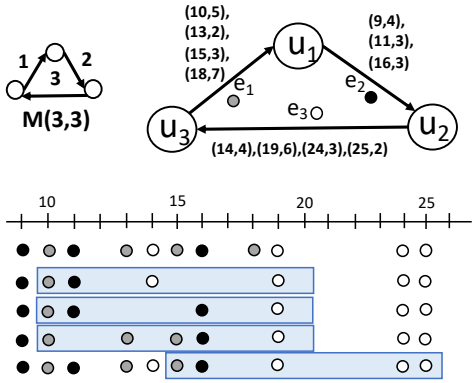


Figure 7: Example for Algorithm 1

Algorithm 1 is applied in phase P2 to find all instances of the motif M in a match G_s (found in phase P1). The algorithm slides a window T of length δ over the time domain, to find subsets of edges in G_s that satisfy the duration constraint δ and can generate maximal motif instances. Given a specific window T we run procedure FindInstances in order to generate all possible maximal flow-motif instances that satisfy the flow constraint ϕ. The procedure is recursive on the length m of the spanning path $SP_M = e_1e_2 \ldots e_m$ of the motif.

FindInstances takes as input the graph instance G_s, a spanning path SP, a time-window T and the threshold ϕ. Let $R(e_i)$ be the interaction time series on the edge of G_s which is mapped to edge e_i of the motif. If the spanning path consists of a single edge e_1, then the procedure finds the set $R_T(e_1) \subseteq R(e_1)$ of all elements in $R(e_1)$, which are within the time-window T, and aggregates their flow. If the total flow $f(R_T(e_i))$ of these elements satisfies the flow constraint ϕ, the edge-set of G corresponding to $R_T(e_i)$ becomes an instance of SP and it is returned. For longer spanning paths, the procedure considers again the first edge $e_1 = SP[1]$. For every prefix T_p of the window T that contains instances of the edge e_1, it computes the set $R_{T_p}(e_1) \subseteq R(e_1)$ of all (t, f) interaction elements in $R(e_1)$ for which $t \in T_p$. If $R_{T_p}(e_1)$ is non-empty and satisfies the flow constraint, then FindInstances is recursively called on the rest of the spanning path $SP_{next} = SP[2:m]$, with time window $T_{next} = T - T_p$. This recursive call will return the set of valid instances within time-window T_{next} for the sub-motif defined by SP_{next}. Each of these instances is *concatenated* to $R_{T_p}(e_1)$ to create a new valid instance for SP.

The condition at line 16 of the algorithm helps us to find invalid prefixes of the motif instances early. In other words, if a sub-series $R_{T_p}(e_1)$ which is candidate for instantiating a motif edge does not qualify ϕ, we do not consider the possible instances that include the elements of $R_{T_p}(e_1)$ as an instance of e_1. Hence, the search space is effectively pruned.

Figure 7 illustrates the functionality of Algorithm 1. On top, the figure shows motif $M(3,3)$ and a structural match G_s of it, where each edge is labeled by the time series of flows between the corresponding nodes (e.g., at time 10, u_2 sent to u_1 a flow of 5). The elements on the edges of G_s are illustrated (as sequences of dots ordered by time) at the bottom of the figure, colored by

Algorithm 1 Instance finding module

Require: δ, ϕ, time window T, structural match G_s

$\mathcal{I} \leftarrow \emptyset$ ▷ set of instances of G_s in T
for each maximal time window T that satisfies δ **do**
  $\mathcal{I} \leftarrow \mathcal{I} \cup$ FindInstances(G_s, SP_M, T, ϕ)
end for
return $\mathcal{I}$

procedure FindInstances(G_s, SP, T, ϕ)
  $\mathcal{I} \leftarrow \emptyset$ ▷ set of instances of G_s in T
  if $length(SP) = 1$ **then**
    $R_T(e_1) \leftarrow$ all (t, f) elements of $R(e_1)$ in T
    if $f(R_T(e_1)) \geq \phi$ **then** ▷ ϕ condition check
      add $R_T(e_1)$ to $\mathcal{I}$
    end if
  else
    for each prefix T_p of time window T **do**
      $R_{T_p}(e_1) \leftarrow$ all (t, f) elements of $R(e_1)$ in T_p
      if $f(R_{T_p}(e_1)) \geq \phi$ **then** ▷ ϕ condition check
        $SP_{next} \leftarrow SP[2:m]$ ▷ suffix of SP
        $T_{next} \leftarrow T - T_p$ ▷ suffix of T
        $\mathcal{I}_{next} \leftarrow$ FindInstances(G_s, SP_{next}, T_{next}, ϕ)
        for each $I \in \mathcal{I}_{next}$ **do**
          add $R_{T_p}(e_1) \circ I$ to $\mathcal{I}$
        end for
      end if
    end for
  end if
  return $\mathcal{I}$
end procedure

the edge they belong to (e.g., black for e_2). The first row of dots includes all (t, f) elements, i.e., the first black dot corresponds to element $(9, 4)$ on edge (u_1, u_2), which is mapped to the second edge e_2 of $M(3, 3)$. To find the motif instances that comprise of nodes and edges in G_s, we slide a window of length δ along the timeline. Assuming that $\delta = 10$, the first position of the sliding window is $[10, 20]$. The algorithm finds all prefixes of elements in $R(e_1)$ that fall in this window and for each such prefix, it generates recursively the combinations of elements from other edges that form valid instances (according to δ). For example, for the prefix $T_p = [10, 10]$, which includes just the first element $(10, 5)$ from e_1, the 2nd and the 3rd line of dots in the figure show the valid instances formed. Specifically, these instances are $[e_1 \leftarrow \{(10, 5)\}, e_2 \leftarrow \{(11, 3)\}, e_3 \leftarrow \{(14, 4), (19, 6)\}]$ and $[e_1 \leftarrow \{(10, 5)\}, e_2 \leftarrow \{(11, 3), (16, 3)\}, e_3 \leftarrow \{(19, 6)\}]$. Note that the ϕ constraint is applied at every prefix in order to prune the search space if it is violated (e.g., if $\phi = 5$, any instance $[e_1 \leftarrow \{(10, 5)\}, e_2 \leftarrow \{(11, 3)\}, \dots]$ would be rejected. Note also that there is no instance which contains just the first two elements of e_1 but not the third one, because there is no element from e_2 which is temporally between $(13, 2)$ and $(15, 3)$. Finally, note that the next position of the sliding window is $[15, 25]$ because the position $[13, 23]$ which starts from the 2nd element of e_1 does not include any new elements from e_3 compared to the previous window position $[10, 20]$; hence, considering window position $[13, 23]$ would result in redundant (i.e., non-maximal) instances and this position is skipped.

