Pattern Search in Temporal Social Networks

Maximilian Franzke
Institute of Informatics,
Ludwig-Maximilians-Universität
Munich, Germany
franzke@dbs.ifi.lmu.de

Tobias Emrich
Institute of Informatics,
Ludwig-Maximilians-Universität
Munich, Germany
emrich@dbs.ifi.lmu.de

Andreas Züfle
Department for Geography and GeoInformation Science
George Mason University
Fairfax, Virginia, United States
azufle@gmu.edu

Matthias Renz
Department for Computational and Data Sciences
George Mason University
Fairfax, Virginia, United States
mrenz@gmu.edu

ABSTRACT

Due to the wide availability of social media and the wide range of real-life and human-centered applications, social networks have become an attractive research area. However, the temporal aspect of relations between entities in a social network has been widely ignored. We argue that the temporal aspect of social networks is the key to understand interactions and other phenomena happening in these networks and should thus be considered more closely. In this work we address the problem of pattern search in temporal social networks, thus finding all occurrences of a temporal pattern in a large temporal social network. As a first step, we define a temporal pruning criterion, which allows to quickly reduce the search space of candidates. Then, we present an index structure which allows to quickly find the occurrences of simple temporal network structures, from which more complex query structures can be derived from. Our experimental evaluation on a real-world temporal social network shows the effectiveness of our pruning approach and our proposed index structures, reducing the search-time for small temporal patterns by many orders of magnitude.

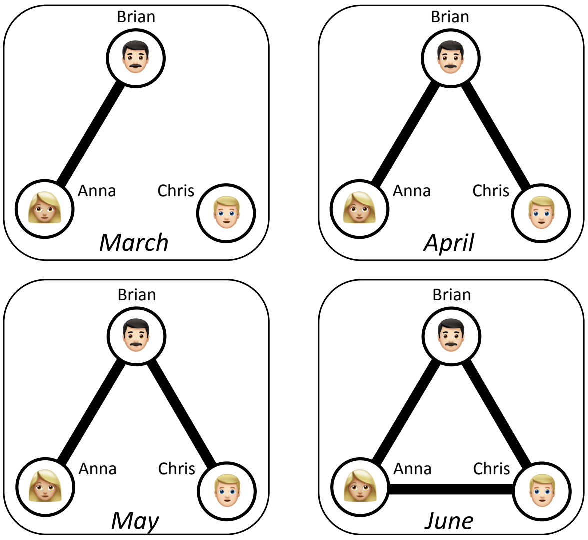


Figure 1: Example evolution of a social network.

1 INTRODUCTION

Social networks and other interaction networks[1] are dynamic by nature. Bonds of friendship can last for an eternity, but often break and fade away over time. Active collaborations between researchers naturally change over time. Thus, any social network today will look significantly different tomorrow. Keeping in mind the dynamic nature of human beings, the history of a social network showing how interactions between individuals evolve over time should be considered when inferring knowledge from it, because the knowledge about the evolution of a social network yields further semantic information. For instance, in a collaboration network it might be interesting to see in which order a group of researchers formed a network, which researchers only have short-term collaborations and which have rather long and sustainable research collaborations. However, existing solutions that consider only a snapshot in time, or building the union of all past collaborations, are not able to take such information into account. By considering this dynamic aspect of a social network, it becomes possible to identify more interesting and meaningful patterns. As a minimal example, consider Figure 1, showing an evolving social network. Initially, only one social link exists, between Anna and Brian. In the next month, an additional link is added between Brian and Chris. Finally, the triangle is closed with a link between Chris and Anna. In this example, Brian might act as a social hub, who brings Anna and Chris together. In practice, more detailed temporal social patterns could matter, as for instance the duration of a social link between Anna and Chris, which was induced by Brian, may be of interest. The temporal development of edges like in our example has been adressed for more than a hundred years through the concept of 'triadic closure', where the future creation of the third connection in a

[1] Though this paper mainly refers to social network, the proposed concepts are also applicable for other interaction networks, e.g. economical networks, etc.

Series ISSN: 2367-2005

10.5441/002/edbt.2018.26

triange is tried to be estimated, e.g. through link prediction in social networks. A more complex temporal structure in a social graph is the 'microtaboo', where it is frowned upon when persons *Alice* and *Dave* want to engage in a relationship, but there exist prior relationships between *Alice* and *Bob*, *Bob* and *Carol*, and *Carol* and *Dave* ("don't date your ex-girlfriend's boyfriend's ex-girlfriend") [2].

Another example are communication and transportation networks, where links between nodes and hubs are only established temporarily. Routing policies of computer networks may decide differently at various time points on how to create links between network nodes in order to transmit data. Transportation networks with feeder trucks, cargo aircraft and delivery vehicles link hubs differently according to current demand. To analyzes the behaviour of those networks, a temporal pattern analysis of the interaction graph can help in mining information from the graph's history to deduct findings and information for future optimization.

In this paper we address the problem of efficient evolution pattern search in large temporal social networks. Our approach bridges the gap between social network analysis and temporal logic. The contributions of this paper can be summarized as follows:

- We formally introduce the temporal subgraph matching query as a new problem.
- We introduce a language to express such pattern-based queries.
- We introduce and discuss several query filter strategies.
- We propose an index structure that allows us to search for temporal subgraph patterns in large temporal social network graphs.
- We provide a broad evaluation of the performance of our approaches based on real world datasets and show that our approach significantly outperforms state-of-the-art approaches. Though our problem is a generalization of the subgraph isomorphism problem, which is known to be NP-complete [6, 11], we can show that our index-based solution is able to find simple temporal social patterns in large real-world social networks efficiently.

We define the problem of temporal social subgraph search in Section 2. In Section 4, we propose filter strategies and introduce our new indexing method in Section 5. Our experimental evaluation is given in Section 7.

2 PROBLEM DEFINITION

We first introduce our representation of a temporal social network which we define as a graph where each node refers to an individual and each edge between two nodes is associated with a discrete function that maps time to the domain {0,1} specifying the presence and absence of the edge over time.

Definition 2.1 (Temporal Social Graph). Let $\mathcal{T} = \{0, 1, \ldots, m\}$ be a discrete time domain. A temporal social graph (TSG) $G = (V^G, E^G, F^G)$ is a graph, where $V^G = \{v_1^G, v_2^G, \ldots, v_n^G\}$ is the set of nodes, $E^G \subseteq V^G \times V^G$ the set of links (with $(v_i^G, v_k^G) \in E^G$), and
$F^G = \{f^G_{v_i^G, v_k^G}(t) | (v_i^G, v_k^G) \in E^G\}$ a set of discrete time-dependent functions, where $f^G_{v_i^G, v_k^G}(t) \in \{0, 1\}$ describes the existence of a connection between v_i^G and v_k^G (0 indicating no connection and 1 indicating there is a connection) at time $t \in \mathcal{T}$. Furthermore, an edge (v_i^G, v_k^G) is only an element of E^G if $\exists t \in \mathcal{T} : f^G_{v_i^G, v_k^G}(t) \neq 0$.

Based on this definition we are now able to define temporal subgraph matching which finds a subgraph from a TSG that exactly matches a given temporal subgraph query.

Definition 2.2. [Temporal Subgraph Matching] Let $G = (V^G, E^G, F^G)$ be a TSG defined on the time domain $\mathcal{T} = \{0, 1, \ldots, m\}$. And let $q = (V^q, E^q, F^q)$ be a query TSG defined on the time domain $\mathcal{T}^q = \{0, 1, \ldots, t^q\}$ where $t^q \leq m$. A temporal subgraph matching query retrieves the set $\mathcal{S}$ of all temporal subgraphs $\mathcal{S} \ni S = (V^S \subseteq V^G, E^S \subseteq E^G, F^S \subseteq F^G)$, such that there exists a bijection $h : V^q \rightarrow V^S$ and $\Delta_t \in \{0, ..., m - t^q\}$ that satisfies $\forall (v_i^q, v_k^q) \in E^q : (h(v_i^q), h(v_k^q)) \in E^S$ and $\forall t \in [0, t^q] :$ $f^q_{v_i^q, v_k^q}(t) = f^S_{h(v_i^q), h(v_k^q)}(t + \Delta_t)$.

