Appendix A

Proof. **(Proof of Theorem 1)** *Line 2 iterates through each edge in $\mathcal{E}(\mathcal{G})$ once, therefore, the total time complexity is $O(m)$. $\mathcal{G}_q$ takes $O(n+m)$ space, therefore, the total space complexity is $O(n+m)$.*

Proof. **(Proof of Theorem 2)** *Lines 1-2 initialize the $\mathcal{A}(u)$ and $\mathcal{D}(u)$ for each vertex $u \in \mathcal{V}(\mathcal{G})$, which takes $O(n)$ time. Line 6 scans each edge in $\mathcal{E}(\mathcal{G})$ once, which takes $O(m)$ time. Therefore, the total time complexity is $O(n+m)$.*

Throughout the algorithm, we maintain $\mathcal{A}(u)$, $\mathcal{D}(u)$ and pointer in $N_{out}(u,\mathcal{G})$ (resp. $N_{in}(u,\mathcal{G})$) for each vertex $u \in \mathcal{V}(\mathcal{G})$; during the traversal, in Line 9, at most one copy of each vertex is in Q, resulting in a space complexity of $O(n)$.

Proof. **(Proof of Theorem 3)** *Lines 4 initialize $TCV_\tau(s,u)$ for each $\tau \in \mathcal{T}_{in}(u,\mathcal{G}_q)$ of each vertex $u \in \mathcal{V}(\mathcal{G}_q) \setminus \{s,t\}$, which takes $O(m)$ time. Lines 5-6 take $O(n)$ time to initialize the* completed *indicator and the pointer of each vertex $u \in \mathcal{V}(\mathcal{G}_q) \setminus \{s,t\}$. For each edge in $\mathcal{E}(\mathcal{G}_q)$, Lines 8-15 take $O(1)$ time, Lines 17 and 19 take $O(\theta)$ time as the number of vertices in each entry of $TCV_\cdot(s,\cdot)$ is bounded by $\theta-1$. Therefore, the total time complexity is $O(n+\theta\cdot m)$, which is polynomial with respect to the input size since $\theta \le |\mathcal{T}| \ll m$.*

There are $O(m)$ entries in $TCV_\cdot(s,\cdot)$ and $TCV_\cdot(\cdot,s)$, and each entry has a length bounded by $\theta-1$. Throughout the algorithm, we maintain the completed *indicator and the pointer for each vertex $u \in \mathcal{V}(\mathcal{G}_q)$. Therefore, the total space complexity is $O(n+\theta\cdot m)$.*

Proof. **(Proof of Theorem 4)** *The pointer initialization in Line 1 takes $O(n)$ time and the pointer operations in Lines 8-10 take $O(m)$ time in total. Line 3 iterates through each edge in $\mathcal{E}(\mathcal{G}_q)$ once and the intersection operation in Line 17 is performed at most m times. Each intersection operation takes $O(\theta)$ time as the length of each entry in $TCV_\cdot(s,\cdot)$ and $TCV_\cdot(\cdot,s)$ is bounded by $\theta-1$. Therefore, the total time complexity is $O(n+\theta\cdot m)$, which is polynomial with respect to the input size, since $\theta \le |\mathcal{T}| \ll m$. $\mathcal{G}_t$ takes $O(n+m)$ space, therefore, the total space complexity is $O(n+m)$.*

Proof. **(Proof of Theorem 5)** *Line 2 initializes a* verified *indicator for each edge, therefore takes $O(m)$ time. Lines 3-5 can be implemented as a traversal in $\mathcal{G}_t$ from s (resp. t), which takes $O(m)$ time. For each unverified edge $e(u,v,\tau)$, the Bidirectional DFS in Line 9 has a depth of $\theta-1$ at most with a time complexity bounded by $O(d_t^{\theta-1})$, Lines 11-18 takes $O(d_t\cdot\theta)$ time to verify edges in the batch of paths. Therefore the total time complexity is $O(m\cdot d_t^{\theta-1})$.*

$tspG$ takes $O(n+m)$ space and the verified indicator for all edges takes $O(m)$ space. The space for stacks $\mathcal{S}_v$ and $\mathcal{S}_e$ is in $O(n)$. Thus, the total space complexity is $O(n+m)$.