We have not explained yet how window positions are skipped in Algorithm 1. First, only window positions which start at elements of $R(e_1)$ are considered; in-between positions (e.g., window $[11, 21]$ in Figure 7) would result in redundant (non-maximal) instances because there will be a subsequent position for which $R(e_1)$ (and the other sets) can only expand (e.g., window $[13, 23]$ in Figure 7). Second, from those window positions that are considered, we skip those, where $R(e_m)$ (i.e., the interaction time series, which is mapped to the last edge e_m of the motif) is not expanded with new elements, compared to the previous valid window position. In our example, $[13, 23]$ is skipped because no element is added to $R(e_3)$, compared to position $[10, 20]$. If we used window position $[13, 23]$, we would generate instances that would not be maximal because we could add to each of them element $(10, 5)$ of e_1 without violating the δ constraint. In summary, in consecutive window positions where module FindInstances is applied, the first elements of $R(e_1)$ should be different and the last elements of $R(e_m)$ should also be different.

Algorithm 1 does not miss any maximal instances because it systematically explores the combinations of edge-sets which are time-respecting and maximal within a window. Moreover, the windows have maximal lengths and in each of them the produced instances essentially include the temporally first (t_i, f_i) element that maps to e_1 and the temporally last (t_i, f_i) element that maps to e_m. At least one of these pairs changes in the next window position; therefore, instances produced at different windows do not violate the maximality condition.

Complexity Analysis. In the worst case, for each G_s and each time window, we should consider all combinations of edges in G that instantiate the edges of the motif. For example, when $\phi = 0$, prefix-based pruning cannot be applied. In the worst case, $G_s = G$ and the edges in G ordered by timestamp are assigned to the sequence of motif edges in a round-robin fashion. That is, the temporally first edge of G is mapped to e_1, the second to e_2, etc. In this case, assuming the loosest possible constraints $\delta = \infty, \phi = 0$, the number of combinations of pairs to be considered (which all form valid motif instances) is $O(|E|/m)^m$, i.e., exponential to the number of edges m in the motif. In addition, the number of structural matches is also exponential to m. In practice, G_T is sparse (or V is small) and the constraints δ and ϕ help in pruning combinations of edges that do not form instances early, which renders the algorithm scalable, as we will show in the experimental evaluation.

5 TOP-K FLOW MOTIF SEARCH

Setting an appropriate value for the parameters δ and ϕ could be hard for non-experts of the domain. Parameter δ is intuitively easier to be set to a time constraint that makes sense to the application (for example, the analyst could be interested in patterns of bitcoin transactions which happen within an hour or day). On the other hand, ϕ is less intuitive, as too large values could result in too few or zero instances, whereas too small values could result in thousands of instances which may overwhelm the user. One solution to this problem is to replace the ϕ constraint by a ranking of the motif instances G_I with respect to their flow (see Equation 1). In other words, we may opt to search for the k instances G_I of the motif (with $\phi = 0$) that satisfy δ, which have the maximum flow $f(G_I)$.

To solve this top-k flow motif search problem, we can use our algorithm with a small number of modifications. Phase P1 is identical; we should still find the set S of all structural matches. Then, for each $G_s \in S$, we apply phase P2, by making the following changes to Algorithm 1. First, we keep track in a priority queue (heap) the top-k instances in terms of their minimum flow so far. Second, in place of ϕ, we use the flow $f(G_I^k)$ of the k-th instance G_I^k so far as a dynamic (floating) threshold.

5.1 Finding the top motif instance

For the special case, where k=1, the top-1 motif instance search problem can potentially be solved faster with the help of a dynamic programming (DP) algorithmic module. Recall that the objective of procedure FindInstances in Algorithm 1 is to find the motif instances in a structural match G_s, within a time window T, which qualify ϕ. We can replace this module by a dynamic programming algorithm that finds the instance of maximum flow within T. This DP module can be described by Algorithm 2.

Algorithm 2 DP module for top-1 instance search

Require: δ, time window T, structural match G_s

```
maxflow ← 0        ▷ keeps track of max flow found at any instance
for each maximal time window T that satisfies δ do
    for all timestamps t_i in T do
        compute Flow([t_1, t_i], 1) = flow([t_1, t_i], 1)
    end for
    for κ = 2 to n do
        for all timestamps t_i in T do
            compute Flow([t_1, t_i], κ) by Eq. 2
        end for
    end for
    maxflow = max(maxflow, Flow([t_1, t_τ, m]))
end for
return maxflow
```

Specifically, let $[t_1, t_2, \ldots, t_\tau]$ be the sequence of timestamps in T for which there is a (t, f) interaction element in G_s. Let M_κ be the prefix of M which includes its fist κ edges only and $Flow([t_1, t_i], \kappa)$ be the flow of the top-1 motif instance of M_κ in the time window $[t_1, t_i]$. Then, $Flow([t_1, t_i], \kappa)$ can be recursively computed as follows:

$$Flow([t_1, t_i], \kappa) = \max_{1<j\leq i}\{\min(Flow([t_1, t_{j-1}], \kappa-1), flow([t_j, t_i], \kappa))\}, \quad (2)$$

where $flow([t_j, t_i], \kappa)$ is the total flow of all (t, f) elements of the time series $R(e_\kappa)$ on the κ-th edge of G_s, whose timestamps are in the time interval $[t_j, t_i]$. The $Flow([t_1, t_i], 1)$ array is initialized by scanning the elements of the first edge of G_s in T. Then, for each $\kappa > 1$, $Flow([t_1, t_i], \kappa)$ is computed using array $Flow([t_1, t_i], \kappa-1)$. Finally, $Flow([t_1, t_\tau], m)$ corresponds to the top-1 flow of any motif instance in G_s within time window T. By applying this algorithm for every window T, we can find the top instance in G_s. Repeating this for each G_s gives us the top-1 instance of M in G.