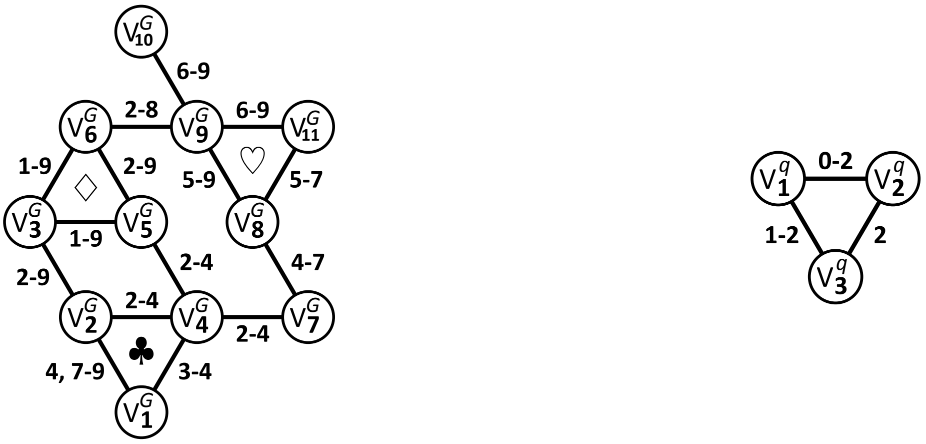
An example for a temporal social graph G is given in Figure 2(a). For convenience we labeled the edges (v_i, v_k) with the set of time steps $t \in \mathcal{T}$ for which the function $f_{v_i, v_k}(t) = 1$. Figure 2(b) illustrates a temporal subgraph query q. A pattern match of q can be found at ♣ in G for $\Delta_t = 2$, $h(v_1^q) = v_4^G$, $h(v_2^q) = v_2^G$ and $h(v_3^q) = v_1^G$. A more sophisticated query q' is depicted in Figure 6, which matches for $h(v_1^{q'}) = v_{11}^G$, $h(v_2^{q'}) = v_8^G$, $h(v_3^{q'}) = v_9^G$, $h(v_4^{q'}) = v_{10}^G$, and $h(v_5^{q'}) = v_6^G$ at $\Delta_t = 5$ (around the ♡ marker).

A summary of the notations used throughout this paper can be found in Table 1.

3 RELATED WORK

Various applications of temporal graphs and sources of temporal graph data can be found in surveys on temporals graphs [4, 10]. Existing research on temporal graphs primarily focuses to temporal paths and their applications [1, 4, 12, 13, 15, 19, 22, 23]. None of these works study the search of a given query pattern. Most related is existing work on temporal community detection over temporal networks [9, 14, 25, 26] and multi-layer networks [3]. These work first identify communities in a static network, then identify the evolution of the communities from the changes of the network. Specifically, the problem of finding dense patterns in temporal graphs has been studied in [25, 26]. This work allows to find diversified dense regions, thus minimizing the temporal and social redundancy of the returned patterns. Such diversification may also be applied to the arbitrary patterns mined in this work, but is not in this paper's scope. A recent approach [16] considers relations between edges, namely the time-respecting subgraph isomorphism problem, where edges are put into temporal sequence of each other. This is useful to model propagation in a network, but cannot handle more complex temporal constraints in the query. The authors propose a time-first, topology-first and a hybrid solution to approach their problem. A consideration of the fact that edges can appear and disappear over time is made in [18], which focuses on finding structural subgraph patterns in the graph that persist over the longest period of time. This is supported by three index structures that store label information, neighbourhood constellations and path maps.

In summary, these works can be used to find dense regions such as cliques and quasi-cliques in a temporal network, but do not allow to find patterns arbitrarily shaped over time. Thus, to



(a) Temporal Social Graph G

(b) Temporal Subgraph Query q

Figure 2: Example TSG and TSQ. Time points where the time dependent function of an edge returns a non-zero value are noted next to the edge: A dash ('-') is used to denote intervals and commas (',') indicate enumeration of timepoints or intervals. The suit markers (♡, ♣, ♢) give visual guidance for the text description.

$t \in \mathcal{T} = \{0, 1, \ldots, m\}$	time-domain (discrete)
$G = (V^G, E^G, F^G)$	temporal social graph G vertices/nodes V, edges/links E, time-dependant function $F^G = \{f^G_{v_i^G, v_k^G}(t) \in \{0, 1\}\|$ $(v_i^G, v_k^G) \in E^G\}$ ex. Figure 2(a)
$q = (V^q, E^q, F^q)$	temporal social query graph q ex. Figure 2(b) ex. Figure 6
$S \in \mathcal{S}$ $S = (V^S, E^S, F^S)$	subgraphs of G matching q
$G^\perp = (V^G, E^G)$	non-temporal projection of G
$h : V^q \rightarrow V^S$	bijection mapping
M^S	assignment map of an isomorph S to q within G size of M^S: $\|V^q\| \times \|V^G\|$
M^0	aggregation of all M^S ex. Figure 7
$SSG = (V^{SSG}, E^{SSG})$	simple subgraph structure ex. Figure 3 ex. Figure 9
♠, ♡, ♣, ♢	visual markers

Table 1: Notations used throughout this paper.

the best of our knowledge, our work is the first one for finding patterns in temporal graphs such that the query pattern exhibits temporal constraints.

A further line of related work is pattern search on static (non-temporal) graphs. The solution to this problem was proposed by Ullmann [21] and serves as our baseline. This problem, for which a survey of solutions is found in [5], still attracts vivid research attention (e.g., [17, 20]). This pattern search has further been extended to labelled vertex-graphs, as surveyed in [8]. While this research field has received extensive attention, these solutions are not applicable to our problem setting since they do not consider any time-dependent network structures, which increases the complexity of the problem.

Furthermore, solutions have been presented for the problem of finding subgraphs in a large collection of small graphs [24]. This approach first mines frequent structures and stores for each frequent structure the IDs of the graphs that contain it (similar to an inverted file). A query can then be answered by identifying the frequent structures contained in it and intersecting the corresponding lists of IDs. Although, this problem setting is also fundamentally different, it nonetheless inspired us for the index structure proposed in this work.

4 BASIC TEMPORAL SUBGRAPH MATCHING

The problem of temporal subgraph matching is related to the classic subgraph isomorphism counting problem, which is to find the set $\mathcal{S}$ of *all* subgraphs of a non-temporal graph G that are isomorphic to a given non-temporal query graph q. This problem is a generalization of the NP-complete subgraph isomorphism problem [6, 11], where the challenge is to decide whether *any* such subgraph $S \in \mathcal{S}$ exists in G. Consequently, the subgraph isomorphism counting problem is NP-hard, since its result can be used in to decide the subgraph isomorphism problem in $O(1)$, by testing the number of subgraphs for positivity. The current best-known algorithm for obtaining the exact count of an arbitrary query graph q is in $O(n^{\frac{\omega k}{3}})$, where k is the size of query graph q and ω is the exponent of fast matrix multiplication [7]. Our problem is at least as hard as subgraph counting, as we want to enumerate all instances of q in G, while also considering temporal constraints on edges.

The problem of finding all subgraph isomorphisms on static (non-temporal) graphs can be extended to temporal subgraphs as follows: Given a temporal query subgraph $q = (V^q, E^q, F^q)$, initialize the non-temporal graph $q^\perp = (V^q, E^q)$, where $F^{q^\perp} \neq 0$. In informal words, two nodes in $q^\perp$ are connected iff their corresponding vertices in q are connected at least at one point of time. This can be seen as a projection of the temporal query q to a single point of time. Now this projection is applied on the temporal graph $G = (V^G, E^G, F^G)$ as well, yielding the non-temporal graph $G^\perp = (V^G, E^G)$. Now solve the subgraph isomorphism problem on the "flattened" by finding the set of all subgraphs $\mathcal{S}^\perp$ of $G^\perp$ that are isomorphic to $q^\perp$. This yields, for each any edge e in any resulting graph $S^\perp \in \mathcal{S}^\perp$ a mapping $h(e)$ to an edge in G as described in Definition 2.2.