Proof. **(Proof of Lemma 3)** ***Sufficiency***. *If there exists a temporal simple path $p^*_{[\tau_b,\tau_e]}(s,t)$ through $e(u,v,\tau)$, there exist two temporal simple paths $p^*_{[\tau_b,\tau_i]}(s,u), p^*_{[\tau_j,\tau_e]}(v,t)$ s.t. $\tau_i < \tau < \tau_j$, and $\mathcal{V}(p^*_{[\tau_b,\tau_i]}(s,u)) \cap \mathcal{V}(p^*_{[\tau_j,\tau_e]}(v,t)) = \emptyset$. Based on the definition of time-stream common vertices, we have $TCV_{\tau_i}(s,u) \subseteq \mathcal{V}(p^*_{[\tau_b,\tau_i]}(s,u))$ and $TCV_{\tau_j}(v,t) \subseteq \mathcal{V}(p^*_{[\tau_j,\tau_e]}(v,t))$, thus, $TCV_{\tau_i}(s,u) \cap TCV_{\tau_j}(v,t) = \emptyset$.*

Necessity. *Consider $e(c,f,4) \in \mathcal{E}(\mathcal{G}_q)$ in Fig. 3(c) as an counterexample. There only exist $\tau_i = 3$ and $\tau_j = 5$ satisfying $\tau_i < 4 < \tau_j$, and we have $TCV_3(s,c) \cap TCV_5(f,t) = \emptyset$ as $TCV_3(s,c) = \{b,c\}$ and $TCV_5(f,t) = \{f\}$. However, there does not exist a temporal simple path $p^*_{[2,7]}(s,t)$ through $e(c,f,4)$. Therefore, the necessity is not established.*

Proof. **(Proof of Lemma 5)** *To proof Lemma 5, we first proof $\mathcal{P}^*_{[\tau_b,\tau_l]}(s,u) = \mathcal{P}^*_{[\tau_b,\tau]}(s,u)$ by contradiction. Since $\tau_l \le \tau$, $\mathcal{P}^*_{[\tau_b,\tau_l]}(s,u) \subseteq \mathcal{P}^*_{[\tau_b,\tau]}(s,u)$. Suppose $\mathcal{P}^*_{[\tau_b,\tau_l]}(s,u) \neq \mathcal{P}^*_{[\tau_b,\tau]}(s,u)$, this implies that there exists $p^*_{[\tau_b,\tau]}(s,u) \in \mathcal{P}^*_{[\tau_b,\tau]}(s,u)$ such that $p^*_{[\tau_b,\tau]}(s,u) \notin \mathcal{P}^*_{[\tau_b,\tau_l]}(s,u)$. It is evident that such $p^*_{[\tau_b,\tau]}(s,u)$ contains an in-coming edge $e(v,u,\tau')$ of u where $\tau_l < \tau' \le \tau$, which contradicts $\tau_l = \max\{\tau_i | \tau_i \in \mathcal{T}_{in}(u,\mathcal{G}_q), \tau_i \le \tau\}$. In conclusion, $\mathcal{P}^*_{[\tau_b,\tau_l]}(s,u) = \mathcal{P}^*_{[\tau_b,\tau]}(s,u)$, and we have $TCV_\tau(s,u) = TCV_{\tau_l}(s,u)$ following the definition of the time-stream common vertices. We omit the proof for $TCV_\tau(u,t) = TCV_{\tau_r}(u,t)$ as it follows a similar approach.*

Proof. **(Proof of Lemma 6)** *For each temporal path $p \in \mathcal{P}_{[\tau_b,\tau]}(s,u)$, there exists a corresponding temporal simple path $p^* \in \mathcal{P}^*_{[\tau_b,\tau]}(s,u)$ such that the path p^* contains all edges in the path p except those forming cycles. Then, we have $\mathcal{V}(p^*) \subseteq \mathcal{V}(p)$ to derive that $\mathcal{V}(p^*) \cap \mathcal{V}(p) = \mathcal{V}(p^*)$. Thus, $TCV_\tau(s,u) = \bigcap_{p^* \in \mathcal{P}^*_{[\tau_b,\tau]}(s,u)\ s.t.\ t \notin \mathcal{V}(p^*)} \mathcal{V}(p^*) \setminus \{s\} = \bigcap_{p \in \mathcal{P}_{[\tau_b,\tau]}(s,u)\ s.t.\ t \notin \mathcal{V}(p)} \mathcal{V}(p) \setminus \{s\}$. The proof for $TCV_\tau(u,t)$ follows the same approach.*

Proof. **(Proof of Lemma 8)** *If $TCV_{\tau_l}(s,u) \cap TCV_{\tau_r}(v,t) \neq \emptyset$, i.e., there exits w such that $w \in TCV_{\tau_l}(s,u)$ and $w \in TCV_{\tau_r}(v,t)$, then based on the definition of time-stream common vertices, $\forall\ p^* \in \mathcal{P}^*_{[\tau_b,\tau_l]}(s,u)\ s.t.\ t \notin \mathcal{V}(p^*)$, $w \in p^*$. Since $\forall\ \tau_b \le \tau_i < \tau_l$, we have $\mathcal{P}^*_{[\tau_b,\tau_i]}(s,u) \subseteq \mathcal{P}^*_{[\tau_b,\tau_l]}(s,u)$, thus, $\forall\ p^*_i \in \mathcal{P}^*_{[\tau_b,\tau_i]}(s,u)\ s.t.\ t \notin \mathcal{V}(p^*_i)$, $w \in p^*_i$, that is, $w \in TCV_{\tau_i}(s,u)$. Similarly, $\forall\ \tau_r < \tau_j \le \tau_e$, $w \in TCV_{\tau_j}(v,t)$. Therefore, $TCV_{\tau_i}(s,u) \cap TCV_{\tau_j}(v,t) \neq \emptyset$.*

