APPENDIX A

***Proof*. (Proof of Lemma 4)** *If V_S is a MalKLP, then any vertex in $V\backslash V_S$ cannot be added to V_S to form a (k,l)-plex. The vertices in $V\backslash V_S$ can be divided into two categories: one category is the unprocessed vertices, and the other category is the processed vertices. The unprocessed vertices that have the potential to form a (k,l)-plex with V_S will be stored in C_V. Therefore, if there is a vertex in C_V that satisfies the condition, V_S is not maximal. The previously processed vertices that are contained in at least one (k,l)-plex are stored in X_V. Thus, if there exists a vertex in X_V that can form a (k,l)-plex with the current processing vertex set V_S, V_S is the subset of one found result and V_S is not maximal. The lemma holds.*

***Proof*. (Proof of Theorem 1)** *In Algorithm 1, the procedure* EnumKLP *will be executed for every subset of $\mathcal{V}$, and the total number is $2^{|\mathcal{V}|}$ times. The main cost of* EnumKLP *is the procedure* CheckKLP. *In* CheckKLP, *we need to verify whether each vertex in V_S has sufficient neighbors in V_S at each timestamp t, which costs $O(|\mathcal{T}|\delta^4_{max})$ since $|V_S|$ is bounded by δ^2_{max} according to Lemma 3. Therefore, the whole running time of Algorithm 1 is $O((|\mathcal{T}|\delta^4_{max})2^{|\mathcal{V}|})$.*

***Proof*. (Proof of Lemma 6)** *Note that, an edge (u,v) contained in less than $\theta-2k$ triangles of a snapshot G_t means that u and v have less than $\theta-2k$ common momentary neighbors at G_t. Suppose to the contrary that u and v belong to a k-plex P at G_t with $|P|\geq\theta$. Let N_{uv} be the common momentary neighbor set of u and v in P with $|N_{uv}|<\theta-2k$. Let $nonN_u$ and $nonN_v$ denote the disconnected vertex set (including itself) of u and v in P, respectively. Then, we can obtain that $|nonN_u|+|nonN_v|\geq|P\backslash N_{uv}|=|P|-|N_{uv}|>2k$. Based on the definition of k-plex, u or v must disconnect no more than k vertices (including itself) in P. If $|nonN_u|\leq k$, then $|nonN_v|\geq|P\backslash N_{uv}|-k$. We can obtain that $|nonN_v|\geq|P\backslash N_{uv}|-k>k$, which violates the definition of k-plex. Hence, removing an edge within less than $\theta-2k$ triangles from a snapshot will not change the k-plexes in this snapshot, and the lemma holds.*

***Proof*. (Proof of Lemma 7)** *Since we prune the search branches for the vertex in $V_A\backslash\{v_i\}$, we only need to prove that MalKLPs originally obtained in these skipped branches still can be found in the remaining search branches. We claim that any MalKLP obtained in the branches for the vertex in $V_A\backslash\{v_i\}$ must contain at least one vertex in $C_V\backslash V_A$. This can be proven by contradiction. Suppose there is a MalKLP V'_S obtained in the branch for $v_a\in V_A\backslash\{v_i\}$, it excludes any vertex in $C_V\backslash V_A$. This means that all the vertices included in V'_S belong to V^S_A, i.e., $V'_S\subseteq V^S_A$. This contradicts the condition that V'_S is maximal. Therefore, for each MalKLP obtained in the branches for the vertex in $V_A\backslash\{v_i\}$, it can be obtained by the branch of the vertex in $C_V\backslash V_A$ in the remaining search.*

APPENDIX B

Graph Reduction. By combining Lemmas 5 and 6, we present our Graph Reduction method in Algorithm 5. For each vertex u, we use $f[u]$ to store the number of timestamps at which u has momentary neighbors in lines 2-4. Then, we process each vertex $u\in\mathcal{V}$ at each timestamp $t\in\mathcal{T}$ in lines 5-8. If $0<\delta(u,t)<\theta-k$ or $0<f[u]<l$, we invoke procedure VertexPrune to remove vertex u from G_t, whose details are shown in lines 17-28. Specifically, we set $\delta(u,t)=0$ in line 18. Then we iteratively process u's momentary neighbors at t in lines 19-22. For u's momentary neighbor v, if it has the momentary neighbor at t, we decrease $\delta(v,t)$ by 1. If $\delta(v,t)<\theta-k$, we invoke VertexPrune to remove v at t. We decrease $f[u]$ by 1 if $f[u]>0$ in lines 23-24. In lines 25-28, if $f[u]<l$, we set $f[u]$ as 0 and remove it from all snapshots. In lines 9-10, for each vertex $u\in\mathcal{V}$, if $\delta(u,t)=0$, we remove it from the corresponding snapshot. For each edge $(u,v)\in E_t$, if it exists in less than $\theta-2k$ triangles, we remove it from the current snapshot. The time complexity of this algorithm can be bounded by $O(|\mathcal{E}|^{1.5})$.