Since the temporal subgraph query is more selective than the non-temporal query by considering temporal constraints, an additional refinement step is necessary. For any $S^\perp \in \mathcal{S}^\perp$ to be verified as a result of the overall temporal subgraph query, there must exist a time offset Δ_t such that the time-dependent function F^q matches the time dependent function of $\mathcal{S}$. More formally, $S^\perp = (V^S, E^S)$ satisfies the temporal subgraph query of q iff $\exists \Delta_t \in \{0, ..., m - q^t\} : \forall e \in S^\perp : \forall t \in [0, t^q] : f_e^q(t) = f_{h(e)}^G(t + \Delta_t)$.

4.1 Subgraph Isomorphism in Non-Temporal Graphs

Ullmann [21] introduces a viable method for solving the (non-temporal) subgraph isomorphism problem, which we will extend and briefly describe in this chapter: Let G be a non-temporal graph, which is the special case of a temporal graph having a singular time domain $\mathcal{T} = \{0\}$ and having all edges in E^G exist at time 0. For every subgraph $S = (V^S, E^S) \in G$ that is isomorph to a query graph q, we can define a $|V^q| \times |V^G|$ matrix M^S, such that $M^S_{i,j} = 1$ iff $h(v^q_i) = v^G_j$ and 0 otherwise. M^S can therefore be interpreted as an assignment map that locates the vertices of the subgraph in the larger graph. Note that in every row of M^S, there is exactly one cell with the value of 1, while in every column there is at most one cell to contain a 1.

Let furthermore M^0 be a matrix having the same dimensions as M^S and $M^0_{i,j} = 1 - \prod_{S \in G}(1 - M^S_{i,j})$, so that M^0 gives information about whether there exists any unspecified subgraph S so that $h(v^S_i) = v^G_j$. It is now possible to retrieve the individual subgraph matrices M^S from M^0, along with possible other matrices, which do not belong to a valid subgraph query solution: Alter the cells of M^0 by setting different cells from 1 to 0, until the constraints of a subgraph representation are fulfilled (exactly one 1 per row, max. one 1 per column).

The main idea is to mine candidate subgraphs S from M^0, which are matching in the "flattened" graph $G^\perp$ that is oblivious to the time constraints. Therefore, a cell $M^0_{i,j} = 0$ implies that there exists no subgraph S where $h(v^S_j)$ would map vertex v^S_j to vertex v^G_i in the social network. Analogously, a value of 1 implies that such a graph may possibly exist. A trivial case is to set every $M^0_{i,j}$ to 1, which means that every vertex in G can be part in any subgraph that fulfills q. But in order to improve the runtime of the algorithm, it is desireable to reduce the number of subgraph combination. This can be achieved by setting as many cells in M^0 to 0 as possible. There are different methods to prune candidates with varying complexity and efficiency:

- **Pruning based on a node's degree**. If $deg(v^S_i) > deg(v^G_j)$, then $M^0_{i,j}$ can be set to 0. Note however, that in social networks the degree of the vertices is usually much higher than the degree of vertices in the query pattern q, which is why this approach yields limited pruning power.
- **Pruning based on invalid neighbour mappings**. Vertices can be pruned if there is no valid assignment for its neighbours, although the node itself can be mapped between S and q. More formally, a cell $M^0_{i,j} = 1$ can be set to 0 (and thus be pruned), if there is a neighbour vertex v^S_u of v^S_i (i.e. $(v^S_i, v^S_u) \in E^S$) and no neighbour v^G_w of v^G_j (i.e. $(v^G_j, v^G_w) \in E^G$) such that $M^0_{u,w} = 1$:

$$M^0_{ij} \leftarrow M^0_{i,j} \cdot \left(\prod_{(v^S_i, v^S_u) \in E^S} \left(1 - \prod_{(v^G_j, v^G_w) \in E^G} (1 - M^0_{uw})\right)\right)$$

  Pruning a cell may allow for further pruning of other cells, so a new pruning iteration should be invoked after a successfully setting a cell to 0. This method can be stacked with other methods to further remove further candidates after another method was successful in removing candidates.

4.2 Additional Pruning Filters for Temporal Graphs

Besides the generic filter steps enlisted in Section 4.1, we furthermore introduce two more viable filter steps that can be applied in the context of *temporal* subgraph isomorphisms:

- **Pruning based on time offset.** As described before, every derived subgraph S that is a candidate to be isomorph to the query (represented through M^S) needs to be refined in the sense that there needs to exist a Δ_t so that the time-dependant functions match. When testing various Δ_t, it is feasible to create a copy $M^S_{\Delta_t}$ of M^S. Since only the time frame from Δ_t until $\Delta_t + t^q$ is relevant for the matching of the functions, the graph G can be projected onto a more sparse graph N_{Δ_t} than N by only inserting an edge into

$E^{N_{\Delta t}}$, if the corresponding edge in G exists in this more narrow time frame.

- **Pruning based on network distance.** When iterating through M^0, after setting a candidate value for the first processed row of M^0, we can try to cut down the the number of columns that can contain 1s at all. Let i be the index of a row where exactly one column j is set to one. Then we can compute the maximum hops from v_i^S to any other node in S. Then we determine all nodes in the graph N, whose hops distance to v_j^N is larger than that distance. Those columns can then be set to zero. This is a viable approach if $|V^N| \gg |V^S|$ and N is sparse. For efficiency reasons, it is recommended to pick the first row i in a way so that v_i^S lies in a *central* position in S, e.g. minimizing the maximum hops distance. As temporal aspects are not a pruning criterion for this filter, it can generally be applied to non-temporal subgraph isomorphism queries as well.

In our experiments, we will take a deeper look at the effectiveness and performance of the basic and our extended filters. We will also evaluate the processing order, in which these filters are applied.

5 AN INDEX STRUCTURE FOR TEMPORAL GRAPHS

In this section we will give an in depth description of how to build an index for temporal social graphs and how to perform pattern queries on this index.

The construction of an index structure that supports subgraph pattern search on temporal graphs can be summarized in four steps: (1) Select one or more simple subgraph structures (SSGs) and do the following steps for each of them. (2) Find each occurrence of the SSG in the graph G without consideration of the temporal aspect. (3) Transform each occurrence into a string reflecting its unique behaviour over time considering the functions f of the edges. (4) Index the obtained strings using an index structure for substring search. In the following we will consider each of these steps in detail.

Simple Subgraph Structure Selection: The selection of suitable SSGs ($SSG = (V^{SSG}, E^{SSG})$) is crucial for the performance of the index, since the index can later on only answer queries that contain at least one of the selected SSG. A good set of SSGs should thus contain even the simplest possible query structures. Let us note that a temporal pattern query on a TSG must involve at least a relationship (edge) of two entities (nodes). The most simple SSGs involving 2 and 3 nodes, illustrated in Figure 3, should thus always be indexed in order to allow index support for all possible queries. When challenging the trade off between simplicity and ubiquitousness of SSGs, multiple different SSGs may be indexed in parallel to suit a wider array of queries. In the following we will showcase the construction of the index based on the triangle structure.

Finding SSG Occurrences: To find all occurances of the SSG in the graph, the temporal aspect of G will be neglected thus using "flattened" version of $G^{\perp}$ of G as used in Section 4. Within $G^{\perp}$, we search for all occurrences of the SSG using a traditional subgraph isomorphism algorithm such as [21]. An SSG occurs at a set of nodes $V^O \subset V^G$ iff $(v_i^O, v_j^O) \in E^N \Leftrightarrow (v_i^{SSG}, v_j^{SSG}) \in E^{SSG}$. Please note that due to possible symmetries, several occurrence can happen for the same set of V^O, depending on how V^O is mapped to V^{SSG}.

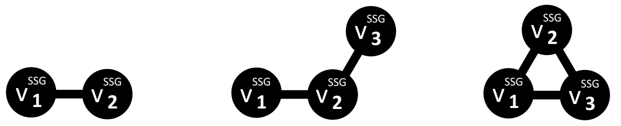


Figure 3: Simple Subgraph Structures (SSGs) with 2 and 3 nodes

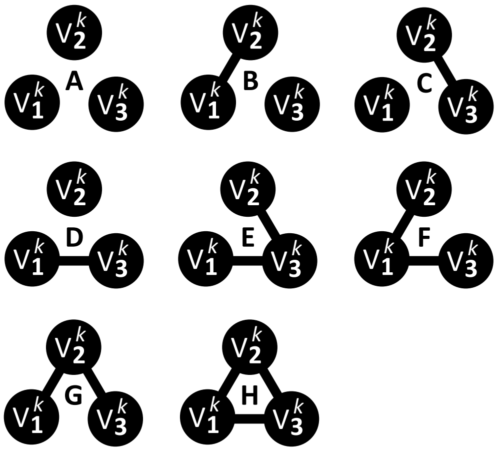


Figure 4: Encoding the edges of a graph. Each possible graph topology is encoded by an uppercase Latin character A-H.

