Time-Constrained Graph Pattern Matching in a Large Temporal Graph

Yanxia Xu[1], Jinjing Huang[1], An Liu[1], Zhixu Li[1], Hongzhi Yin[2], and Lei Zhao[1]

[1]School of Computer Science and Technology, Soochow University, China
[2]School of ITEE, The University of Queensland, Brisbane, QLD, Australia
xyx.edu@gmail.com;huangjj@siit.edu.cn;h.yin1@uq.edu.au
{anliu,zhixuli,zhaol}@suda.edu.cn

Abstract. Graph pattern matching(GPM) is an important operation on graph computation. Most existing work assumes that query graph or data graph is static, which is contrary to the fact that graphs in real life are intrinsically dynamic. Therefore, in this paper, we propose a new problem of Time-Constrained Graph Pattern Matching(TCGPM) in a large temporal graph. Different from traditional work, our work deals with temporal graphs rather than a series of snapshots. Besides, the query graph in TCGPM contains two types of time constraints which are helpful for finding more useful subgraphs. To address the problem of TCGPM, a baseline method and an improved method are proposed. Besides, to further improve the efficiency, two pruning rules are proposed. The improved method runs several orders of magnitude faster than the baseline method. The effectiveness of TCGPM is several orders of magnitude better than that of GPM. Extensive experiments on three real and semi-real datasets demonstrate high performance of our proposed methods.

1 Introduction

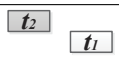
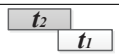
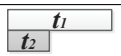
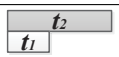
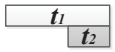
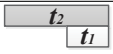
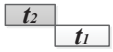
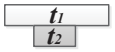
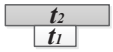
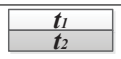
Graph pattern matching(GPM) plays a significant role in many fields, such as information retrieval[1], community detecting[2] and biology[3]. The development of GPM undergoes three periods. Firstly, a lot of researchers investigate the problem of querying static graph pattern in static graphs[4–9]. However, graphs in real life are inherently dynamic. Therefore, a lot of efforts have been made in querying static graph pattern in dynamic graphs[10, 11]. Recently, to discover more interesting patterns, there exists a few work about querying dynamic graph pattern in dynamic graphs[12].

However, there exist following issues in the traditional work: (1) The dynamic graph is often modeled as a series of static snapshots. However, many complex events in real life cannot be treated as a series of points, such as spread of epidemics, information diffusion, and so on[13]. A series of snapshots cannot illustrate all temporal information and a part of information is missing. (2) The traditional solutions often offer users exponential number of matched subgraphs[14]. It is daunting for users to inspect all matched subgraphs and find what they really want.

To solve the first issue, we focus on studying temporal graphs. Edges and vertices in a temporal graph exist during a period of time. Temporal graphs can show more information than a series of snapshots. For example, we can clearly see the phenomenon in a temporal graph that event "fever" is during the event "flue". The phenomenon is important for therapy. However, it cannot be illustrated by a series of snapshots.

In order to settle the second issue, we design two types of time constraints in the query graph to reduce the number of matched subgraphs drastically. The first type of time constraint is directed against a single object. For example, we want to find an old classmate who worked at Google for a few years around 2006-2008. The second type of time constraint is directed against the temporal relation among several objects. ALLEN [15] divides the temporal relation into 13 categories which are shown in Table 1.

Table 1. Presentation of ALLEN's 13 temporal relations

Relation Code	Relation	Figure Illustration	Relation Code	Relation	Figure Illustration
1	t_1 before t_2		8	t_2 before t_1	
2	t_1 overlaps t_2		9	t_2 overlaps t_1	
3	t_1 starts t_2		10	t_2 starts t_1	
4	t_1 finishes t_2		11	t_2 finishes t_1	
5	t_1 meets t_2		12	t_2 meets t_1	
6	t_1 contains t_2		13	t_2 contains t_1	
7	t_1 equals t_2				

In this paper, we propose a new problem of querying time-constrained graph pattern in a temporal graph. We utilize an example in Figure 1 to illustrate the problem.

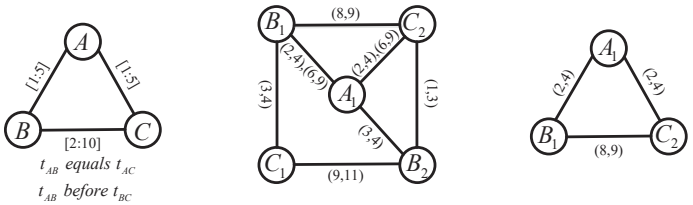


Fig. 1. An example of querying time-constrained graph pattern in a temporal graph

Example 1 : Figure 1(b) shows a biology network. A vertex represents a protein and an edge denotes protein-protein interaction . Each edge is associated with a list of time intervals denoted as $(s_1, f_1), (s_2, f_2), \ldots, (s_n, f_n)$ where (s_i, f_i) denotes that interaction starts at time s_i and finishes at time f_i. If a biologist wants to know if there exists a dynamic module[16] in Figure 1(b) and the dynamic module is described as follows: (1)The module consists of three proteins, i.e., A, B and C. (2) The interaction between A and B exists during time 1 to 5. The interaction between A and C exists during time 1 to 5. The interaction between B and C exists during time 2 to 10. (3) The time of AB equals that of AC and the time of AB should be earlier than that of BC. Figure 1(a) shows the dynamic module. The matched subgraph of Figure 1(a) is shown in Figure 1(c)(computing process is discussed in Section 4).