Algorithm 5: Graph Reduction$(\mathcal{G},k,l,\theta)$

```
   Input  : G: a temporal graph, k, l and θ: positive integers
   Output : A reduced temporal graph
1  for each u ∈ V do f[u] = 0;
2  for each t ∈ T do
3      for each u ∈ V do
4          if δ(u,t) > 0 then f[u]++;
5  for each t ∈ T do
6      for each u ∈ V do
7          if 0 < δ(u,t) < θ − k ∨ 0 < f[u] < l then
8              VertexPrune(u,t);
9  for each t ∈ T do
10     for each u ∈ V do
11         if δ(u,t) = 0 then G_t ← G_t\{u};
12 for each t ∈ T do
13     Compute the triangle count △(·,·) for all edges in G_t;
14     while ∃ (u,v) ∈ E_t s.t △(u,v) < θ − 2k do
15         G_t ← G_t\{(u,v)};
16 return {G_1, G_2, ..., G_|T|};
17 Procedure VertexPrune(u,t)
18 δ(u,t) = 0;
19 for each v ∈ Γ(u,t) do
20     if δ(v,t) > 0 then
21         δ(v,t) − −;
22         if δ(v,t) < θ − k then VertexPrune(v,t);
23 if f[u] > 0 then
24     f[u] − −;
25     if f[u] < l then
26         f[u] = 0;
27         for each t ∈ T do
28             if δ(u,t) > 0 then VertexPrune(u,t);
```

IB-pruning. In Algorithm 6, by integrating the proposed techniques, we introduce the invalid branch pruning approach (IB-pruning). I is the proposed array structure that counts the number of disconnected vertices in V_S for all the vertices in $V_S\cup C_V$ at each timestamp. In the output, *i*) 0/1 denotes if $V_S\cup\{v\}$ is a (k,l)-plex or not, *ii*) C_V is the shrunk candidate set, and *iii*) $\mathcal{T}_{V_S}$ is the valid timestamp set for $V_S\cup\{v\}$. At first, if $|V_S|<\theta$, we only maintain the one-hop and two-hop structural neighbors of v in C_V according to Lemma 3 (lines 1-2). Algorithm 6 mainly consists of the following three parts. *i*) Check (k,l)-plex & compute valid timestamps (lines 3-7). In lines 4-5, we calculate the valid timestamp set for $V_S\cup\{v\}$ by checking the number of disconnected vertices

for v at timestamp t. If $I[v][t] > k-1$, it means that v disconnects more than $k-1$ vertices in V_S, and we remove t from $\mathcal{T}_{V_S}$. In lines 6-7, we calculate the valid timestamp set for $V_S \cup \{v\}$ by processing each vertex in V_S. If there exists a vertex $u \in V_S$ that disconnects v at timestamp t and $I[u][t] > k-2$, we remove t from $\mathcal{T}_{V_S}$. After processing all the timestamps in $\mathcal{T}_{V_S}$ following the above operations, we set $\mathcal{T}_{V'_S}$ as the updated $\mathcal{T}_{V_S}$ to store the valid timestamp set for $V_S \cup \{v\}$ in line 8. If $|\mathcal{T}_{V'_S}| < l$, which means that $V_S \cup \{v\}$ is not a (k,l)-plex, we return $(0, \emptyset, \emptyset)$. *ii*) Condition I (lines 10-11). For each timestamp $t \in \mathcal{T}_{V'_S}$, we access each vertex $u \in C_V$ that disconnects v at timestamp t, and increment value $I[u][t]$. *iii*) Condition II (lines 12-19). We initialize $count[u]$ for all the vertices in C_V as 0, and process each timestamp $t \in \mathcal{T}_{V'_S}$ iteratively. We traverse each vertex $u \in V_S$ that disconnects v at timestamp t, and increment $I[u][t]$ by 1. If $I[u][t] = k-1$, we continue processing u's disconnected vertices in C_V. In line 18, if $I[w][t] \leq k-1$ and $mark[w] = 1$, it means that w cannot be pruned based on Condition I and w is examined by Condition II. Then we set $mark[w]$ as 0 and increment $count[w]$ by 1. *iv*) Shrink candidate set (lines 20-22). After processing all the timestamps in $\mathcal{T}_{V'_S}$, we can shrink the candidate set. For each vertex $u \in C_V$, $|\{t \in \mathcal{T}_{V'_S} | I[u][t] > k-1\}| + count[u]$ is the number of timestamps when u cannot be added into $V_S \cup \{v\}$. If this value is larger than $|\mathcal{T}_{V'_S}| - l$, we remove u from C_V. Finally, we return $\{1, C_V, \mathcal{T}_{V'_S}\}$ as the result.