Table 2 shows the steps of the DP module in the course of finding the top-1 instance in time window $[10, 20]$ (assuming that δ=10) for the structural match of $M(3, 3)$ shown in Figure 7. The first row shows the values of $Flow([t_1, t_i], 1)$ for the first edge of the motif and for all values of t_i (i.e., columns of the table). (Recall that the starting timestamp t_1 of the time window is 10.) The second row shows, for the first two edges of the motif, the value of $Flow([t_1, t_i], 2)$ for all values of t_i, as well as the value of t_j, which determines $Flow([t_1, t_i], 2)$. For all t_i, the value of t_j that maximizes the flow is 11 and for $t_i \geq 16$ the flow becomes $min(5, 3 + 3) = 5$. Finally, the last row shows the maximum flow for the best arrangement of (t, f) pairs to all three edges of the motif, for all prefixes of the time window. Note that the last value corresponds to the entire window and contains the flow of the best instance of the entire motif in $[10, 20]$, which is 5. The cells of the matrix in bold show how the top-1 instance, i.e., $[e_1 \leftarrow \{(10, 5)\}, e_2 \leftarrow \{(11, 3), (16, 3)\}, e_3 \leftarrow \{(19, 6)\}]$, can be identified.

Table 2: Example of the DP module

t_i	10	11	13	14	15	16	18	19
κ=1	**5**	5	7	7	7	7	10	10
κ=2		3 (t_j=11)	3 (t_j=11)	3 (t_j=11)	3 (t_j=11)	5 (t_j=11)	**5 (t_j=11)**	5 (t_j=11)
κ=3			0 (t_j=13)	4 (t_j=14)	4 (t_j=14)	4 (t_j=14)	4 (t_j=14)	**5 (t_j=19)**

Complexity Analysis. For each G_s and each time window, we should consider all binary splits of the window at each iteration (i.e., for each edge in M). Hence the time complexity is $O(\tau^2|E|)$, where τ is the number of timestamps in T for which there is an (t_i, f_i) element in G_s. The space complexity is $O(\tau \cdot |E|)$ because we only need all $Flow([t_1, t_i], \kappa-1)$ for $\kappa-1$ when we process the κ-th edge. The overall time complexity per structural match in S is $O(|S|\delta\tau^2|E|)$, since the number of windows to be considered is $O(\delta)$. The number of structural matches $|S|$ is exponential to m, as discussed in our previous analysis.

Extensibility. The DP module can be used to solve top-1 problems at a finer granularity. In particular, it can be used to find the top-1 instance for each structural match G_s. This may be desirable if we want to compare the sets of entities that constitute the structural instances (e.g., groups of bitcoin users) based on their max-flow interactions. In addition, we might be interested in finding the top-1 instance for each position of the sliding time window T. This can be part of analysis tasks that compare the volume of interactions (according to the motif structure) at different periods of time.

6 EXPERIMENTAL EVALUATION

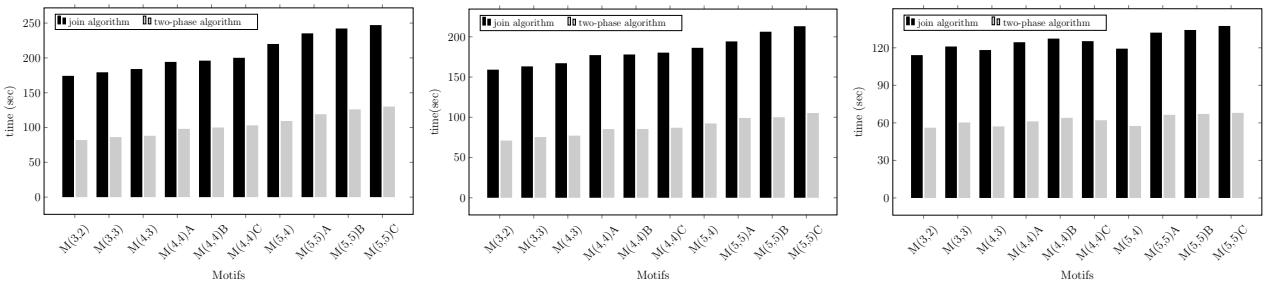
The goal of our experimental evaluation is twofold: test the performance and scalability of our algorithms and study the significance of flow motifs. We implemented the algorithm presented in Section 4 and its two variants proposed in Section 5 (top-k instance search, dynamic programming module for top-1 search). As a baseline, we also implemented an alternative motif instance finding method based on finding and joining instances of motif components in a hierarchical manner.

We evaluate the performance of all these methods on three real networks, to be described in Section 6.1. We measure the efficiency and scalability of the tested methods as a function of the problem parameters δ and ϕ on the motif structures shown in Figure 3. These graphs model representative flows of interaction that could be of interest to data analysts (e.g., $M(3, 3)$ corresponds to cyclic transactions in a money-exchange network, $M(4, 3)$ corresponds to paths of region-to-region movements in a passenger flow network). We also assess the statistical significance of the tested motifs in three real graphs. All algorithms were implemented in Python3 and we ran all the experiments on a machine with an Intel Xeon CPU E5-2620 prossesor running Ubuntu 18.04.1 LTS.

6.1 Dataset Description

We used three datasets extracted from real interaction networks: the **Bitcoin** network, the **Facebook** network and a **Passenger** flow network. Table 3 shows statistics of the datasets. The third column is the distinct number of node pairs $(u, v) \in V$, for which there is at least one edge (i.e., interaction) from u to v. This number equals to the number $|E_T|$ of edges in the corresponding time-series graph G_T. We now provide more details about them.

Bitcoin network. We downloaded all transactions in the bitcoin blockchain [16] in the period February 1st 2014 to November



(a) Bitcoin Network (b) Facebook Network (c) Passenger Network

Figure 8: Our two-phase algorithm vs. the join algorithm

Table 3: Statistics of Datasets

Dataset	#nodes	#connected node pairs	#edges	Avg. flow per edge
Bitcoin	24.6M	88.9M	123M	4.845
Facebook	45800	264000	856000	3.014
Passenger	289	77896	215175	1.933

30 2014 and converted them to a bitcoin *user graph*.[1] Nodes correspond to users and for each transaction of f bitcoins in the blockchain from user u to user v at time t, we added an edge from u to v with label (t, f). Since the same bitcoin user may control and use multiple addresses, we applied a well-known heuristic [1, 7] to *merge* addresses that are considered to belong to the same user to a single network node. Specifically, we merged addresses that appear together as input in the same transaction. We did not take into account insignificant transactions with amounts under 0.0001 BTC. Bitcoin is a relatively sparse graph and the cases of two nodes being connected by multiple edges is rare. Finding motif instances in the Bitcoin network can help towards understanding complex interactions between users and can possibly help toward identifying suspicious transactions like money laundering and bitcoin theft [14].