Example 5.1. The triangle SSG occurs within the graph of Figure 2(a) at the positions marked with ♡, ♣, and ◇. If there would be an edge (v_6^G, v_{10}^G) at time 1, the set of nodes v_6^G, v_9^G, v_{10}^G would also form an occurrence, even though at no point of time an actual triangle is formed between the nodes.

The identification of those SSG occurances does not come free of cost: In particular, finding all subgraphs of G that are an occurance of the SSG has a runtime complexity of $O(n^{\frac{\omega k}{3}})$, where k is the size of query graph q and ω is the exponent of fast matrix multiplication [7]. However, this task can be performed offline and will not affect the query performance. Furthermore, the actual runtime is in general low enough due to two reasons: First, the SSGs are usually very small (in our example not more than 3 nodes) and second, the graph G is not fully connected in a real-world setting.

String Transformation: At every discrete point of time, a set of nodes in the graph that belong to an SSG can form a certain constellation via their edges. Figure 4 shows all possible combinations for a triangle SSG consisting of three, distinguishable nodes (v_1^{SSG}, v_2^{SSG}, and v_3^{SSG}) and assigns each constellation a unique symbol; here we are using uppercase Latin characters. To encode an SSG's temporal behaviour over time, at each time frame the currently present edges have to be figured out and be mapped via a pre-defined assignment table (such like Figure 4) to a unique symbol or character. In general, the alphabet to encode all constellations of an SSG having k edges consists of 2^k

characters. Concatenating these characters along the chronological time-series will yield in a string representing the temporal behaviour.

Example 5.2. The graph q denoted in Figure 2(b) shall be represented by a string using the triangle SSG. When each node v_i^q is mapped to v_i^{SSG}, the symbol representing the graph constellation at each time frame t_j $(0 \leq j \leq t^q)$ can be looked up in Figure 4 (t_0: *B*, t_1: *F*, t_2: *H*), thus yielding the string *BFH*. For other possible mappings of v^q to v^{SSG} the string representations *BGH*, *CEH*, *CGH*, *DEH*, and *DFH* are valid as well.

This schema can be applied to all substructure occurrences found in the graph, so that each occurrence's temporal behaviour can be described through a string. It is then feasible to index those strings in a way to efficiently support substring search. We propose to employ a suffix-tree to index these substrings concisely.

Example 5.3. Consider the triangle SSG. It occurs three times in our example graph G (Figure 2(a)), namely at ♡, ♣, and ◇. For every occurrence, there exist six possible permutations of how the substructure can occur at this position, due to the ways v_i^G may be mapped to v_i^{SSG}. We depict all of these occurrences and permutations in Figure 5.

6 QUERY PROCESSING

Next, we describe how our string-index can find all occurrences of a given temporal query pattern q. As described in Section 5, in the following, we assume that an index has been build for a specific simple subgraph structures SSG.

(1) Identify occurrences of the SSG in the "flattened" temporal graph query $q^{\perp}$.
(2) For each such occurrence, perform the same string transformation than performed for the index (i.e., use the same character map).
(3) Index-supported search for the transformed string to find candidates for verification.
(4) Refine the candidates through verifying that the part of q which is not contained in the SSG is isomorph to the surrounding of a candidate.

In more detail, to answer queries according to Definition 2.2 using the index support of the suffix-tree, we first have to isolate those SSG occurrences in the graph topology of the query graph of the SSG that was used for the string transformation process before. An SSG may occur not at all, once, or multiple times in the graph. If no SSG occurrence can be found, the index is of no help and the search has to default to a full scan, which is why there is a motivation to keep SSGs small and simple. In case of one or several occurrences of the SSG in q, we isolate the temporal behaviour of that part of the query graph and transform it using the same string encoding method used for the index construction.

Since the queried time frame is usually smaller than the indexed time span, the length of the string derived from the SSG occurrence in q is shorter than the length of string belonging to the occurrence in the graph which is stored in the index. To answer the query, we now must find all those strings in the index that contain the substring belonging to the query.

Example 6.1. **Identification and String Transformation of SSG in query** q**.** In our example query q' (Figure 6), the triangle SSG occurs at the ♠ marker. Since there are six possible permutations of the occurrence, valid string transformations are *EHH*, *EHH*, *FHH*, *FHH*, *GHH*, and *GHH*. With each of these unique query strings, we can search our encoding index (suffix tree) for entries that contain the query string. The substring positions are indicated through bold and underlined text in Figure 5. Entries which do not contain one of the substrings can be pruned.

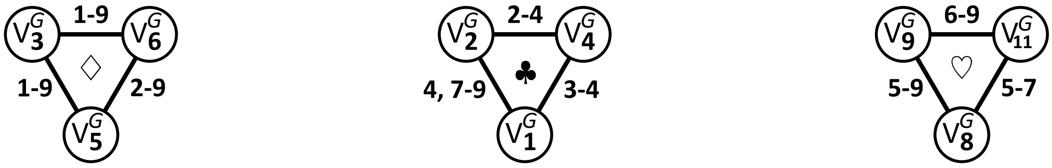
Since practical SSGs consist of more than one node, there is usually always more than one way it can be mapped to either the nodes of the occurrence in the query graph or the occurrence in the main. I.e., there are also several string transformations of the occurrence. There are in general several ways to approach these permutations:

- Account for permutations at index creation and query processing. This means that every permutation is indexed, thus resulting in a larger index, and that index is queried multiple times (once for each permutation of the query)
- Consider the permutations in the index, but not for the query
- Consider the permutations only within the query
- Neglect the permutations in the query graph and at index creation.

Neglecting all permutations may result in correct results not being found, as there is no guarantee that an 'identical' permutation has been used for the index and the query. On the other hand, if permutations are considered both times, the result will also show how the query 'fits' into the graph, i.e. the direct mapping from the nodes can be deducted. However, multiple queries have to be performed on an increased index, thus increasing query cost. As a trade-off, it is possible to only consider all permutations on one side (either the index *or* the query), and then find out the mapping in a refinement step. We recommend to consider all permutations within the index and not for the query, as submitting multiple queries increases the overall query cost linearly, while linearly enlarging the index results in a sublinear increase in query performance. Compared to the approach where permutations are considered on both sides, a refinement step is now necessary to deduct the exact mapping of the query to the substructure (one mapping per possible permutation). This is likely to be done faster after the query than it is to do multiple queries (one for each permutation).

Example 6.2. **Index-supported Search for Transformed String Representation of the SSG.** If following our advice to only consider permutations within the index, querying our example query q' (Figure 6) with a triangle SSG, the search string will either be *EHH* **or** *FHH* **or** *GHH*. The exemplary index in Figure 5 will then yield the set of nodes $\{v_3^G, v_5^G, v_6^G\}$ for the ◇, and the set $\{v_8^G, v_9^G, v_{11}^G\}$ for the ♡ occurance. However, the mapping of $v_i^{q'}$ to v_j^G can**not** be deducted from the index and has to be refined computationally.