***Theorem* 5** *Algorithm 6 needs $O(|\mathcal{T}|\delta^4_{max})$ time, where δ_{max} denotes the largest momentary degree of the vertex in $\mathcal{G}$.*

***Proof*. (Proof of Theorem 5)** *In line 2, we will retrieve the one-hop and two-hop neighbors of v from C_V when $|V_S| < \theta$, which costs $O(\delta^2_{max})$ time. The computation of valid timestamps in lines 3-7 takes $O(|\mathcal{T}||V_S|)$ time, where $|V_S|$ is bounded by δ^2_{max}. Since C_V is the one-hop and two-hop neighbors of one vertex, the time cost of Condition I in lines 10-11 is $O(|\mathcal{T}|\delta^2_{max})$. In lines 13-19, the Condition II takes $O(|\mathcal{T}|\delta^4_{max})$ time. Finally, we shrink the candidate set by Conditions I and II in lines 20-22 which consumes $O(|\mathcal{T}|\delta^2_{max})$ time. Hence, the time complexity of Algorithm 6 is $O(|\mathcal{T}|\delta^4_{max})$.*

KLPE+. By combining all the pruning rules and optimized strategies, we present our KLPE+ algorithm, as detailed in Algorithm 7. First, we apply the graph reduction technique (Algorithm 5) to shrink the given temporal graph, and initialize C_V with $\mathcal{V}$, X_V with $\emptyset$ and $\mathcal{R}$ with $\emptyset$ in lines 1-2. In this paper, we access vertices based on their structural degree (line 3). Then, for each vertex $v \in C_V$, we initialize $I[v][t]$ as 0 to denote the number of v's disconnected vertices in V_S at each timestamp $t \in \mathcal{T}$ (lines 4-5). We initialize V^S_A with $\emptyset$ to store the MalKLP including the attack vertex set and StateA with 0 to denote the existence of the attack vertex set in line 6. Note that, V^S_A and StateA are global variables used in the traversal. We iterate over each $v \in \mathcal{V}$ in lines 7-15. In line 8, we store v's one-hop and two-hop structural neighbors in C'_V according to Lemma 3. In lines 9-10, for the vertex $u \in C'_V$ that disconnects v at the timestamp t, we increment