Querying time-constrained graph pattern in a temporal graph is not an easy work. As we all know, graph pattern matching is typically defined in terms of subgraph isomorphism which is a NP-hard problem[17]. Besides, taking time constraints into consideration slow down the process drastically. For example, there exists a graph consisting of m nodes and each edge contains n time intervals on average. There may exist $\frac{m\times(m-1)}{2}$ edges. Therefore, there can be $n^{\frac{m\times(m-1)}{2}}$ temporal subgraphs to be checked (the definition of the temporal subgraph is shown in Section 3).

In all, the contributions of this paper can be summarized as follows:

- We propose a new problem, i.e., time-constrained graph pattern matching in a large temporal graph. To the best of our knowledge, we are the first to study the problem.
- We propose a baseline algorithm to solve the problem. To improve the efficiency, an improved algorithm is proposed. Besides, two pruning rules are proposed to improve efficiency. The improved algorithm runs several orders of magnitude faster than the baseline algorithm.
- We have carried out a large number of experiments based on three real and semi-real datasets to evaluate the effectiveness and efficiency of our solutions. Experimental results show the high performance of proposed algorithms.

Organization : The rest of the paper is organized as follows. Section 2 reviews related work. In Section 3, we introduce several basic concepts and define the problem formally. Details of algorithms are described in Section 4. In Section 5, we conduct a series of experiments to evaluate the effectiveness and efficiency of proposed methods. Finally, Section 6 concludes the paper in brief.

2 Related Work

In this section, we discuss related work on temporal graphs and graph pattern matching.

Temporal Graphs: Recently, a lot of researchers investigate temporal graphs deeply. Huang et al. find minimum spanning trees in a temporal graph[18]. Yang et al. study how to find k dense temporal subgraphs in a temporal graph[19].

Various types of temporal paths are defined to study temporal graphs[20–22]. However, no work queries time-constrained subgraphs in a temporal graph.

Graph Pattern Matching: The problem of querying static graph in static graphs has two lines. One line is subgraph isomorphism which is a NP-hard problem[17]. Recently, Lee et al.[23] re-implement five state-of-the-art subgraph isomorphism algorithms and compare them based on real-life data. However, isomorphism-based graph pattern matching is too strict to find useful patterns. Besides, it is prohibitively expensive when it is applied in large scale graphs. Another line of graph pattern matching is graph simulation[24, 25]. Currently, Fan et al.[2] propose a revision of the notion of graph pattern matching, i.e., bounded graph simulation, which overcomes issues in subgraph isomorphism. However, all of work mentioned above cannot solve our problem because both query graph and data graph are static in their work.

In order to solve more practical problems, a lot of researchers investigate deeply on querying static graph in dynamic graphs. These work can be divided into two categories: transactional query and single graph query. In transactional query[26, 27], there exist a graph dataset where old graphs can be deleted and new graphs can be added. Yuan et al.[28] propose a one-pass algorithm to update graph indices. In single graph query, nodes and edges in the data graph can be deleted or added. Fan et al.[11] develop incremental algorithms to quickly find results by revising old results. However, none of these work can solve our problem because the query graphs in these work are static.

Recently, a few work focuses on querying dynamic patterns in a dynamic graph. Song et al.[12] aim to find an event pattern over graph stream. However, their work models the dynamic graph as a series of snapshots, which causes the loss of temporal information. Besides, the query pattern only considers partial order constraint on the time of edges[12]. Hence, the algorithms in their work cannot solve our problem.

3 Preliminaries

Temporal Data Graph : Given a directed labeled graph $G = (V, E, L)$, V is a set of vertices, E is a set of edges, L is a label function that assigns labels to vertices and edges. Each edge in E is in the form of (u, v, T) where $u, v \in V$ and T denotes a set of time intervals such that $(s_1, f_1), (s_2, f_2), \ldots, (s_n, f_n)$ where s_i is starting time of a time interval and f_i is finishing time of a time interval(All algorithms can be used for undirected graphs and each edge in an undirected graph can be seen as a bidirectional edge).

Time-Constrained Query Pattern : A time-constrained query graph $Q = (V_q, E_q, L_q, T_q, TS_q)$ is also a directed graph, V_q is a set of vertices, $E_q \subseteq V_q \times V_q$, L_q is a label function, T_q and TS_q denote two types of time constraints. The first type of time constraint T_q aims at limiting time for a single edge. If there exists a time constraint (e_i, l, h) in T_q, it means that starting time of e_i cannot be earlier than l and finishing time of e_i cannot be later than h. For example, a user wants to find an old friend who worked at a company around 2003-2008 years

where 2003 year is a lower bound and 2008 year is an upper bound. Another time constraint TS_q is the constraint on temporal relations between edge e_i and edge e_j. For example, a user wants to go to v_3 from v_1 through v_2 by flight. There exist hidden time constraints on temporal relationship in this query. The finishing time of the flight v_1v_2 must be earlier than the starting time of the flight v_2v_3. ALLEN[15] divides temporal relationship into thirteen types. Table 1 illustrates these thirteen widely used relationships[29]. TS_q is a matrix where $TS_q[i][j]$ is a relation code(Table1) limited for the temporal relationship between *i-th* edge and *j-th* edge.

In the following context, several basic concepts are introduced.

Definition 1 Subgraph Isomorphism*[23]: Given a query graph $Q = (V, E, L)$, a subgraph $g = (V', E', L')$ in the data graph G, a subgraph isomorphism is a bijective function f: $V -> V'$ such that:*

(1) $\forall u \in V, L(u) \subseteq L'(f(u))$.
(2) $\forall (u_i, u_j) \in E$ *if and only if* $(f(u_i), f(u_j)) \in E'$ *and* $L(u_i, u_j) = L'(f(u_i), f(u_j))$.