Facebook network: We consider Facebook as an interaction network between users. We divide the time into 30-second intervals $[t_s, t_e)$ and for each pair of users u and v we aggregate all interactions from u to v and add an edge from u to v with label (t_s, f), where f is the total number of interactions from u to v in this interval. We consider as interactions the posts of likes by u targeting v or the messages sent from u to v. We created the Facebook user network using data from April 2015 to October 2015; the same dataset is used in [19]. The Facebook network is relatively sparse and each pair of connected nodes have about four edges on average. Motif search on this graph can help in analyzing influence [3, 11] and finding important interactions among users [13].

Passenger flow network: We processed trips of yellow taxis in NYC in January 2018.[2]. Each record includes the pick-up and drop-off taxi zones (regions) the date/time of the pick-up and drop-off, and the number of passengers inside the taxi. Using these records, we created an interaction network where the nodes are the taxi zones; for each record, we generate an edge that links the corresponding nodes and carries the timestamp of the activity (i.e., the pickup time) and the corresponding flow (i.e., the number of passengers). This Passenger flow network is dense; in addition, each pair of connected nodes have about three edges on average. Motif instances found in this passenger flow graph can help in understanding the flow of movement between different regions.

[1] data obtained from http://www.vo.elte.hu/bitcoin
[2] obtained from http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

6.2 Efficiency and Scalability

In this section, we evaluate the efficiency and scalability of our algorithm when applied to find the instances of the motifs depicted in Figure 3. The default values for the duration constraint δ are 600 sec., 600 sec., and 900 sec. on Bitcoin, Facebook, and Passenger, respectively. These value represent realistic time intervals for the corresponding applications. The corresponding default values for ϕ are 5, 3, and 2, respectively.

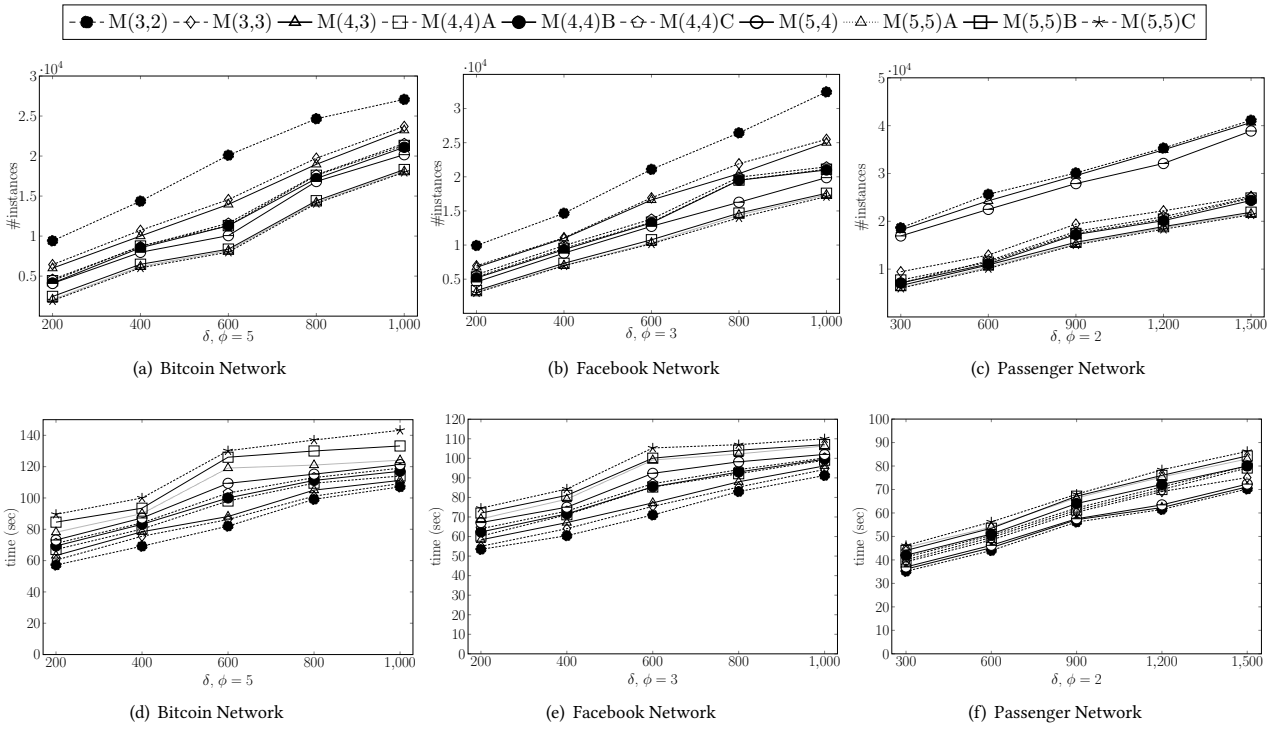
6.2.1 Comparison to a competitor. In our first set of experiments, we compare our algorithm with an alternative motif instance finding algorithm which is based on progressively finding and joining instances of motif subgraphs.

Specifically, this *join algorithm* starts by accessing each edge (u, v) of the time series graph G_T and finding all time-intervals of length at most δ and their aggregated flows. For each such interval $[t_s, t_e]$ a quintuple (u, v, t_s, t_e, f) is generated. These tuples are kept in two tables; C_1 sorts them by starting vertex u and C_2 sorts them by ending vertex v. In the next step, C_2 and C_1 are merge-joined to find all pairs (c_2, c_1) having $c_2.u = c_1.v$ and also satisfying $c_1.t_e - c_2.t_s \leq \delta$. The set P of all these tuple pairs constitute results of all sub-motifs of M which include two consecutive edges. In the next step, P is self-joined again to produce instances of sub-motifs of M with three consecutive edges. This is done by finding pairs $\{(c_2, c_1), (c'_2, c'_1)\}$ of couples in P for which $c_1 = c'_2$ and $c'_1.t_e - c_2.t_s \leq \delta$. The next steps are applied in a similar manner until the instances of the entire motif M are constructed. Note that for each motif or sub-motif that closes a cycle (e.g., $M(3, 3)$), we check the additional condition that the starting vertex of the first motif edge in the instance is the same as the target vertex of the last edge. At each step, we apply a merge join for the production of sub-motif instances, after having sorted the tuples produced in the previous step accordingly.