Algorithm 6: IB-pruning$(V_S, C_V, \mathcal{T}_{V_S}, v, I)$

Input : V_S: the current result, C_V: the candidate set, $\mathcal{T}_{V_S}$: the valid timestamp set for V_S, v: the processing vertex that is added to V_S from C_V, I: the array structure for all the vertices in $V_S \cup C_V$
Output : $\{0/1, C_V, \mathcal{T}_{V'_S}\}$

```
1  if |V_S| < θ then
2      C_V ← v's one-hop and two-hop structural neighbors in C_V;
   // Check (k,l)-plex & compute valid timestamps
3  for each t ∈ T_{V_S} do
4      if I[v][t] > k − 1 then
5          T_{V_S} ← T_{V_S}\{t}; continue;
6      if ∃ u ∈ V_S\Γ(v,t) ∧ I[u][t] + 1 > k − 1 then
7          T_{V_S} ← T_{V_S}\{t};
8  T_{V'_S} ← T_{V_S};
9  if |T_{V'_S}| < l then return {0, ∅, ∅};
   // Condition I
10 for each t ∈ T_{V'_S} do
11     for each u ∈ C_V\Γ(v,t) do I[u][t]++;
   // Condition II
12 for each u ∈ C_V do count[u] = 0;
13 for each t ∈ T_{V'_S} do
14     for each u ∈ C_V do mark[u] = 1;
15     for each u ∈ V_S\Γ(v,t) do
16         if ++I[u][t] = k − 1 then
17             for each w ∈ C_V\Γ(u,t) do
18                 if I[w][t] ≤ k − 1 ∧ mark[w] = 1 then
19                     mark[w] = 0; count[w]++;
   // Shrink candidate set
20 for each u ∈ C_V do
21     if |{t ∈ T_{V'_S} | I[u][t] > k − 1}| + count[u] > |T_{V'_S}| − l then
22         C_V ← C_V\{u};
23 return {1, C_V, T_{V'_S}};
```

$I[u][t]$ by 1. Then, we invoke the procedure OptEnumKLP to enumerate all the MalKLPs in line 11. The returned result of procedure OptEnumKLP is stored in Flag. If Flag = 1, which means that we can find a large (k,l)-plex, we store v into X_V. In lines 13-14, $I[u][t]$ is restored by decreasing 1 for each disconnected vertex u of v to be utilized in the next iteration. In line 15, we restore V^S_A and StateA.

The details of OptEnumKLP are shown in lines 16-39. We initialize FlagMal and FlagPlex as 0 to denote the existence of (k,l)-plex and the large (k,l)-plex, respectively. Then, we process each vertex in C_V iteratively in lines 19-33. If $V^S_A \subseteq V_S \cup C_V \wedge v \in V^S_A \wedge$ StateA = 1, which means that v exists in the obtained attack vertex set $V^S_A \backslash V_S$ and the current processing vertex set is the ancestor node of the one finding V^S_A in the recursive tree, we skip the current search branch (lines 20-21). In line 23, we obtain the shrunk candidate set by applying Algorithm 6. Remark that, State returned by Algorithm 6 denotes if $V_S \cup \{v\}$ is a (k,l)-plex and $\mathcal{T}_{V'_S}$ denotes the valid timestamp set for $V_S \cup \{v\}$. If $V_S \cup \{v\}$ is a (k,l)-plex, we invoke the procedure OptEnumKLP to process the updated sets. If Flag = 1, which means that we can find a large (k,l)-plex in the search branch, we add v into X'_V and set FlagPlex as 1 in lines 27-28. In lines 29-33, we restore I for v's disconnected vertices in C_V and V_S.

If FlagMal=0, which means that we cannot get a (k,l)-

Algorithm 7: KLPE+ ($\mathcal{G}$, k, l, θ)

Input : $\mathcal{G}$: a temporal graph, k, l and θ: positive integers
Output : $\mathcal{R}$: all the large maximal (k,l)-plexes