Definition 2 Temporal Subgraph*: Given a temporal data graph $G = (V, E, L)$, a temporal subgraph is a directed labeled graph $g = (V_t, E_t, L_t)$ where $V_t \subseteq V$, L_t is a label function and E_t is a set of edges. Each edge in E_t is in the form of (u, v, t) where $u, v \in V_t$ and t is a time interval. $\forall (u, v, t) \in E_t, \exists (u, v, T) \in E$ and $t \in T$.*

Definition 3 Temporal Subgraph Isomorphism*: Given a query graph $Q = (V_q, E_q, L_q, T_q, TS_q)$, a temporal subgraph $g = (V_t, E_t, L_t)$, a temporal subgraph isomorphism is a bijective function M: $V_q -> V_t$ such that:*

(1) $\forall u \in V_q, L_q(u) \subseteq L_t(M(u))$.
(2) $\forall (u_i, v_i) \in E_q$ *if and only if* $(M(u_i), M(v_i), t_i) \in E_t$ *and* $L_q(u_i, v_i) = L_t(M(u_i), M(v_i), t_i)$.
(3) $\forall (e_i, l_i, h_i) \in T_q(e_i = (u_i, v_i))$, $\exists (M(u_i), M(v_i), t_i) \in E_t$ *and* $t_i.s >= l_i$ *and* $t_i.f <= h_i$ *where s is starting time and f is finishing time.*
(4) $\forall (M(u_i), M(v_i), t_i), (M(u_j), M(v_j), t_j) \in E_t$, *the temporal relationship between* t_i *and* t_j *is* $TS[i][j]$.

4 Algorithms

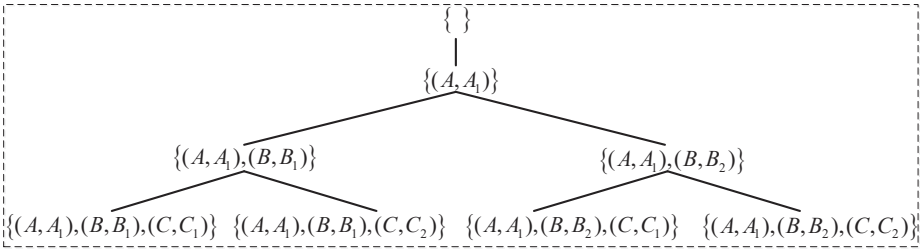
The definitions of formal problem have been proposed in previous section. This section shows details of solutions. We firstly introduce a baseline method, i.e., Time-Constrained Graph Pattern Matching based on Vertex mapping(TCGPM-V). Then, another method, i.e., Time-Constrained Graph Pattern Matching based on Edge mapping(TCGPM-E) is proposed to improve the efficiency. Besides, two pruning rules are proposed to further improve the efficiency.

4.1 *TCGPM-V* Algorithm

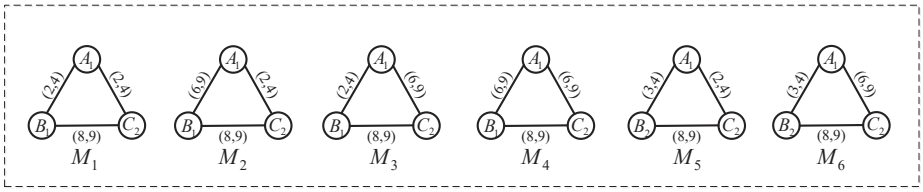
Before discussing details of $TCGPM\text{-}V$, we firstly define a complete vertex mapping set formally.

Definition 4 A Complete Vertex Mapping Set : *Given a query graph* $Q = (V_q, E_q, L_q, T_q, TS_q)$ *and a data graph* $G = (V, E, L)$*, a complete vertex mapping set* $S_v \subseteq V_q \times V$ *such that:*

(1) For any two elements $(v_i, v_i'), (v_j, v_j') \in S_v$*, if* $v_i \neq v_j$*,* $v_i' \neq v_j'$*.*
(2) $\forall v_i \in V_q$ *if and only if* $\exists (v_i, v_i') \in S_v$*.*



(a) Process of Generating Complete Vertex Mapping Sets



(b) Possible Temporal Subgraphs

Fig. 2. Process of Graph Pattern Matching based on Vertex Mapping

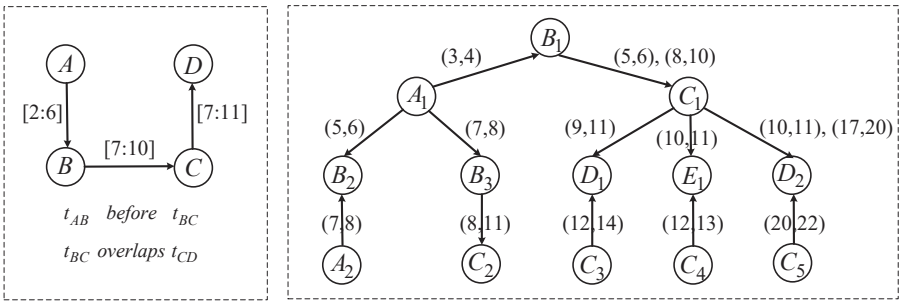
Algorithm $TCGPM\text{-}V$ can be divided into two steps. The first step, using depth-first tree search, aims to find all complete vertex mapping sets. The second step enumerates all possible temporal subgraphs and finds correct temporal subgraphs. We utilize Figure 2 to illustrate the process of $TCGPM\text{-}V$. Given a query graph(Figure 1(a)) and a data graph(Figure 1(b)), it firstly enumerates all complete vertex mapping sets, i.e., $\{(A, A_1), (B, B_1), (C, C_1)\}$, $\{(A, A_1), (B, B_1), (C, C_2)\}$, $\{(A, A_1), (B, B_2), (C, C_1)\}$ and $\{(A, A_1), (B, B_2), (C, C_2)\}$(shown in Figure 2(a)). Then, it generates all possible temporal subgraphs shown in Figure 2(b). The complete vertex mapping sets $\{(A, A_1), (B, B_1), (C, C_1)\}$ and $\{(A, A_1), (B, B_2), (C, C_1)\}$ cannot generate correct temporal subgraphs because there exists no edge between A_1 and C_1. $M_1 - M_4$ are generated by $\{(A, A_1), (B, B_1), (C, C_2)\}$. M_5 and M_6 are generated by $\{(A, A_1), (B, B_2), (C, C_2)\}$. Only M_1 meets the time constraints in Figure 1(a) and finally M_1 is returned shown in Figure 1(c). M_2, M_3, M_5 and M_6 cannot meet the constraint that t_{AB} equals t_{AC}. M_4 cannot meet the constraint that t_{AB} is before t_{BC}.