Searching for a substring in the index then retrieves two important facts for every match: (1) a subset of V^G that corresponds to an SSG occurrence, and (2) the temporal offset Δ_t at which it occurred (calculated by the offset position of the substring from the beginning of the indexed string). Both are crucial for



Mapping			String Transformation
v_1^{SGG}	v_2^{SGG}	v_3^{SGG}	
v_3^G	v_5^G	v_6^G	AFHHHHHHHH
v_3^G	v_6^G	v_5^G	AFHHHHHHHH
v_5^G	v_3^G	v_6^G	AGHHHHHHHH
v_5^G	v_6^G	v_3^G	AEHHHHHHHH
v_6^G	v_3^G	v_5^G	AGHHHHHHHH
v_6^G	v_5^G	v_3^G	AEHHHHHHHH

Mapping			String Transformation
v_1^{SGG}	v_2^{SGG}	v_3^{SGG}	
v_1^G	v_2^G	v_4^G	AACEHAABBB
v_1^G	v_4^G	v_2^G	AACGHAADDD
v_2^G	v_1^G	v_4^G	AADEHAABBB
v_2^G	v_4^G	v_1^G	AABGHAAFFF
v_4^G	v_1^G	v_2^G	AADFHAACCC
v_4^G	v_2^G	v_1^G	AABFHAACCC

Mapping			String Transformation
v_1^{SGG}	v_2^{SGG}	v_3^{SGG}	
v_8^G	v_9^G	v_{11}^G	AAAAAFHHGG
v_8^G	v_{11}^G	v_9^G	AAAAAFHHGG
v_9^G	v_8^G	v_{11}^G	AAAAAGHHGG
v_9^G	v_{11}^G	v_8^G	AAAAAEHHGG
v_{11}^G	v_8^G	v_9^G	AAAAAGHHGG
v_{11}^G	v_9^G	v_8^G	AAAAAEHHGG

Figure 5: Transforming the temporal behaviour of the triangular SSG occurances in G to strings.

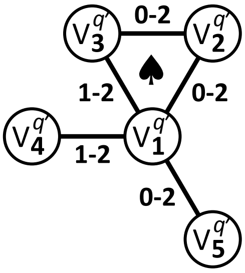


Figure 6: A more complex TSQ q'.

M^0	v_1^G	v_2^G	v_3^G	v_4^G	v_5^G	v_6^G	v_7^G	v_8^G	v_9^G	v_{10}^G	v_{11}^G
$v_1^{q'}$	0	0	0	1	0	0	0	0	1	0	0
$v_2^{q'}$	1	1	0	0	1	1	1	1	0	0	1
$v_3^{q'}$	1	1	0	0	1	1	1	1	0	0	1
$v_4^{q'}$	1	1	0	0	1	1	1	1	0	1	1
$v_5^{q'}$	1	1	0	0	1	1	1	1	0	1	1

Figure 7: Assignment matrix M^0 for q' and Q after applying the degree and neighbour filters.

$M^{\diamond 4}$	v_1^G	v_2^G	v_3^G	v_4^G	v_5^G	v_6^G	v_7^G	v_8^G	v_9^G	v_{10}^G	v_{11}^G
$v_1^{q'}$	0	0	0	1	**?!**	0	0	0	0	0	0
$v_2^{q'}$	0	0	**?!**	0	0	0	0	0	0	0	0
$v_3^{q'}$	0	0	0	0	0	**1**	0	0	0	0	0
$v_4^{q'}$	1	1	0	0	0	0	1	1	0	1	1
$v_5^{q'}$	1	1	0	0	0	0	1	1	0	1	1

Figure 8: Assignment matrix $M^{\diamond 4}$ after applying the assignment from the index candidate. This means that the bold cells should be set to 1. However, cells depicted with '?!' already contain a 0 (ref. Figure 7) and cannot be set to 1 in a valid way.

efficiently refining the candidates. A refinement is necessary, as the query may be more restrictive than the SSG itself, i.e. through additional edges attached to the SSG, the retrieved candidates are a superset of the results. Therefore, this set has to be refined, i.e. it has to be checked whether the found SSG is part of a larger subgraph structure that can fulfill the query constraints with regards to graph structure as well as temporal behaviour. To refine the candidates, we return to the concept of Section 4.1. Before evaluating the substring candidates, we can still apply the degree filter for q' and G to M^0, as it is a quick way to eliminate some impossible assignments.

Example 6.3. Figure 7 shows M^0 for our sample query q' after applying the degree and neighbour filters. It shows that for example $v_1^{q'}$ can only be mapped to nodes with a degree of at least 4, thus leaving only cells $M^0_{1,4}$ and $M^0_{1,9}$ in the first row with a 1. On the other hand, $M^0_{2,3}$ is set to 0, as $v_2^{q'}$ cannot be mapped to v_4^G, as v_4^G (unlike $v_2^{q'}$) does not have a neighbour of degree 4.

After optimizing M^0, for each found SSG x, we initialize a copy of M^0 denoted as M^x and thereby set $M^x_{i,j} := M^0_{i,j}$. Each found SSG instance x gives us a 'hint', where an occurrence of q' in G may occur as well as the temporal offset Δ_t at which the temporal pattern of the subgraph structure matched. This hint will either lead to a correct result or may be false – but its correctness will not have any effect on other SSG instances, which is why we can process them individually and in parallel. Since the index found a matching SSG which implicitly matches the vertices of the structure to vertices of the graph (after considering the permutations of the mapping), we can assign $|V^{SSG}|$ fix assignments, thus nullifying all other cells in those rows (since M^x is usually much wider than tall, this drastically increases the number of cells containing a 0). Since Δ_t is known at this point of time, we can now apply the more sophisticated time offset and network distance filters (Section 4.2) to M^x.

Example 6.4. Let us return to our running example, where we search for occurrences of the more complex query q' (Figure 6) in the running social net work G (Figure 2(a)). Let's consider the fourth permutation of the $\diamond$ substructure, which is returned as a candidate through the query permutation *EHH* ($v_1^{q'} \rightarrow v_1^{SSG}$, $v_2^{q'} \rightarrow v_3^{SSG}$, $v_3^{q'} \rightarrow v_2^{SSG}$) (see Figure 5). Mapping this SSG to G yields the mappings $v_1^{q'} \rightarrow v_1^{SSG} \rightarrow v_5^G$; $v_2^{q'} \rightarrow v_3^{SSG} \rightarrow v_3^G$; and $v_3^{q'} \rightarrow v_2^{SSG} \rightarrow v_6^G$ also shown in Figure 5. We attempt to make this assignment in the corresponding matrix $M^{\diamond 4}$ depicted in Figure 8. Therefore, we have to set the values in $M_{1,5}^{\diamond 4}$, $M_{2,3}^{\diamond 4}$ and $M_{3,6}^{\diamond 4}$ to 1, but we see that the first two entries already contain a zero in the global assignment matrix M^0, such that this assignment does not yield a valid matching. Therefore we can stop here and prune this candidate. In fact, in our example we can prune all permutations of $\diamond$ in the same way, as well as permutations 1, 3, 5, and 6 of $\heartsuit$; with $\clubsuit$ not even providing candidates. In summary, this just leaves $M^{\heartsuit 2}$ and $M^{\heartsuit 4}$ for further refinement.

We are now left with a set of matrices that we need to derive final assignment candidates from. A naive way would be to iterate over all possible assignments and verify them; one would do that by choosing a single $M_{i,j}^X = 1$ and using it as an assignment, thus nullifying other values row i and column j, and then proceeding to row $i+1$, re-applying the same concept. This needs to be done iteratively for all $m_{i,j}^X = 1$ of a row. We aim to improve this exponentially expensive approach by using heuristics. Therefore, we first take effort in finding a clever order of which we process the rows. We process rows in breadth-first-order starting at the corresponding SSG occurrence, skipping lines having a "1" assigned by the SSG occurrence. For any other line b, we look at any previous row a such that $(v_a^{q'}, v_b^{q'}) \in E^{q'}$, i.e., a neighbour of $v_b^{q'}$ that we have already assigned. Since we always start with a pre-assigned row and proceed in a breadth-first-manner, such a row must exist. That row a contains a single assignment $h(v_a^q) = v_c^G$. Assuming this assignment is correct, we only need to look at columns d where v_d^G is a neighbour of v_c^G, thus having $(v_c^G, v_d^G) \in E^G$. This can be deducted from the following:

$$h(v_a^q) = v_c^G \wedge h(v_b^q) = v_d^G \wedge (v_a^q, v_b^q) \in E^q$$
$$\Rightarrow (v_c^G, v_d^G) \in E^G$$

because $E^q \subseteq E^G$.

For every neigbour v_d^G where $M_{b,d}^x = 1$ and the temporal patterns match, we create a copy of the current M^x, nullify all other cells in row b and proceed to the next row in M^x. When the last row is reached, every vertex in v^q has been assigned exactly one partner in v^G, thus being a result.