1 $\mathcal{G} \leftarrow$ GRAPH REDUCTION($\mathcal{G}, k, l, \theta$); /* Algorithm 5 */
2 $C_V \leftarrow \mathcal{V}$, $X_V \leftarrow \emptyset$, $\mathcal{R} \leftarrow \emptyset$;
3 Reassign vertex id of $\mathcal{V}$ in ascending order of the structural degree;
4 **for each** $v \in C_V$ **do**
5 **for each** $t \in \mathcal{T}$ **do** $I[v][t] = 0$; /* Initialize I */
6 $V_A^S \leftarrow \emptyset$, STATEA $\leftarrow 0$;
7 **for each** $v \in \mathcal{V}$ **do**
8 $C'_V \leftarrow v$'s one-hop and two-hop structural neighbors in $C_V \backslash \{v\}$;
9 **for each** $t \in \mathcal{T}$ **do**
10 **for each** $u \in C'_V \backslash \Gamma(v,t)$ **do** $I[u][t]$++;
11 FLAG $\leftarrow$ OPTENUMKLP($\{v\}, C'_V, X_V, \mathcal{T}$);
12 **if** FLAG = 1 **then** $X_V \leftarrow X_V \cup \{v\}$;
13 **for each** $t \in \mathcal{T}$ **do**
14 **for each** $u \in C'_V \backslash \Gamma(v,t)$ **do** $I[u][t]$ − −;
15 $V_A^S \leftarrow \emptyset$, STATEA $\leftarrow 0$;
16 **Procedure** OPTENUMKLP($V_S, C_V, X_V, \mathcal{T}_{V_S}$)
17 **if** $|V_S| + |C_V| < \theta$ **then return** 0;
18 FLAGMAL $\leftarrow 0$, FLAGPLEX $\leftarrow 0$, $X'_V \leftarrow X_V$;
19 **for each** $v \in C_V$ **do**
20 **if** $V_A^S \subseteq V_S \cup C_V \wedge v \in V_A^S \wedge$ STATEA = 1 **then**
21 continue; /* Lemma 7 */
22 $C_V \leftarrow C_V \backslash \{v\}$;
23 $\{$STATE, $C'_V, \mathcal{T}_{V'_S}\} \leftarrow$ IB-PRUNING($V_S, C_V, \mathcal{T}_{V_S}, v, I$);
24 **if** STATE = 0 **then** continue;
25 FLAGMAL $\leftarrow 1$;
26 FLAG $\leftarrow$ OPTENUMKLP($V_S \cup \{v\}, C'_V, X'_V, \mathcal{T}_{V'_S}$);
27 **if** FLAG = 1 **then**
28 $X'_V \leftarrow X'_V \cup \{v\}$; FLAGPLEX $\leftarrow 1$;
29 **for each** $t \in \mathcal{T}_{V'_S}$ **do**
30 **for each** $u \in C_V \backslash \Gamma(v,t)$ **do**
31 $I[u][t]$ − −;
32 **for each** $u \in V_S \backslash \Gamma(v,t)$ **do**
33 $I[u][t]$ − −;
34 **if** FLAGMAL = 0 $\wedge$ $|V_S| \geq \theta$ **then**
35 **if** $\nexists\, u \in X_V$ *s.t.* $V_S \cup \{u\}$ *is the* (k,l)*-plex* **then**
36 $\mathcal{R} \leftarrow \mathcal{R} \cup \{V_S\}$; /* Lemma 4 */
37 **if** STATEA = 0 **then** STATEA $\leftarrow 1$, $V_A^S \leftarrow V_S$;
38 **return** 1;
39 **if** FLAGPLEX = 1 **then return** 1 **else return** 0;

plex by adding the new vertex from C_V to V_S, in line 35, we check the maximality of V_S by applying Lemma 4. If V_S is maximal, we store V_S into $\mathcal{R}$. Then, we check whether we have obtained the attack vertex set. If STATEA = 0, we set it as 1 and set V_A^S as V_S. In line 39, FLAGPLEX = 1 means that we can find a large (k,l)-plex in the current search branch.

***Theorem* 6** *Algorithm 7 needs* $O(|\mathcal{E}|^{1.5} + |\mathcal{T}|\delta_{max}^4|\mathcal{V}'|2^{\delta_{max}^2})$ *time, where* δ_{max} *denotes the largest momentary degree in* $\mathcal{G}$.

***Proof*. (Proof of Theorem 6)** *In Algorithm 7, we first apply Algorithm 5 to filter unpromising vertices and edges in line 1, which cost* $O(|\mathcal{E}|^{1.5})$. *The initialization of I for each remaining vertex in every timestamp in lines 4-5 takes* $O(|\mathcal{T}||\mathcal{V}'|)$ *times. Note that,* $|\mathcal{V}'|$ *is much smaller than* $|\mathcal{V}|$ *after graph reduction. Then, for each remaining vertex, we invoke the procedure* OPTENUMKLP *to enumerate all the MalKLPs containing it in lines 7-15. The procedure* OPTENUMKLP *runs* $2^{|C_V|}$ *times for each remaining vertex, and* $|C_V|$ *is bounded by* δ_{max}^2. *In procedure* OPTENUMKLP, *we apply Algorithm 6 in line 23 to shrink the candidate vertex set* C_V, *which cost* $O(|\mathcal{T}|\delta_{max}^4)$. *Then, in lines 29-33, we need to restore I, which takes* $O(|\mathcal{T}|\delta_{max}^2)$. *The maximality checking in line 35 runs* $O(|X_V||\mathcal{T}|\delta_{max}^2)$, *where* $|X_V|$ *is bounded by* δ_{max}^2. *Overall, the time complexity of Algorithm 7 is* $O(|\mathcal{E}|^{1.5} + |\mathcal{T}|\delta_{max}^4|\mathcal{V}'|2^{\delta_{max}^2})$.