Figure 8 compares the runtime cost of the join algorithm with that of our two-phase algorithm presented in Section 4. For all motifs, we used the default values for δ and ϕ. Our two-phase algorithm is typically twice as fast as the join algorithm. This is due to the fact that the join algorithm produces a large number of intermediate results (i.e., sub-motif instances), which are avoided by our method. Many of these sub-motif instances do not end up as components of any instance of the complete motif, so their generation is redundant. In the rest of this section, we do not include additional comparisons with the join algorithm since it was always found to be slower than our approach.

Table 4: Number of structural matches and runtime in phase P1 of motif search

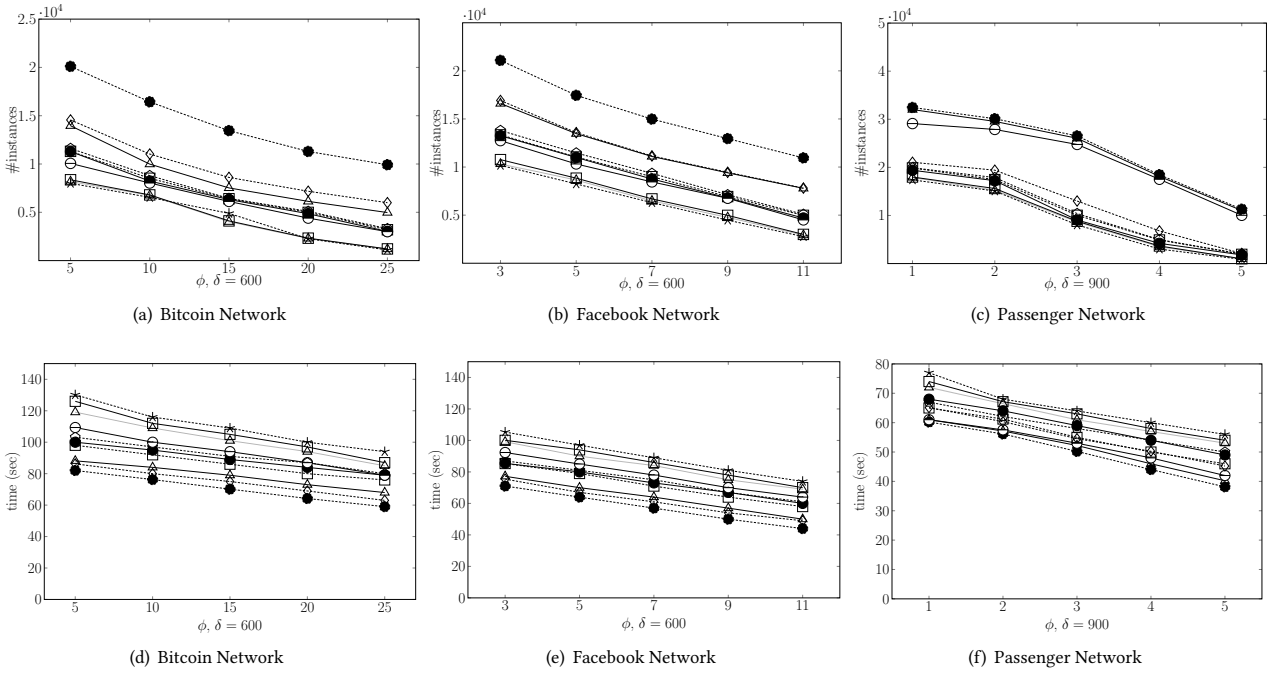
	Motif	M(3,2)	M(3,3)	M(4,3)	M(4,4)A	M(4,4)B	M(4,4)C	M(5,4)	M(5,5)A	M(5,5)B	M(5,5)C
Bitcoin	Instances	634K	485K	484K	210K	205K	213K	145K	122K	124K	121K
	Time (sec)	47.02	49.23	50.15	57.05	60	61.16	64.35	69.11	73.02	75.15
Facebook	Instances	415K	276K	272K	113K	113K	114K	97K	90K	91K	90K
	Time(sec)	40.02	43.43	44.21	48.45	49.32	49.01	52.33	50.12	52.07	54.31
Passenger	Instances	27893	16455	25778	14877	14569	14903	22134	12345	12567	12009
	Time(sec)	19.14	21.33	22.15	26.22	29.03	29.11	25.04	30.45	31.14	32



(a) Bitcoin Network (b) Facebook Network (c) Passenger Network

(d) Bitcoin Network (e) Facebook Network (f) Passenger Network

Figure 9: Number of instances and time for different values of δ



(a) Bitcoin Network (b) Facebook Network (c) Passenger Network

(d) Bitcoin Network (e) Facebook Network (f) Passenger Network

Figure 10: Number of instances and time for different values of ϕ

6.2.2 Sensitivity to δ and ϕ. The next set of experiments evaluate the performance of our algorithm on the different datasets and motifs, for various values of the constraints δ and ϕ. Table 4 shows the number of structural matches found and the time spent by the algorithm just for its first phase, which is independent of the δ and ϕ values (since these constraints are not used when searching for the structural matches). This cost constitutes a lower bound for our algorithm. Naturally, more complex motifs require more time but they also have fewer structural matches.

Figures 9 and 10 show the number of instances and total runtime of our algorithm for different values of δ (in seconds) and ϕ. When we vary δ, we set ϕ to its default value and vice versa. As expected, in all cases, when δ increases the number of instances and the runtime increases. The algorithm scales well as its cost increases at a lower pace compared to the results found.

When comparing the different motifs, note that the simpler ones (e.g., $M(3,2)$ and $M(3,3)$) naturally have more instances and are cheaper to search compared to the more complex ones (e.g., $M(5,5)A$). The relative order between the motifs is similar in the Bitcoin and Facebook networks. In both networks, cyclic flow is quite common; i.e., motifs containing cycles have a similar number of instances as motifs without cycles having the same number of edges. On the other hand, in the Passenger network, acyclic motifs dominate in terms of number of instances. This is expected, as it is relatively rare that passengers move between regions on a map forming cycles compared to moving along a chain of different regions.

The behavior is also consistent to our expectation when ϕ varies; the number of instances and the runtime drop when ϕ increases. The algorithm becomes faster because partial motif instances that do not qualify ϕ are pruned early.