Example 6.5. For our running example, we retrieve the following assignments from $M^{\heartsuit 2}$ and $M^{\heartsuit 4}$:

$$M^{\heartsuit 2} : h(v_1^{q'}, v_2^{q'}, v_3^{q'}) = (v_9^G, v_8^G, v_{11}^G)$$

$$M^{\heartsuit 4} : h(v_1^{q'}, v_2^{q'}, v_3^{q'}) = (v_{11}^G, v_8^G, v_9^G)$$

The later one is invalid, as $M_{1,11}^0$ is already 0 and $v_1^{q'}$ can therefore not be mapped to v_{11}^G. Following the first assignment, we now retrieve two final candidates for the complete occurance of q':

$$C_1 : h(v_1^{q'}, v_2^{q'}, v_3^{q'}, v_4^{q'}, v_5^{q'}) = (v_9^G, v_8^G, v_{11}^G, v_6^G, v_{10}^G$$

$$C_2 : h(v_1^{q'}, v_2^{q'}, v_3^{q'}, v_4^{q'}, v_5^{q'}) = (v_{11}^G, v_8^G, v_9^G, v_{10}^G, v_6^G)$$

of which the first one C_1 is invalid, as the time-dependant-function on edges $(v_1^{q'}, v_4^{q'})$ and (v_9^Q, v_6^Q) do not match. C_2 is a valid result to the query.

7 EXPERIMENTS

In this section we show experimental results of our proposed methods. As a baseline approach, we resort to Ullmann's algorithm as described in Section 4.1. This represents the expansion of traditional solutions to the temporal domain. We further evaluate the included filters as well as our proposed additional filters individually and in combination to distinguish the naive baseline approach from a more advanced setup. We then compare this baseline approach, which has been extended to temporal graphs, to our advanced query processing approach proposed in Section 6 supported by the index structured introduced in Section 5. Additionally, we introduce the evaluated queries and the employed datasets. All experiments were performed on a 3Ghz workstation having 32 GB of RAM. The experiments were run on a single core.

7.1 Datasets

As a datasource for our real data evaluation we use a snapshot of the ACM Digital Library[2] taken on Dec 15, 2014 consisting of 582,150 publications with author information. Using only the co-author relationship for each calendar year, we build a temporal social graph $G = \{V^G, E^G, F^G\}$, reflecting the collaboration network over time, in the following way:

- Each researcher present in the dataset is represented by a node v^G.
- Two researcher nodes v_i^G and v_j^G are connected by an edge (v_i^G, v_j^G) if they have at least co-authored one publication at any time $t \in \mathcal{T}$.
- The time dependent function for an edge (v_i^G, v_j^G) indicates collaboration over time and is set the following way:

$$f_{v_i^G, v_j^G}^G(t) = \begin{cases} 1 & \text{if } v_i^G \text{ and } v_j^G \text{ co-authored a publication in year } t \\ 0 & \text{if } v_i^G \text{ and } v_j^G \text{ did not co-author a publication in year } t \end{cases}$$

The resulting temporal social graph is called *PUBS* in the remainder of this section. In order to evaluate naive approaches of the proposed algorithms we also use small subgraphs of PUBS

[2] http://dl.acm.org/

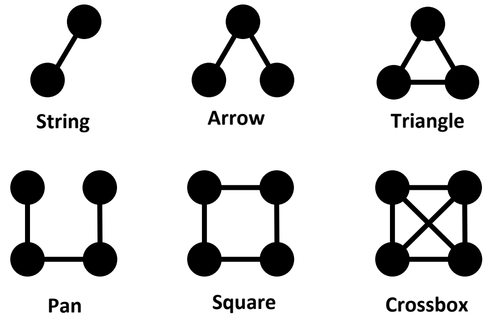


Figure 9: Evaluated subgraph structures in our experiments along with assigned names.

	PUBS	MiniPUBS	MicroPUBS
Nodes	379,188	10,000	3,792
Edges	2,114,720	77,568	36,548
Timepoints	69	46	43
# Strings	2,114,720	77,568	36,548
# Arrows	44,379,646	1,541,152	917,810
# Triangles	13,191,264	422,736	176,286
# Squares	449,684,160	5,179,608	2,259,456
# Pans	1,438,921,874	25,601,204	21,972,438
# Crossboxes	411,978,792	3,967,824	1,196,184

Table 2: Characteristics of the researcher collaboration datasets

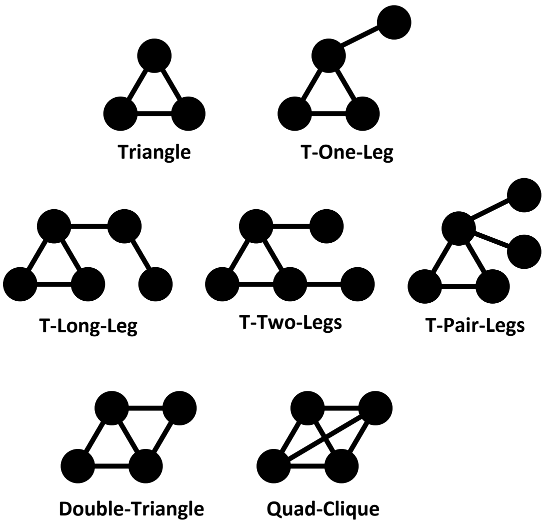


Figure 10: Small queries used in our experiments.

called *MiniPUBS* and *MicroPUBS*, with 10000 nodes and 3792 nodes, respectively. These subgraphs were generated from PUBS by performing a breadth-first search rooted at the first two (anonymous) authors of this work.

Table 2 summarizes the characteristics of the three datasets. In addition to the number of nodes, the number of edges and the duration of the network in years Table 2 contains the number of the subgraph structures illustrated in Figure 9.

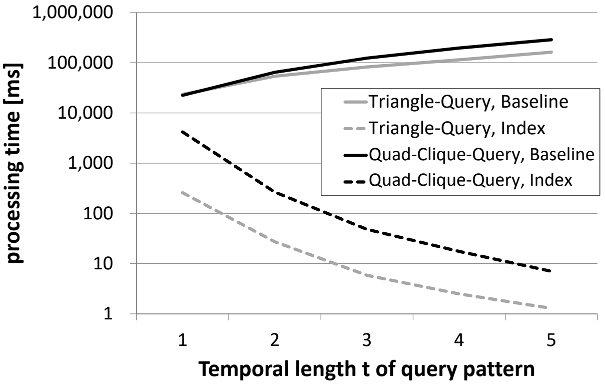


Figure 11: Querying with a triangle and a crossbox structure of constant edges and variable length t^q.

7.2 Queries

Figure 10 shows a set of standard query subgraphs that we used in our experimental evaluation. In the first set of experiments, the query time domain t^q is set to relatively small values of $3 \leq t^q \leq 5$, to all the baseline approaches to terminate in reasonable time. The temporal pattern on these subgraphs is chosen uniformly random, such that at each point of time $t \in \mathcal{T}^q$ of the query graph, any edge has a chance of 50% to be part of the query pattern. To avoid degenerated cases, we ensure that each edge of a standard query is required to exist at at least one time t.

7.3 Baseline vs. Index

In a first experiment we compare the performance of our proposed index structure (cf Section 5) with the baseline approach as discussed in Section 4. Since the baseline approach is very time consuming we did this experiment on the MicroPUBS dataset. As a basic subgraph structure for our index we used the triangle and as queries we evaluated the triangle-query and the quad-clique-query with increasing temporal query length t^q. The results are shown in Figure 11. The baseline approach has to build a projection of the original TSG for each possible start time t over the duration of the query. With increasing duration of the query, this projection becomes more and more dense, which results in increasing runtime. The index based query processing on the other hand performs much faster in this setting. Note that, although the triangle query is beneficial to the index (since the index is built on the triangle structure), the quad-clique query can also be answered efficiently. With increasing query duration, the results quickly decrease, yielding a lower number of candidates, which leads to even lower runtime.