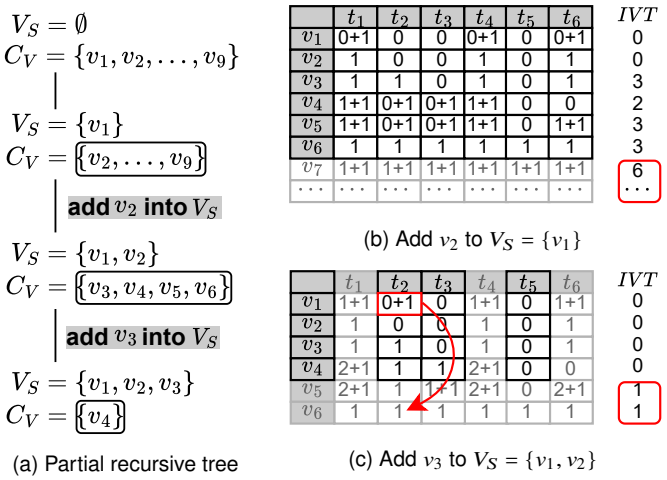


Fig. 10: A running example for IB pruning rule

TABLE I: Statistics of datasets

Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\mathcal{E}$ Type	$\lvert\mathcal{T}\rvert$	Scale	(k,l,θ)
D1 (MathOverflow)	88,581	231,553	answered	28	quarter	(2,4,8)
D2 (Facebook)	46,953	308,405	post	19	quarter	(2,2,5)
D3 (AskUbuntu)	515,281	473,661	answered	31	quarter	(2,3,6)
D4 (Epinions)	131,829	727,295	trust	33	month	(2,3,7)
D5 (SuperUser)	567,316	750,997	answered	32	quarter	(2,4,6)
D6 (Digg)	279,631	1,591,214	vote	49	month	(2,4,11)
D7 (WikiTalk)	519,404	1,894,092	message	57	quarter	(2,8,15)
D8 (DBLP)	1,824,702	11,858,545	collaborate	49	year	(2,6,15)
D9 (itwiki)	1,204,010	28,040,223	Reference	115	month	(2,9,10)
D10 (frwiki)	2,212,683	43,858,464	Reference	122	month	(2,12,13)
D11 (dewiki)	2,166,670	63,553,435	Reference	128	month	(2,10,14)
D12 (DBpedia)	18,268,993	99,662,225	Hyperlink	60	random	(2,5,6)
D13 (dimacs10)	18,483,187	261,787,258	Hyperlink	60	random	(2,6,6)
D14 (wikipedia)	13,593,033	437,167,958	Wikilink	60	random	(2,15,20)

APPENDIX C

***Example* 4 (Example for IB pruning rule)** *Reconsider the temporal graph shown in Fig. 1 with* $k = 2$, $l = 3$ *and* $\theta = 4$. *Fig. 10(a) shows a partial recursive tree generated from the recursive call* ENUMKLP(V_S, C_V) *by incorporating IB pruning rule, where* V_S *and* C_V *are initialized as* $\emptyset$ *and* $\{v_1, v_2, \cdots, v_9\}$, *respectively. In Fig. 10(b) and 10(c), we illustrate the updating of data structures I after adding* v_2 *and* v_3 *successively to* V_S *to form* V'_S *of Fig. 10(a). Notice, the value in the table denotes the number of disconnected vertices of each vertex u (excluding u itself) in* V'_S *at each timestamp t, i.e.,* $I[u][t]$. *The cells with "+" denote the updated values after adding a vertex to* V_S. *IVT denotes the number of invalid timestamps among* $\mathcal{T}_{V'_S}$ *of each vertex.*

In Fig. 10(b), after adding v_2 *into* $V_S = \{v_1\}$ *(i.e.,* $V'_S = \{v_1, v_2\}$*),* $\mathcal{T}_{V'_S} = \{t_1, t_2, \cdots, t_6\}$. *Then, we visit each*

disconnected vertex of v_2 at each timestamp $t \in \mathcal{T}_{V'_S}$ and update the corresponding value in I. Take vertex v_7 as an example, $I[v_7][t]$ for each $t \in \mathcal{T}_{V'_S}$ is increased to 2. Thus, its IVT is 6 according to Condition I, which is larger than $|\mathcal{T}_{V'_S}| - l = 3$, and we need to remove v_7 from the candidate set. Similarly, we remove v_8 and v_9 from the candidate set.