6.2.3 Top-k flow motif instance search. We now evaluate the results and the performance of top-k motif search on the three datasets, when using the default values of δ. In the first experiment, we run the version of our algorithm which finds the top-k motif instances that have the maximum flow. For each run, we record the flow of the k-th instance in Figure 11. As expected, the flow of the k-th instance drops as k increases; the drop rate decreases when k becomes large (note that the x-axis is not linear). In the second experiment, we compare the runtime of the general top-k algorithm with its version that employs the dynamic programming module proposed in Section 5.1. The barcharts show that the second phase of the algorithm benefits from the use of dynamic programming (the runtime drops 20% to 40%). The improvement is better on the Passenger network.

6.2.4 Scalability to the dataset size. In the next experiment, we test the performance of our algorithm on samples of the original datasets having different sizes. For each of the three datasets, we take samples defined by prefixes of the total period covered by the timestamps of the edges included in the sample. Specifically, for the Bitcoin network we define 5 samples: B1, B2, B3, B4, B5. B1 includes all transactions happened in the first month of the 9-month period of the complete dataset. B2, B3, B4, and B5 cover the first 2, 4, 6, and 9 months, respectively. Similarly, F1, F2, F3, F4, and F5 cover the first 1, 2, 3, 4, and 6 months of the entire dataset, respectively. Lastly, T1, T2, T3, and T4 cover the first 8, 16, 24, and 31 days of January 2018, respectively. Figure 13 shows the growth in the number of instances and in the runtime of the algorithm for the different motifs. Observe that the algorithm scales well as its cost grows at a slower pace compared to the number of instances and the size of the input data.

6.3 Significance of Motifs

In the last experiment, we assess the significance of the different flow motifs in our networks. Following the standard practice [18], we generated randomized versions of our datasets, we computed the number of instances of each motif in each of these datasets, and we compared it against the same number for the real dataset. A large divergence between real and randomized numbers indicates a significant motif.

Specifically, from each dataset (e.g.. Bitcoin network) we generated random datasets by keeping the structure of the corresponding graph fixed, and permuting the flows on the edges. Recall that in the original input multigraph $G = (V, E)$ each edge e is associated with a timestamp $t(e)$ and a flow value $f(e)$. A pair of nodes (u, v) is connected by a set of edges $E(u, v)$. Given the entire set of flow values $\{f(e) : e \in E\}$, we compute a random permutation π of the flow values and reassign them to the graph edges in this order. This generates a randomized dataset $G_r(V, E)$ with the same set of nodes and the same set of edges; each edge e has the same timestamp $t(e)$, and flow value $\pi(f(e))$. Hence, G_r is derived from G by "shuffling" the flow values on the edges.

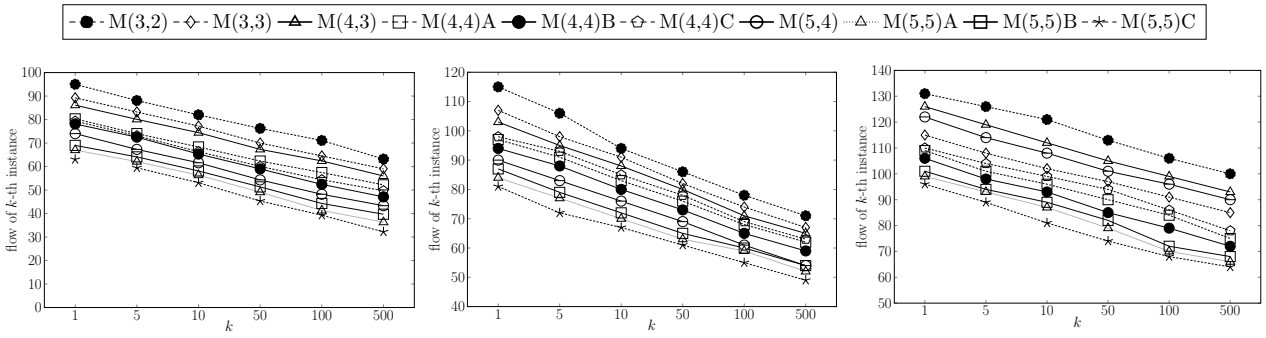
The random graph G_r has the same structure as G and the edges in the graph appear at the same timestamps. Therefore, all structural matches of the motifs in G will also appear in G_r. In addition, putting aside the flow constraint ϕ, the motif instances in the two graphs will be the same, when considering only δ. What changes is the flow value of each motif instance, which will result in a different number of flow motif instances in G_r compared to G, for non-zero values of ϕ. Our goal is to see whether the motif instances that satisfy the ϕ constraint in the real data are significantly more than those in the randomized data.

We generated 20 different random graphs for each real network according to the procedure we described above. We found the instances of each motif in all these random datasets. In addition, we computed the mean and standard deviation of the number of motif instances in all 20 random graphs per real dataset. To assess the significance of a motif in the real data, we compared the number of instances in the real data with those in the random data. Figure 14 shows, for each dataset and motif, the distribution of the numbers of instances for all random graphs in a box plot, and the corresponding number in the real graph (marked by a diamond). Each real value is also associated with the *z-score* (shown above the corresponding diamond), which is computed as follows. For some motif M, let r_M denote the number of instances of the motif in the real data, let μ_M denote the mean number of motif instances in the randomized data, and let σ_M denote the standard deviation. The z-score z_M of the motif is computed as

$$z_M = \frac{r_M - \mu_M}{\sigma_M}$$

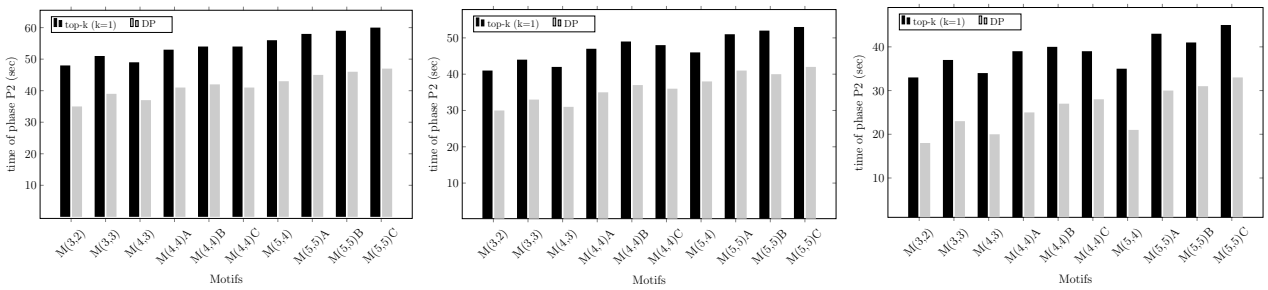
The higher the z-score, the further the value r_M from μ_M.