4.2 *TCGPM-E* Algorithm

In previous years, most of studies about subgraph pattern matching were based on vertex mapping. Previous work tried to reduce the number of recursive calls to improve the efficiency. According to in-depth comparison of subgraph isomorphism algorithms[23], we know that signature-pruning is very important for subgraph isomorphism. A temporal edge in a query graph has more signatures than a vertex. The signatures of a temporal edge e can be divided into three types including temporal signature, semantic signature and topology signature:

- Temporal signature is in the form of (e, l, h) where l denotes time lower bound of e and h denotes time upper bound of e.
- Semantic signature is in the form of (l_s, l_t) where l_s represents the labels of source node and l_t denotes the labels of target node.
- A temporal edge has two types of topology signatures. The first type of topology signature is the number of neighbor edges $|N_n|$ where each edge has a common node with the temporal edge. Another topology signature is the number of shared nodes $|N_s|$. A shared node is a node shared by two edges in N_n.

We propose an improved algorithm $TCGPM\text{-}E$ based on temporal edge mapping. To further improve the efficiency of improved algorithms, we decompose a large data graph into a set of small subgraphs and search the query graph in these small subgraphs. The whole process of $TCGPM\text{-}E$ is illustrated as follows: (1)Selecting an edge in the query graph according to a ranking function. (2)Finding the diameter r of the query graph centered at selected edge. (3)Finding all mapping edges M_e of selected edge e. (4)Decomposing the data graph into $|M_e|$ subgraphs. Each subgraph is centered at an edge in M_e and diameter of each subgraph is r. (5)Performing subgraph isomorphism based on edge mapping in each subgraph. (6)Enumerating matched temporal subgraphs.



(a) Query Graph **(b) Data Graph**

Fig. 3. Example of query and data graphs

(**1**) **Ranking Function** : To improve the efficiency of the algorithm, it is necessary to minimize the number of components $|M_e|$ which is associated with the selected edge in the query graph. The more mappings the selected edge has,

the more components can be obtained. Hence, it is necessary to find the edge which has less mappings. The ranking function of an edge e is defined as follows:

$$f(e) = \frac{\frac{h-l}{h_{max}-l_{min}} \times \frac{freq(l_s,l_t)}{|E_G|}}{|N_n|} \tag{1}$$

where h is the time upper bound of e, l is the time lower bound of e, $h_{max} = max(\sum_{i=1}^{|E_G|} h_i)$, $l_{min} = min(\sum_{i=1}^{|E_G|} l_i)$, $freq(l_s, l_t)$ is the number of edges in the data graph which have the same semantic signature with e, $|E_G|$ is the number of edges in data graph and $|N_n|$ is the number of adjacent edges of e. An edge with lowest value is selected. Intuitively, if the interval between l and h is narrower, then less edges may meet the temporal requirements. Besides, e has less mappings if less edges have the same semantic features with e. Finally, the more adjacent edges e has, the less mappings e may have[23].

Let's take Figure 3 as an example. The value of e_{AB} is $f(e_{AB}) = \frac{\frac{6-2}{11-2} \times \frac{4}{12}}{1} = \frac{4}{27}$. The value of e_{BC} is $f(e_{BC}) = \frac{\frac{10-7}{11-2} \times \frac{2}{12}}{2} = \frac{1}{36}$. The value of e_{CD} is $f(e_{CD}) = \frac{\frac{11-7}{11-2} \times \frac{4}{12}}{1} = \frac{4}{27}$. Finally, e_{BC} is selected.

(**2**) **Graph Diameter** : A line of a graph refers a sequence of edges where adjacent edges must have a common vertex. Given an edge e_0 as the center of a graph, a longest line starting from e_0 is $e_0, e_1, e_2, \ldots, e_{i-1}, e_i$ (if $n \neq m$, then $e_n \neq e_m$), then the diameter of a graph centered at e_0 is i.

Let's look at the query graph in Figure 3. If e_{BC} is set as the center edge, then the longest line is e_{BC}, e_{CD} or e_{BC}, e_{AB}. Hence, the diameter of the query graph is 1.

(**3**) **Edge Mappings** : Given an edge e_i in the query graph, if an edge e_j in the data graph is the mapping of e_i, it must meet all following requirements:

- $e_i.l_s = e_j.l_s$ and $e_i.l_t = e_j.l_t$.
- There exists at least one time interval (s, f) in time list of e_j where s is later than time lower bound of e_i and f is earlier than time upper bound of e_i.
- $|e_j.N_n| \geqslant |e_i.N_n|$ and $|e_j.N_s| \geqslant |e_i.N_s|$.

In Figure 3, only $e_{B_1C_1}$ is a mapping of e_{BC}.

(**4**)**Decomposition** : A component can be obtained by the following process: Given a mapping edge e_i and diameter r of a query graph, e_i is firstly put into an empty edge set S_i. Secondly, putting all adjacent edges of edges in S_i into S_i and repeating it r times. Then, a component S_i is obtained. Repeating the process $|M_e|$ times and $|M_e|$ components are obtained.