In Fig. 10(c), after adding v_3 into $V_S = \{v_1, v_2\}$ (i.e., $V'_S = \{v_1, v_2, v_3\}$), $I[v_1][t_1]$, $I[v_1][t_4]$ and $I[v_1][t_6]$ are increased to 2, respectively. All these values are larger than $k - 1 = 1$. Therefore, t_1, t_4 and t_6 becomes invalid timestamp for V'_S. We can obtain that $\mathcal{T}_{V'_S} = \{t_2, t_3, t_5\}$. Then, for the disconnected vertex v_5 of v_3 at timestamp t_3, $I[v_5][t_3]$ is increased to 2. We can remove v_5 because t_3 becomes an invalid timestamp for v_5 according to Condition I.

On the other hand, for the disconnected vertex $v_1 \in V'_S$ of v_3 at timestamp t_2, $I[v_1][t_2]$ is increased to 1, which equals to $k - 1 = 1$. Timestamp t_2 becomes an invalid timestamp for $V'_S \cup \{v_6\}$ according to Condition II. Then, we remove v_6 from the candidate vertex set, since $IVT[v_6] = 1 > |\mathcal{T}_{V'_S}| - l = 0$.

APPENDIX C

This part is about Experiment Setup in Section V.

Algorithms. Note that, for the MalKLP enumeration problem, KLPE-BK cannot finish in a reasonable time if applied directly. Therefore, we further equip it with the graph reduction technique proposed. In the experiments, we implement and evaluate four algorithms for the MalKLP enumeration problem. *i*) **KLPE-BK+**: the baseline method proposed in Section III-A with graph reduction technique; *ii*) **KLPE+**: our final algorithm with all the optimizations; *iii*) **KLPE-GR**: KLPE+ without the graph reduction technique; *iv*) **KLPE-BR**: KLPE+ without the search branch reduction techniques.

We also implement and evaluate the following three algorithms for the MaxKLP identification problem. *i*) **MKLP-BS**: the baseline method proposed in Section IV by extending the developed enumeration algorithm; *ii*) **MKLP+**: Algorithm 2 proposed in Section IV-A by incorporating all optimized strategies. *iii*) **MKLP-GR**: MKLP+ without the modified graph reduction technique proposed in Section IV-B.

Datasets. We employ 11 real-world temporal graphs in our experiments, whose details are shown in Table I. $|\mathcal{T}|$ is the number of snapshots. Among these datasets, D1, D3 and D5 are obtained from SNAP[1]. The other datasets are obtained from KONECT[2]. Datasets D1–D11 are real-world temporal graphs, while D12 and D13 are synthetic graphs in which edges are randomly assigned timestamps. All these datasets are publicly available.

Parameters and workload. We conduct the experiment by varying parameters k, l and θ, whose default values are shown in the last column of Table I. For those algorithms that cannot finish within 12 hours, we set them as **INF**. All the programs are implemented in standard C++. All the experiments are performed on a server with an Intel Xeon 2.1GHz CPU and 256 GB main memory.

This part is about more Experiment Evaluation in Section V.