The first observation is that the number of instances in all random graphs is much lower compared to that in the corresponding real network and these values do not deviate much from their mean. The empirical p-value (the fraction of random datasets with number of instances greater than that of the real data) is zero, indicating statistical significance of the motif occurrences in all cases. This is consistent with the intuition that the flow is not arbitrarily generated or consumed at the vertices of the network, but it is transferred from one node to another. To discriminate between the different motifs we look at the z-scores. We observe that for the Bitcoin network, two out of the three top z-scores are



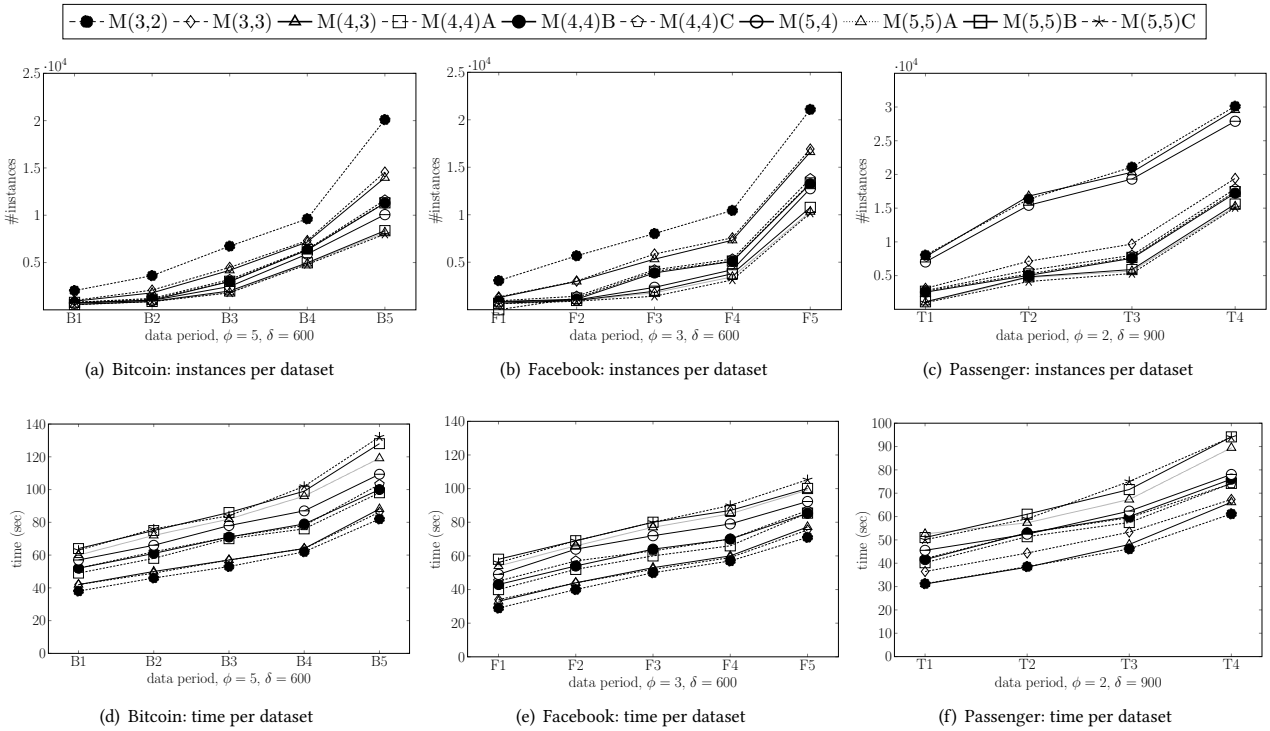
(a) Bitcoin Network (b) Facebook Network (c) Passenger Network

Figure 11: Flow of k-th instance



(a) Bitcoin Network (b) Facebook Network (c) Traffic Network

Figure 12: Efficiency of the dynamic programming module

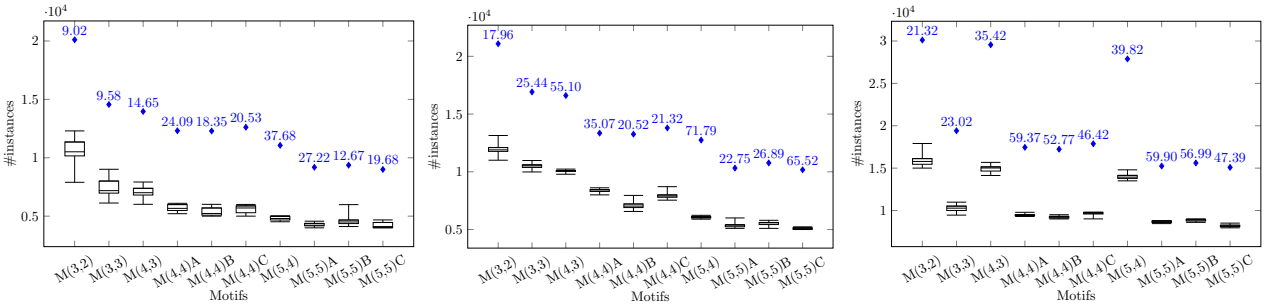


(a) Bitcoin: instances per dataset (b) Facebook: instances per dataset (c) Passenger: instances per dataset

(d) Bitcoin: time per dataset (e) Facebook: time per dataset (f) Passenger: time per dataset

Figure 13: Scalability to input graph size

for motifs that contain cycles, indicating that large flow movements that close a cycle are statistically over-represented in the bitcoin network. A similar observation holds for the Passenger flow network, where three out of the top-three motifs contain a cycle. A different pattern emerges in the Facebook dataset, where two out of the three highest z-scores are for chains of nodes. We conjecture that this due to propagation trees of information in the Facebook network, which result in chains with significantly



(a) Bitcoin Network (b) Facebook Network (c) Passenger Network

Figure 14: Number of instances in random networks (box plots), in real networks (diamonds), and z-scores

high flow movement. It is interesting that the significance of the discovered motifs varies in the different types of interaction networks, indicating differences in the way flow is distributed.