Figure 12 shows the query time for various query patterns. As most of the 4-year-long queries have a highly specific temporal pattern, the index-based approach profits from early pruning of large parts of the data, while the baseline approach is first looking for the general graph structure and can only prune at a second step where the temporal behaviour is considered. We see that the *Triangle* query for $t^q = 3$ has the highest run-time using our index, while having the lowest run-time of the baseline. The reason is that this query yields the largest number of (verified) results which, trivially, cannot be pruned. When changing the query time t^q for the more complex *T-Two-Legs* query, we can

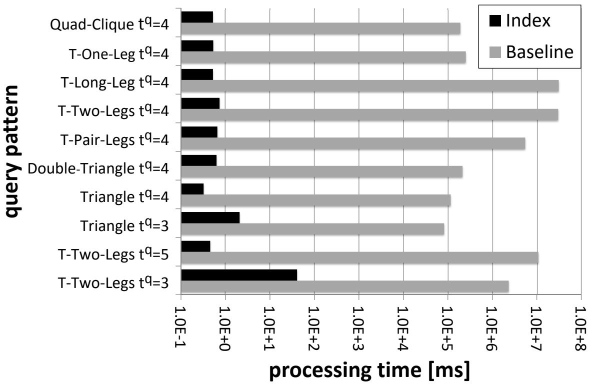


Figure 12: Querying with a distinct queries.

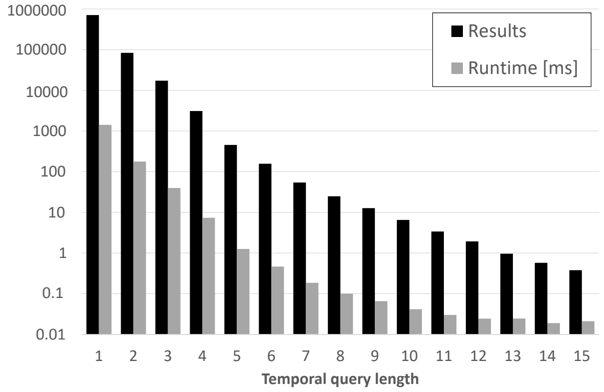


Figure 13: no. of results and runtime with increasing query length

again observe that our index supported approach can benefit from early pruning. Note that the time needed to build the index for this microPUBS dataset was less than a second.

7.4 Evaluating query parameters

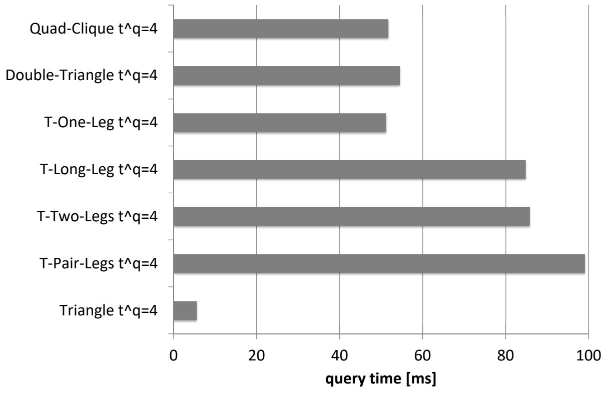


Figure 14: query time for harder queries

In the next set of experiments we demonstrate the effect of the query duration t^q on the large PUBS dataset. The *Baseline* approach was not evaluated on this dataset due to its excessive run-time. In the first experiment, shown in Figure 13, we use the

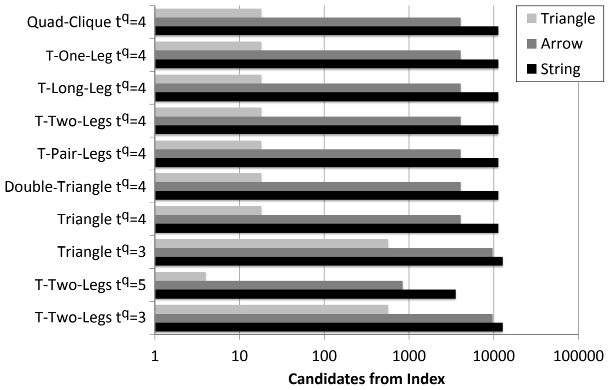


Figure 15: Testing different indexed subgraph structures: number of candidates

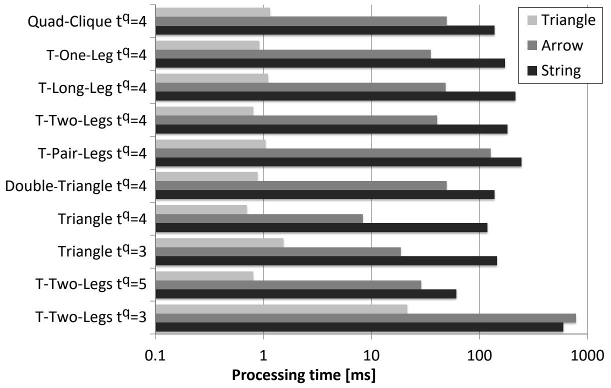


Figure 16: Testing different indexed subgraph structures: processing time

Triangle structure for query and indexing. Index construction on this large dataset took less than 30 minutes. Results using this index are averaged over 1K runs for random temporal patterns of the PUBS dataset. We observe that the run-time is directly proportional to the number of results. This behaviour is expected, since less refinement is needed.

The previous experiment show-cased a best-case scenario, where the structure used for indexing is identical to the query structure. In this case, candidates returning by the index need only to be verified for temporally matching the query pattern.

To show the behaviour in more realistic scenarios, we made the topological structure of the query more complex by adding additional edges, while still using triangle SSGs for indexing. Thus, those added edges are not covered by the index and need to be accounted for in the refinement step. Figure 14 shows that adding additional edges and nodes to the query increases the processing time: Adding two edges to the triangle produces a more complex query than just adding one (than adding none), and a Quad-triangle is more specific than a Double-Triangle than a simple triangle.

Thus, in the next set of experiments shown in Figures 15 and 16, we test the efficiency of different SSGs on the MiniPUBS dataset. Here, we compare different structures used to build the index (specifically, the SSGs *Triangle*, *Arrow* and *String*), and different query structures for different time lengths t^q.

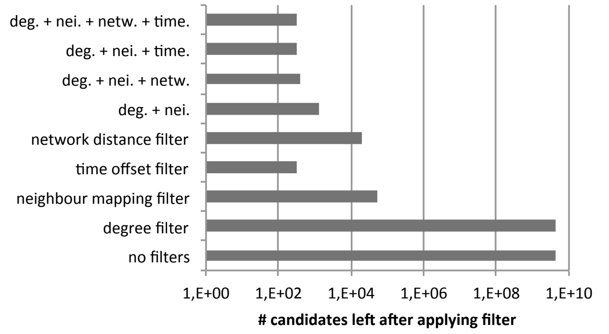


Figure 17: Comparing the number of candidates left after applying filters to M^0.

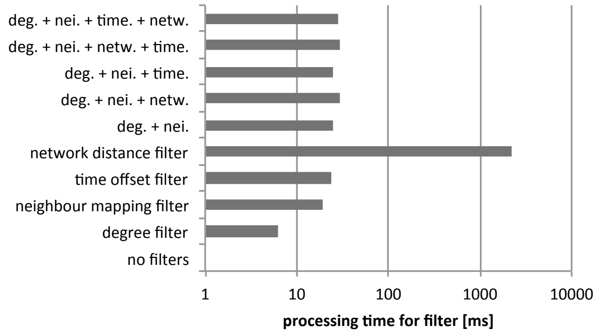


Figure 18: Measuring the computational cost of applying filters to M^0.

Comparing these results to Table 2, we notice that simpler subgraph structures appear more often in graphs, whereas more complex structures appear less and are thus generally more selective when used as the basis for an index. This also reflects in the number of candidates produced by our index-based approach for different basic query structures. We note that in these queries, using triangle structure for indexing yields much less candidates for refinement. This is because many *Arrow* and *String* structures may not be part of a triangle, thus creating additional candidates to be pruned. However, it should be noted that the triangle index is only applicable if the query structures contains at least one triangle, which is the case for query structures featured in this experiment.