[1]http://snap.stanford.edu

[2]http://konect.cc/networks

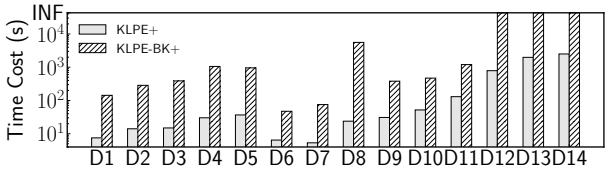


Fig. 11: Response time on all the datasets

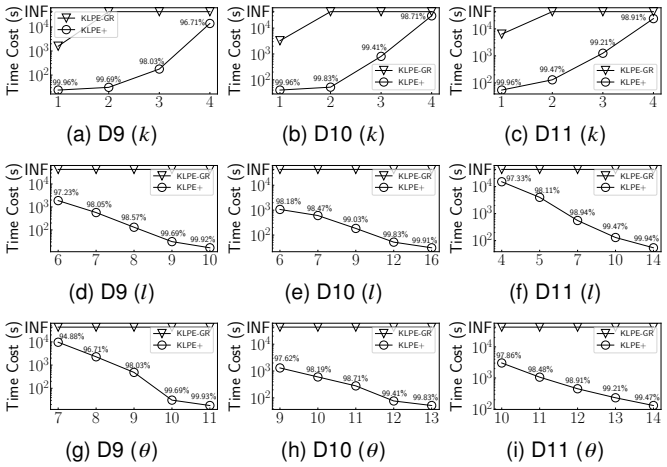
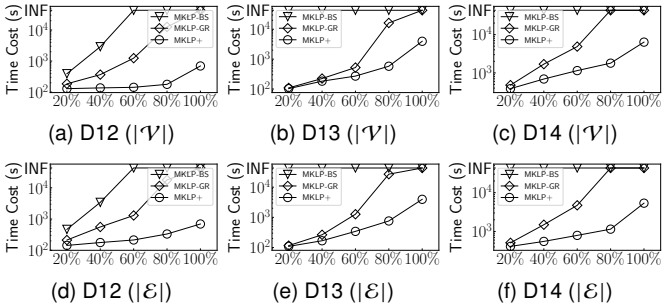


Fig. 12: Evaluation of the graph reduction technique

More experiments about MalKLP Enumeration: Experiments over all the datasets. In this experiment, we report the response time of KLPE-BK+ and KLPE+ on all the datasets with the default settings. The results are shown in Fig. 11. As we can see, KLPE+ outperforms KLPE-BK+ on all the datasets with a significant margin. For instance, on D6 with more than 1.59 million edges, KLPE-BK+ needs 193.17 seconds to enumerate all the MalKLPs, while KLPE+ only requires 11.23 seconds. On D8, compared with KLPE-BK+, KLPE+ can achieve up to two orders of magnitude speedup. This is because *i*) KLPE+ applies the advanced branch filtering techniques, which can extremely reduce the searching space; and *ii*) it optimizes the maximality checking method and can avoid numerous comparisons.

More experiments about MalKLP Enumeration: Evaluation of the graph reduction technique. To evaluate the performance of the graph reduction technique, i.e., Algorithm 5, we conduct the experiments on D9, D10 and D11 by varying k, l and θ, respectively. The results are shown in Fig. 12, where the two lines represent the response time of KLPE+ and KLPE-GR, and the value above the line of KLPE+ denotes the corresponding percentage of edges pruned. With the increase of parameters l and θ, more edges can be pruned due to the tighter constraints. For larger k, the number of pruned edges slightly decreases due to the relaxation in cohesiveness. As shown, the graph reduction technique can significantly shrink the number of edges, which confirms the effectiveness of the proposed strategy. For example, for D10, over 98% of the edges can be pruned. Compared with KLPE+, KLPE-GR cannot finish in 12 hours for most cases, except for $k = 1$.

More experiments about MaxKLP Identification: Scalability evaluation of all the algorithms. We also conduct the ex-



(a) D12 ($|\mathcal{V}|$) (b) D13 ($|\mathcal{V}|$) (c) D14 ($|\mathcal{V}|$)

(d) D12 ($|\mathcal{E}|$) (e) D13 ($|\mathcal{E}|$) (f) D14 ($|\mathcal{E}|$)

Fig. 13: Scalability evaluation by varying $|\mathcal{V}|$ and $|\mathcal{E}|$

periment to evaluate the scalability of the proposed algorithms on D12, D13 and D14. Following the experimental setting of Exp-3, we report the response time of algorithms in Fig. 13. As observed, MKLP+ consistently outperforms MKLP-BS in all settings, demonstrating its better scalability. For example, when $|\mathcal{V}|$ = 20% on D12, the response time of MKLP-BS, MKLP-GR and MKLP+ is 391 seconds, 185 seconds and 129 seconds, respectively. When $|\mathcal{V}|$ = 80%, MKLP-BS cannot return result in 12 hours. MKLP-GR and MKLP+ requires 12591 seconds and 178 seconds, respectively.