For example, given a mapping edge $e_{B_1C_1}$ and diameter 1, we finally obtain the component $\{e_{B_1C_1}, e_{A_1B_1}, e_{C_1D_1}, e_{C_1E_1}, e_{C_1D_2}\}$.

(**5**) **Subgraph Isomorphism Based on Edge Mapping** : Edge mapping is used in the improved algorithm. We firstly define a complete edge mapping set.

Definition 5 A Complete Edge Mapping Set : *Given a query graph* $Q = (V_q, E_q, L_q, T_q, TS_q)$ *and a data graph* $G = (V, E, L)$*, a complete edge mapping set* $S_e \subseteq E_q \times E$ *must meet the following requirements:*

(1) For any two elements $(e_i, e'_i), (e_j, e'_j) \in S_e$*, if* $e_i \neq e_j$*, then* $e'_i \neq e'_j$*.*
(2) $\forall e_i \in E_q$ *if and only if* $\exists (e_i, e'_i) \in S_e$*.*

Algorithm 1: *RecursiveMapE*

Data: a query patten $Q = (V_q, E_q, L_q, T_q, TS_q)$, a data graph $G = (V, E, L)$ and a partial edge mapping set S_e
Result: a set of complete edge mapping sets

```
1  if S_e is a complete edge mapping set then
2  |  Output(S_e);
3  else
4  |  e= NextUnmatchedEdge(Q, S_e);
5  |  compute a set of mappings E for e;
6  |  for each edge e' in E do
7  |  |  put (e, e') into S_e;
8  |  |  if IsFeasible(S_e, e', e) then
9  |  |  |  RecursiveMapE(Q, G, S_e);
10 |  |  remove (e, e') from S_e;
```

Algorithm 1 shows the pseudo-code of subgraph isomorphism based on edge mapping. Firstly, it checks if S_e is a complete edge mapping set(line 1). If S_e is a complete edge mapping set, S_e is returned(line 2). Otherwise, it needs to extend the partial edge mapping set(lines 4-10). Function $NextUnmatchedEdge$ finds a query edge which is not in S_e(line 4). $NextUnmatchedEdge$ has two rules for finding next edge: (1) The next edge must be an edge connected with a mapped edge which is proved to be efficient[4]. (2) It picks up an edge in query graph according to the ranking function. In line 5, E is a set of mapping edges for e. It extends S_e by adding each possible mapping pairs (e, e')(line 7). After adding a mapping pair, it is necessary to check if a partial edge mapping set is feasible(line 8). If S_e is feasible, it invokes $RecursiveMapE$ to extend S_e until S_e becomes a complete edge mapping set(line 9). The function $IsFeasible$ contains four rules for each added edge mapping pair (e, e') such that: (1) $\forall (e_j, e'_j) \in S_e, e_j.v_t = e.v_s \iff e'_j.v_t = e'.v_s$. (2) $\forall (e_j, e'_j) \in S_e, e_j.v_t = e.v_t \iff e'_j.v_t = e'.v_t$. (3) $\forall (e_j, e'_j) \in S_e, e_j.v_s = e.v_s \iff e'_j.v_s = e'.v_s$. (4) $\forall (e_j, e'_j) \in S_e, e_j.v_s = e.v_t \iff e'_j.v_s = e'.v_t$ where v_s is source node and v_t is target node.

(**6**) **Pruning Rules** : After obtaining all complete edge mapping sets, it needs to enumerate all possible temporal subgraphs. The time complexity of enumeration is $|T_a|^{|E_q|} * |R|$ where $|E_q|$ is the number of edges in the query pattern, $|T_a|$ is the average number of time intervals of an edge and $|R|$ is the number of complete edge mapping sets. There are two ways of improving the efficiency. The first way is reducing $|R|$ and the second way is reducing $|T_a|$($|E_q|$ is decided by users).

Pruning 1: Given a query pattern $Q = (V_q, E_q, L_q, T_q, TS_q)$ and a complete edge mapping set S_e, $\forall(e_i, e'_i), (e_j, e'_j) \in S_e$, if there are not two time intervals $t_{e'_i}$ and $t_{e'_j}$ that the temporal relation between $t_{e'_i}$ and $t_{e'_j}$ is $TS_q[i][j]$, then S_e can be eliminated. For example, in Figure 3, $\{(e_{AB}, e_{A_1B_1}), (e_{BC}, e_{B_1C_1}), (e_{CD}, e_{C_1D_2})\}$ is a complete edge mapping set. However, there are not two time intervals $t_{B_1C_1}$ and $t_{C_1D_2}$ that $t_{B_1C_1}$ overlaps $t_{C_1D_2}$. Hence, this complete edge mapping set can be eliminated.

Pruning 2: Given a query pattern $Q = (V_q, E_q, L_q, T_q, TS_q)$ and a complete edge mapping set S_e, $\forall(e_i, e'_i), (e_j, e'_j) \in S_e$, if there exists no time interval $t_{e'_j}$ that $t_{e'_i}$ has $TS_q[i][j]$ temporal relation with $t_{e'_j}$, then $t_{e'_i}$ can be eliminated. For example, in Figure 3, $\{(e_{AB}, e_{A_1B_1}), (e_{BC}, e_{B_1C_1}), (e_{CD}, e_{C_1D_1})\}$ is a complete edge mapping set. The time interval $(5, 6)$ of $e_{B_1C_1}$ can be eliminated because there exists no time interval $t_{C_1D_1}$ that $(5, 6)$ overlaps $t_{C_1D_1}$.