7 CONCLUSION

In this paper, we introduced the novel concept of *network flow motifs*. To the best of our knowledge we are the first to define and study motifs in interaction networks, which consider both the temporal and flow information of the interactions. We proposed an efficient algorithm for enumerating flow motif instances in large graphs and variants of that find the top-k instances of maximal flow. We evaluated our algorithm on three real datasets and demonstrated its scalability. In addition, we compared it to a baseline motif instance finding method based on joining instances of motif components and showed its superiority. Finally, we studied the statistical significance of a wide range of representative motifs on the real graphs and showed that they indeed appear more frequently than in random networks with the same characteristics. This indicates that the flow is transferred from one node to another (as opposed to being arbitrarily consumed or generated) and that there are subgraphs in the network where significant flow is transferred at certain periods of time.

In the future, we plan to investigate in more detail the distribution of motif instances in the real networks. For example, we can group the motif instances per structural match, in order to identify the structural matches (i.e., sets of vertices in the graph G) with the largest activity and how this activity is spread along the timeline. Another direction is to improve the efficiency of the algorithm, by processing multiple structural instances together in phase P2. Since two or more structural matches may share the same prefix, we can compute the flow instances of their common prefix simultaneously before expanding these instances to complete ones for the different motifs. In addition, we will work towards a version of the algorithm which focuses on counting instances of (possibly multiple) motifs without constructing them (along the direction of previous work [17]). Finally, we will generalize the definition of flow motifs to capture other graph structures besides paths (e.g., directed acyclic graphs with forks and joins) and study their search in large networks.

ACKNOWLEDGMENT

This research has been co-financed by the European Regional Development Fund of the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH – CREATE – INNOVATE (project code: T1EDK-00108).

REFERENCES

[1] Rémy Cazabet, Rym Baccour, and Matthieu Latapy. 2017. Tracking Bitcoin Users Activity Using Community Detection on a Network of Weak Signals. In *COMPLEX NETWORKS*. 166–177.
[2] Luis Enrique Correa da Rocha and Vincent D. Blondel. 2013. Flow Motifs Reveal Limitations of the Static Framework to Represent Human interactions. *CoRR* abs/1303.3245 (2013).
[3] Manuel Gomez-Rodriguez, Jure Leskovec, and Andreas Krause. 2012. Inferring Networks of Diffusion and Influence. *TKDD* 5, 4 (2012), 21:1–21:37.
[4] Saket Gurukar, Sayan Ranu, and Balaraman Ravindran. 2015. COMMIT: A Scalable Approach to Mining Communication Motifs from Dynamic Networks. In *SIGMOD*. 475–489.
[5] Petter Holme. 2015. Modern temporal network theory: A colloquium. *CoRR* abs/1508.01303 (2015). arXiv:1508.01303 http://arxiv.org/abs/1508.01303
[6] David Kempe, Jon M. Kleinberg, and Amit Kumar. 2002. Connectivity and Inference Problems for Temporal Networks. *J. Comput. Syst. Sci.* 64, 4 (2002), 820–842.
[7] Dániel Kondor, Márton Pósfai, István Csabai, and Gábor Vattay. 2013. Do the rich get richer? An empirical analysis of the BitCoin transaction network. *PLoS ONE* 9, 2 (2013), e86197.
[8] Lauri Kovanen, Márton Karsai, Kimmo Kaski, János Kertész, and Jari Saramäki. 2011. Temporal motifs in time-dependent networks. *CoRR* abs/1107.5646 (2011). arXiv:1107.5646 http://arxiv.org/abs/1107.5646
[9] Rohit Kumar and Toon Calders. 2017. Information Propagation in Interaction Networks. In *EDBT*. 270–281.
[10] Rohit Kumar and Toon Calders. 2018. 2SCENT: An Efficient Algorithm to Enumerate All Simple Temporal Cycles. *PVLDB* 11, 11 (2018), 1441–1453.
[11] Jure Leskovec, Mary McGlohon, Christos Faloutsos, Natalie S. Glance, and Matthew Hurst. 2007. Patterns of Cascading Behavior in Large Blog Graphs. In *SDM*. 551–556.
[12] Yuchen Li, Zhengzhi Lou, Yu Shi, and Jiawei Han. 2018. Temporal Motifs in Heterogeneous Information Networks. In *MLG Workshop @ KDD*.
[13] Julian J. McAuley and Jure Leskovec. 2012. Learning to Discover Social Circles in Ego Networks. In *NIPS*. 548–556.
[14] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. 2013. A fistful of bitcoins: characterizing payments among men with no names. In *IMC*. 127–140.
[15] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon1. 2004. Network Motifs: Simple Building Blocks of Complex Networks. *Science* 298, 5594 (2004), 824–827.
[16] Satoshi Nakamoto. 2007. Bitcoin: A peer-to-peer electronic cash system http://bitcoin.org/bitcoin.pdf.
[17] Ashwin Paranjape, Austin R. Benson, and Jure Leskovec. 2017. Motifs in Temporal Networks. In *WSDM*. 601–610.
[18] Sayan Ranu and Ambuj K. Singh. 2009. GraphSig: A Scalable Approach to Mining Significant Subgraphs in Large Graph Databases. In *ICDE*. 844–855.
[19] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. *Computer Communication Review* 45, 5 (2015), 123–137.
[20] Konstantinos Semertzidis and Evaggelia Pitoura. 2016. Durable graph pattern queries on historical graphs. In *ICDE*. 541–552.
[21] Sebastian Wernicke and Florian Rasche. 2006. FANMOD: a tool for fast network motif detection. *Bioinformatics* 22, 9 (2006), 1152–1153.
[22] Rongjing Xiang, Jennifer Neville, and Monica Rogati. 2010. Modeling relationship strength in online social networks. In *WWW*. 981–990.
[23] Qiankun Zhao, Yuan Tian, Qi He, Nuria Oliver, Ruoming Jin, and Wang-Chien Lee. 2010. Communication motifs: a tool to characterize social communications. In *CIKM*. 1645–1648.
[24] Andreas Züfle, Matthias Renz, Tobias Emrich, and Maximilian Franzke. 2018. Pattern Search in Temporal Social Networks. In *EDBT*. 289–300.