7.5 Evaluating Pruning Filters

In Section 4.2 we introduced additional pruning filters applicable to temporal graphs. We evaluated the various filters on the candidates retrieved after querying the index with a simple subgraph structure (SSG) for our running query example q' on the MiniPUBS graph. Therefore, we measure the performance of different pruning strategies (and their combinations) in their number of candidates generated from M^0 that need to be refined. Figure 17 shows that the degree filter has no effect, because the only nodes not matched in the initial assignment M^0 are the two legs not part of the SSG which just have a degree of 1. The network distance filter and the neighbour mapping filter reduce the number of candidates by a factor of about $10,000$ and $30,000$, respectively. The time offset filter has the highest pruning power, reducing the number of candidates to less than two hundred. Furthermore, variations of sequential filter combinations are depicted as well: A combination allows to narrow down the candidate size even more. Figure 18 shows the corresponding computational time of the filters: the network distance filter is by far the most expensive one, even though it cannot outperform the time filter. We contribute the bad performance of the network distance filter to the dataset, in which network distances are generally very short, as the dataset was generated by a breadth-first search of a larger network. A more detailed look into the combination of filters is shown in 19, where it becomes clear that an expensive, but selective filter like the time offset filter, becomes cheap if applied after another more generic filter and in combination gives great results and a very small candidate set. We summarize that the time-offset filter is the most powerful pruning step in our setting. This is because most candidates to not match the specific temporal patterns exhibited by the temporal query graph, such that temporal pruning is very powerful. This also shows how traditional pruning methods only, which do not consider temporal patterns, are not sufficient to efficiently find patterns on temporal graphs.

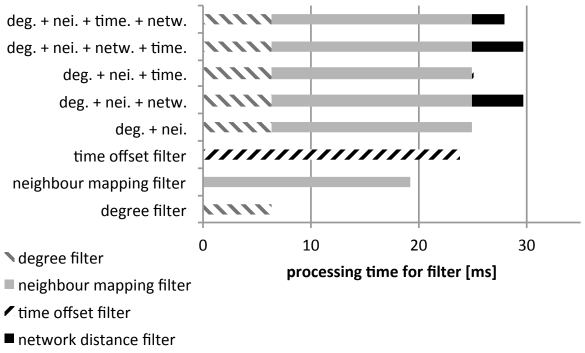


Figure 19: Measuring the computational cost of applying filters to M^0 with highlight of individual cost.

8 CONCLUSIONS

We proposed first solutions for the problem of searching patterns on temporal social networks. For this problem, existing solutions for graph isomorphism can not be applied directly, since temporal conditions need to be handled. As a baseline approach, we define a temporal pruning heuristic to augment an existing subgraph isomorphism search algorithm. Due to the high run-time of such approach on real social networks, we proposed a data structure to index all occurrences of basic graph structures, to find a candidate set of isomorphic subgraphs quickly at query time. This data structure transforms temporal graphs and temporal graph queries in strings and employs a suffix tree to organize these strings efficiently. Our experimental evaluation shows that this index structure can reduce the run-time of searching small temporal query patterns by orders of magnitude. Still challenges remain open: since the problem of isomorphic subgraph search is exponentially hard in the size of the query graph, we cannot hope to scale to large query graphs. Thus, approximate solutions are required for larger and more complex query patterns. Further, we can relax the problem to estimation of *the number* of isomorphic subgraphs, rather than returning all occurences and their location in the graph. This relaxation may allow more efficient approximations. Another important future aspect of this work is the diversification of patterns as studied in [25, 26] for dense subgraphs only. Applying such diversification to arbitrary subgraphs is a non-trivial task, as the notion of social and temporal overlap has to be redefined.

As you have seen in our running examples, temporal subgraph queries need to be defined very detailed, i.e., each configuration at every point of time needs to be stated. In cases where such level of detail is unneeded (for example when a certain link may or not exist as well at a specific point of time) our algorithm cannot specifically account for this fact. While intuitively, allowing more configurations makes queries less hard to verify, it actually increases the query complexity as the program must now consider several possible configurations instead of one. A future version could benefit in these scenarios.

REFERENCES

[1] Miguel Araujo, Stephan Günnemann, Spiros Papadimitriou, Christos Faloutsos, Prithwish Basu, Ananthram Swami, Evangelos E. Papalexakis, and Danai Koutra. 2016. Discovery of "comet" communities in temporal and labeled graphs (Com2). *Knowl. Inf. Syst.* 46, 3 (2016), 657–677. DOI:http://dx.doi.org/10.1007/s10115-015-0847-2

[2] Peter S Bearman, James Moody, and Katherine Stovel. 2004. Chains of affection: The structure of adolescent romantic and sexual networks. *American journal of sociology* 110, 1 (2004), 44–91.

[3] Brigitte Boden, Stephan Günnemann, Holger Hoffmann, and Thomas Seidl. 2012. Mining coherent subgraphs in multi-layer graphs with edge labels. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1258–1266.

[4] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. 2012. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems* 27, 5 (2012), 387–408.

[5] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2004. Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence* 18, 03 (2004), 265–298.

[6] Stephen A Cook. 1971. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*. The MIT Press, 151–158.

[7] Friedrich Eisenbrand and Fabrizio Grandoni. 2004. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science* 326, 1-3 (2004), 57–67.

[8] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Yinghui Wu. 2011. Adding regular expressions to graph reachability and pattern queries. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*. IEEE, 39–50.

[9] Mao-Guo Gong, Ling-Jun Zhang, Jing-Jing Ma, and Li-Cheng Jiao. 2012. Community detection in dynamic social networks based on multiobjective immune algorithm. *Journal of Computer Science and Technology* 27, 3 (2012), 455–467.

[10] Petter Holme and Jari Saramäki. 2012. Temporal networks. *Physics reports* 519, 3 (2012), 97–125.

[11] Richard M Karp. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 85–103.

[12] Gueorgi Kossinets, Jon Kleinberg, and Duncan Watts. 2008. The structure of information pathways in a social communication network. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 435–443.

[13] Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun. 2008. Anytime search in dynamic graphs. *Artificial Intelligence* 172, 14 (2008), 1613–1643.

[14] Yu-Ru Lin, Yun Chi, Shenghuo Zhu, Hari Sundaram, and Belle L Tseng. 2008. Facetnet: a framework for analyzing communities and their evolutions in dynamic networks. In *Proceedings of the 17th international conference on World Wide Web*. ACM, 685–694.

[15] Raj Kumar Pan and Jari Saramäki. 2011. Path lengths, correlations, and centrality in temporal networks. *Physical Review E* 84, 1 (2011), 016105.

[16] Ursula Redmond and Pádraig Cunningham. 2016. Subgraph Isomorphism in Temporal Networks. *CoRR* abs/1605.02174 (2016). http://arxiv.org/abs/1605.02174

[17] Carlos R Rivero and Hasan M Jamil. 2016. Efficient and scalable labeled subgraph matching using SGMatch. *Knowledge and Information Systems* (2016), 1–27.

[18] Konstantinos Semertzidis and Evaggelia Pitoura. 2016. Durable graph pattern queries on historical graphs. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*. IEEE, 541–552.

[19] John Tang, Mirco Musolesi, Cecilia Mascolo, and Vito Latora. 2010. Characterising temporal distance and reachability in mobile and online social networks. *ACM SIGCOMM Computer Communication Review* 40, 1 (2010), 118–124.

[20] Ha-Nguyen Tran, Jung-jae Kim, and Bingsheng He. 2015. Fast subgraph matching on large graphs using graphics processors. In *International Conference on Database Systems for Advanced Applications*. Springer, 299–315.

[21] J. R. Ullmann. 1976. An Algorithm for Subgraph Isomorphism. *J. ACM* 23, 1 (Jan. 1976), 31–42. DOI:http://dx.doi.org/10.1145/321921.321925

[22] Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. 2014. Path problems in temporal graphs. *Proceedings of the VLDB Endowment* 7, 9 (2014), 721–732.

[23] B Bui Xuan, Afonso Ferreira, and Aubin Jarry. 2003. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science* 14, 02 (2003), 267–285.

[24] Xifeng Yan, Philip S. Yu, and Jiawei Han. 2004. Graph Indexing: A Frequent Structure-based Approach. In *Proc. SIGMOD*. 335–346.

[25] Yi Yang, Da Yan, Huanhuan Wu, James Cheng, Shuigeng Zhou, and John CS Lui. 2016. Diversified temporal subgraph pattern mining. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2016), San Francisco, CA, USA*. 1965–1974.

[26] Zhengwei Yang, Ada Wai-Chee Fu, and Ruifeng Liu. 2016. Diversified Top-k Subgraph Querying in a Large Graph. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 1167–1182.