Algorithm 2: *TCGPM–E*

Data: a query patten $Q = (V_q, E_q, L_q, T_q, TS_q)$ and a data graph $G = (V, E, L)$
Result: a set of temporal subgraphs

```
1  select a query edge e with lowest ranking value;
2  compute the query graph diameter r centered at e and the mapping set M_e of e;
3  for each e_i ∈ M_e do
4      g = GenComponent(e_i, r, G);
5      if the size of g is larger than Q then
6          S_e = ∅;
7          put (e, e_i) into S_e;
8          R_g = RecursiveMapE(Q, g, S_e);
9          reduce the number of complete edge mapping sets |R_g| by pruning 1;
10         for each complete edge mapping set S_e ∈ R_g do
11             reduce average time intervals T_a by pruning 2;
12             while exist new temporal subgraphs E_t do
13                 for each edge e_q ∈ E_q, (e_q, e'_q) ∈ S_e do
14                     select a time interval t_{e'_q} from the time intervals list of e'_q;
15                     put (e'_q, t_{e'_q}) into E_t;
16                 if E_t satisfies all time constraints then
17                     Output(E_t);
```

Algorithm 2 shows the pseudo-code of *TCGPM-E*. At first, it selects an edge e in query graph according to the ranking function(line 1). Then, it computes the diameter r of query graph centered at selected edge e by BFS(breadth-first-search) and mappings of e(line 2). For each mapping edge e_i of e, it extracts a component g from the data graph G by BFS(line 4). Then, the first mapping pair (e, e_i) is put into the partial edge mapping set S_e(lines 6-7). All complete edge mapping sets R_g in the subgraph g are computed by invoking function *RecursiveMapE*(line 8). *Pruning*1 is used to reduce the number of complete

edge mapping sets(line 9). Lines 10-17 show the process of generating all correct temporal subgraphs. Lines 13-15 enumerate all possible temporal subgraphs w.r.t. a complete edge mapping set S_e. If a temporal subgraph E_t meets all time constraints, E_t is returned(lines 16-17).

5 Experiments

In this section, we experimentally evaluate effectiveness and efficiency of proposed algorithms. All algorithms are implemented using java in a Linux machine with 96 CPU, 2.60GHz, 1T RAM. We use two real datasets named Contact[1], SEQ[2] and a semi-real dataset named Patents[3]. Patents dataset does not have temporal information. In order to keep the real temporal distribution, we transfer the temporal information of dataset SEQ to dataset Patents. All results are average value based on three runs.

Table 2. Datasets

Dataset	Contact	SEQ	Patents
Num. of vertices	327	12,654	3,774,768
Num. of static edges	5,818	15,522	16,518,948
Num. of temporal edges	188,508	701,585	746,556,059
Avg. time intervals	32	45	45
Num. of labels	9	23	421

5.1 Effectiveness

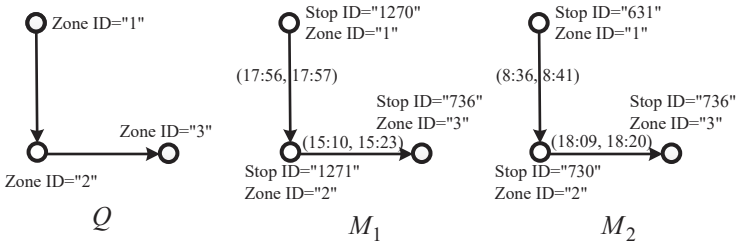


Fig. 4. Illustration of Useless Matched Temporal Subgraphs

At first, we evaluate the effectiveness of Time-Constrained Graph Pattern Matching(TCGPM). Let's look at the Figure 4. If a lady works at a company in zone one and she needs to have dinner with a friend in zone two after work. After dinner, she has to go home in zone three. Hence, she queries a traffic patten shown in Q. Both M_1 and M_2 are returned by traditional solutions in

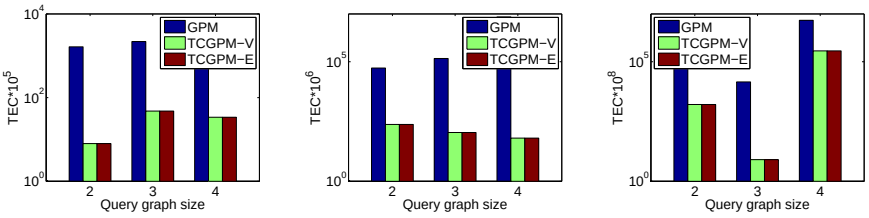
[1] http://www.sociopatterns.org/datasets
[2] https://gtfsrt.api.translink.com.au
[3] https://snap.stanford.edu/data/cit-Patents.html

GPM(Graph Pattern Matching)[23]. M_1 is not a correct result because the time of edge (1270,1271) is later than that of edge (1271, 736). When the lady arrives at stop 1271, she is too late to go to stop 736. M_2 is not a good choice for the lady because she could not be off duty at 8:36 a.m. When constraints on temporal relation between two edges are considered, M_1 is not returned by $TCGPM$. When constraints on the time of a single edge can be considered, M_2 is not returned by $TCGPM$. Hence, $TCGPM$ can filter out a lot of useless subgraph so that users can inspect results and find what they really want quickly. To quantify effectiveness of solutions, we define the following performance metric:

Temporal Edge Coverage(**TEC**) : A temporal edge is an edge attached with one time interval. Temporal edge coverage not only considers the number of temporal edges in returned temporal subgraphs, it also considers frequency of each temporal edge in returned subgraphs. Given a data graph $G = \{V_G, E_G, L_G\}$ and a set of returned temporal subgraphs $\{M_1, M_2, \ldots, M_n\}$, TEC is defined as:

$$TEC = \sum_{j=1}^{n} |E_{M_j}| \; / \; \sum_{i=1}^{|E_G|} |e_i.T| \tag{2}$$

where $|E_{M_j}|$ is the number of temporal edges in M_j and $|e_i.T|$ is the number of time intervals of edge e_i in the data graph. The challenge is to keep TEC as low as possible.



(a) Contact (b) SEQ (c) Patents

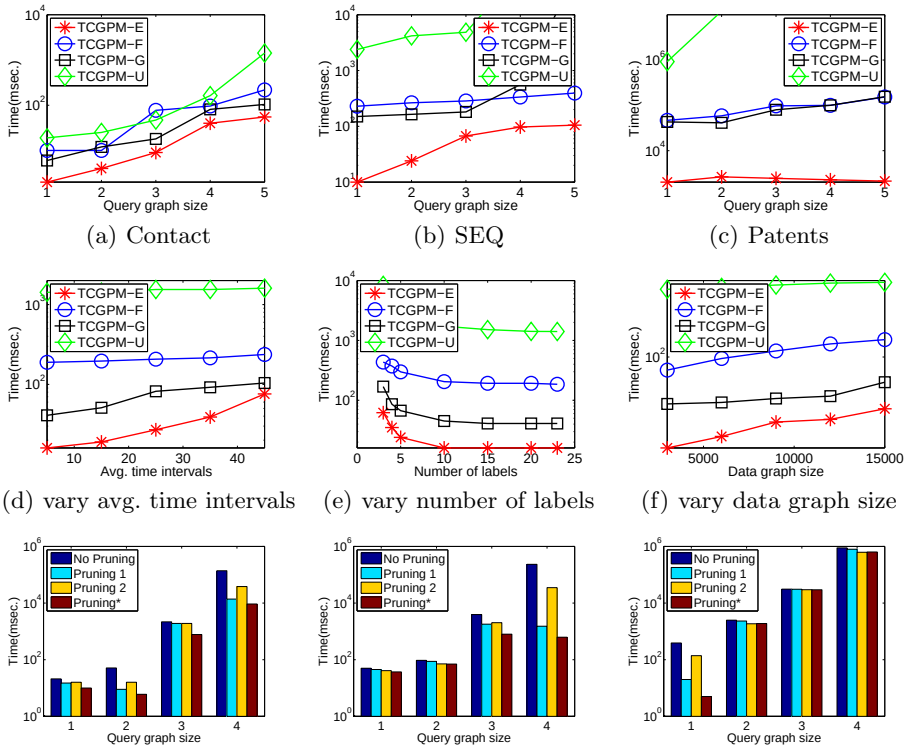
Fig. 5. TEC Evaluation

Figure 5 shows TEC of traditional solution GPM[23] and proposed solutions $TCGPM\text{-}V$, $TCGPM\text{-}E$ based on datasets Contact, SEQ and Patents. We vary the query graph size from 2 to 4. As we can see, TEC of GPM is several orders of magnitude larger than that of $TCGPM\text{-}E$ and $TCGPM\text{-}V$. A lot of irrelevant temporal edges are included because GPM does not consider the temporal constraints. TEC of $TCGPM\text{-}E$ is the same with that of $TCGPM\text{-}V$ because $TCGPM\text{-}E$ offer users the same results with $TCGPM\text{-}V$.

5.2 Efficiency

In this subsection, we mainly evaluate (1) efficiency of proposed algorithm by comparing it with previous algorithms; (2) the effect of parameters on efficiency; (3) efficiency of proposed pruning rules.

A lot of traditional algorithms, i.e., Ullman[30], VF2[4] and GraphQL[5], can be used to discover all complete vertex mapping sets. Hence, we re-implement three traditional algorithms in the baseline method and compare them with the improved method $TCGPM\text{-}E$. VF2 is used in $TCGPM\text{-}F$, GraphQL is used in $TCGPM\text{-}G$ and Ullman is used in $TCGPM\text{-}U$.



(a) Contact (b) SEQ (c) Patents

(d) vary avg. time intervals (e) vary number of labels (f) vary data graph size

(g) Contact (h) SEQ (i) Patents

Fig. 6. Efficiency Evaluation of Proposed Algorithms

Figures 6(a), 6(b), 6(c) show the performance in Contact, SEQ, Patents datasets respectively. The proposed method $TCGPM\text{-}E$ runs several orders of magnitude faster than other algorithms. Then, we analyze the effect of several parameters, i.e., average time intervals of an edge, number of labels and data graph size(the number of edges in the data graph). Figure 6(d) shows that when average time intervals of an edge becomes larger, the runtime of algorithms increases because the time complexity of generating all temporal subgraphs is $|T_a|^{|E_q|} * |R|$ where $|T_a|$ is average time intervals. Figure 6(e) depicts that increasing number of labels leads to decreasing of runtime because when the data graph size is fixed, the number of labels is larger, there exist less mappings. Figure 6(f) demonstrates that the larger the data graph size is, the longer the runtime is.

Figures 6(g), 6(h) and 6(i) illustrate the efficiency of pruning rules. $Pruning*$ denotes that both $Pruning1$ and $Pruning2$ are used. We can see clearly that

both $Pruning1$ and $Pruning2$ can save much runtime. Besides, when two pruning rules are used simultaneously, the performance is better. In Figure 6(i), the performance of pruning rules is not obvious due to semi-log coordinates system. When query graph size is four, $Pruning*$ can save about 30 percent of runtime.

6 Conclusion

In this paper, we propose the new problem, time-constrained graph pattern matching in a temporal graph. We propose two algorithms, $TCGPM$-V and $TCGPM$-E. Besides, we design two pruning rules. Extensive experiments demonstrate the high performance of proposed solutions. Due to large volumes of temporal graph data, we will study disk representation in the future work.

Acknowledgments. This work was supported by the National Natural Science Foundation of China under Grant Nos. 61572335 and 61572336, the Natural Science Foundation of Jiangsu Province of China under Grant No. BK20151223, the Natural Science Foundation of Jiangsu Provincial Department of Education of China under Grant No. 12KJB520017, and Collaborative Innovation Center of Novel Software Technology and Industrialization, Jiangsu, China.

References

1. Bruno, N., Koudas, N., Srivastava, D.,: Holistic twig joins: optimal XML pattern matching. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 310–321. ACM, Wisconsin(2002)
2. Fan, W., Li, J., Ma, S., Tang, N., Wu, Y., Wu, Y.:Graph Pattern Matching: From Intractable to Polynomial Time. Proceedings of Vldb Endowment 3(1), 264–275 (2010)
3. Tian, Y., McEachin, R.C., Santos, C., States, D.J., Patel, J.M.:SAGA: a subgraph matching tool for biological graphs. Bioinformatics 23(2), 232–239(2007)
4. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence 26(10), 1367–1372(2004)
5. He, H., Singh, A.K.: Graphs-at-a-time: query language and access methods for graph databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 405-418. ACM, Canada(2008)
6. Shang, H., Zhang, Y., Lin, X., Yu, J.X.: Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. Proceedings of Vldb Endowment 1(1), 364–375(2008)
7. Zhao, P., Han, J.: On graph query optimization in large networks. Proceedings of Vldb Endowment 3(3), 340–351(2010)
8. Han, W., Lee, J., Lee, J.: $\text{Turbo}_{\text{iso}}$: towards ultrafast and robust subgraph isomorphism search in large graph databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 337-348. ACM, New York(2013)
9. Fan, W., Wang, X., Wu, Y., Deng, D.: Distributed graph simulation: Impossibility and possibility. Proceedings of Vldb Endowment 7(12), 1083–1094(2014)

10. Chen, L., Wang, C.: Continuous subgraph pattern search over certain and uncertain graph streams. IEEE Trans. Knowl. Data Eng. 22(8), 1093–1109(2010)
11. Fan, W., Wang, X., Wu, Y.: Incremental graph pattern matching. ACM Trans. Database Syst. 38(3), 18–118(2013)
12. Song, C., Ge, T., Chen, C.X., Wang, J.: Event pattern matching over graph streams. Proceedings of Vldb Endowment 8(4), 413–424(2014)
13. Khurana, U., Deshpande, A.: Storing and analyzing historical graph data at scale. In: Proceedings of the 19th International Conference on Extending Database Technology, pp. 65–76. France(2016)
14. Fan, W., Wang, X., Wu, Y.: Diversified top-k graph pattern matching. Proceedings of Vldb Endowment 6(13), 1510–1521(2013)
15. Allen, J., Allen, J.: Maintaining knowledge about temporal intervals. Commun. 26(11), 832–843(1983)
16. Jin, R., McCallen, S., Liu, C., Xiang, Y., Almaas, E., Zhou, X.J.: Identifying dynamic network modules with temporal and spatial constraints. In: Proceedings of the Pacific Symposium, pp. 203–214 USA(2009)
17. Shamir, R., Tsur, D.:Faster subtree isomorphism. J. Algorithms. 33, 267–280(1999)
18. Huang, S., Fu, A.W., Liu, R.: Minimum spanning trees in temporal graphs. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 419–430 Melbourne(2015)
19. Yang, Y., Yan, D., Wu, H., Cheng, J., Zhou, S., Lui, J.C.S.: Diversified temporal subgraph pattern mining. In: Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining, pp. 1965–1974 San Francisco(2016)
20. Wu, H., Huang, Y., Cheng, J., Li, J., Ke, Y.: Reachability and time-based path queries in temporal graphs. In: 32nd IEEE International Conference on Data Engineering, pp. 145–156 Helsinki(2016)
21. Wu, H., Cheng, J., Huang, S., Ke, Y., Lu, Y., Xu, Y.: Path problems in temporal graphs. Proceedings of Vldb Endowment 7(9), 721–732(2014)
22. Kossinets, G., Kleinberg, J.M., Watts, D.J.: The structure of information pathways in a social communication network. In: Proceedings of the 14th International Conference on Knowledge Discovery and Data Mining, pp. 435–443 Las Vegas(2008)
23. Lee, J., Han, W., Kasperovics, R., Lee, J.: An in-depth comparison of subgraph isomorphism algorithms in graph databases. Proceedings of Vldb Endowment 6(2), 133–144(2012)
24. Henzinger, M.R., Henzinger, T.A., Kopke, P.W.: Computing simulations on finite and infinite graphs. In: 36th Annual Symposium on Foundations of Computer Science, pp. 453–462 Milwaukee(1995)
25. Ma, S., Cao, Y., Fan, W., Huai, J., Wo, T.: Strong simulation: Capturing topology in graph pattern matching. Trans. Database Syst. 39(1), 4–4(2014)
26. Yuan, D., Mitra, P., Yu, H., Giles, C.L.: Iterative graph feature mining for graph indexing. In: 28th International Conference on Data Engineering, pp. 198–209 USA(2012)
27. Chen, C., Yan, X., Yu, P.S., Han, J., Zhang, D., Gu, X.: Towards graph containment search and indexing. In: Proceedings of the 33rd International Conference on Very Large Data Bases, pp. 926–937 Austria(2007)
28. Yuan, D., Mitra, P., Yu, H., Giles, C.L.: Updating graph indices with a one-pass algorithm. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 1903–1916, Melbourne(2015)
29. Chen, Y., Peng, W., Lee, S.: Mining temporal patterns in time interval-based data. IEEE Trans. Knowl. Data Eng. 27(12), 3318–3331(2015)
30. Ullmann, J.R.: An algorithm for subgraph isomorphism. J. ACM. 23, 31–42(1976)