Proof. **(Proof of Lemma 9)** *($\Rightarrow$) If an edge $e(u,v,\tau) \in \mathcal{E}(\mathcal{G}_q)$, where $u \neq s$ and $v \neq t$, satisfies condition i), based on Lemma 3 and Lemma 8, such an edge will not be excluded as an unpromising edge. Therefore, it should be included in $\mathcal{G}_t$. If an edge $e(u,v,\tau) \in \mathcal{E}(\mathcal{G}_q)$, where $u = s$ or $v = t$, based on the condition ii) of Lemma 2, we can conclude that $e(u,v,\tau)$ belongs to $tspG$. Since $\mathcal{G}_t$ is the upper-bound graph of $tspG$, $e(u,v,\tau)$ must belong to $\mathcal{G}_t$.*

($\Leftarrow$) If an edge $e(u,v,\tau)$ belongs to $\mathcal{G}_t$, there does not exist a vertex w appearing in all $p^*_{[\tau_b,\tau_i]}(s,u)$ and $p^*_{[\tau_j,\tau_e]}(v,t)$ where $\tau_i < \tau < \tau_j$. Based on the Definiton 5, it is easy to derive that $e(u,v,\tau)$ satisfies condition i) s.t. $u \neq s$ and $v \neq t$. Since $e(u,v,\tau) \in \mathcal{G}_t$ must belong to $\mathcal{G}_q$, and we have discussed $w \neq s$ and $w \neq t$ for each edge in $\mathcal{G}_q$, therefore u can be the source vertex s or v can be the target vertex t.

Proof. **(Proof of Lemma 10)** *Let $\tau_l = \max\{\tau'' | \tau'' \in \mathcal{T}_{in}(u, \mathcal{G}_q) \wedge \tau_b \le \tau'' < \tau\}$, $\tau_r = \min\{\tau'' | \tau'' \in \mathcal{T}_{out}(v,\mathcal{G}_q) \wedge \tau < \tau'' \le$*

Algorithm 7: BiDirSearch

Input : an edge $e(u, v, \tau) \in \mathcal{E}(\mathcal{G}_t)$
Output: the existence of $p^*_{[\tau_b, \tau_e]}(s, t)$ through $e(u, v, \tau)$

1 $S_v.push(u)$, $S_v.push(v)$;
2 **if** $\tau - \tau_b > \tau_e - \tau$ **then** $dir = f$ **else** $dir = b$;
3 $F \leftarrow false, B \leftarrow false$;
4 **if** Search (u, v, τ, dir, F, B) **then return** *true*;
5 **else return** *false*;
6 **Procedure** Search (u, v, τ, dir, F, B) **:**
7 **if** $dir = f$ **then** $S_v.push(v)$ **else** $S_v.push(u)$;
8 $S_e.push(e(u, v, \tau))$;
9 **if** $v = t \vee u = s$ **then**
10 **if** $v = t$ **then** $dir = b$, $F \leftarrow true$;
11 **if** $u = s$ **then** $dir = f$, $B \leftarrow true$;
12 **if** $F \wedge B$ **then return** *true*;
13 $e(u_0, v_0, \tau_0) \leftarrow$ the first edge in S_e;
14 **if** Search $(u_0, v_0, \tau_0, dir, F, B)$ **then return** *true*;
15 **if** $dir = f$ **then**
16 **for each** $(w_i, \tau_i) \in N_{out}(v, \mathcal{G}_t)$ **do**
17 **if** $\tau_i \leq \tau$ **then break**;
18 **if** $w_i \in S_v$ **then continue**;
19 **if** Search $(v, w_i, \tau_i, dir, F, B)$ **then return** *true*;
20 **else**
21 **for each** $(w_i, \tau_i) \in N_{in}(u, \mathcal{G}_t)$ **do**
22 **if** $\tau_i \geq \tau$ **then break**;
23 **if** $w_i \in S_v$ **then continue**;
24 **if** Search $(w_i, u, \tau_i, dir, F, B)$ **then return** *true*;
25 $S_v.pop()$, $S_e.pop()$;
26 **return** *false*;
27 **end**

$\tau_e\}$. *For condition i), if there exists an edge* $e(s, u, \tau') \in \mathcal{E}(\mathcal{G}_t)$ *such that* $\tau_b \leq \tau' \leq \tau_l < \tau$, *then we have a temporal simple path* $p^*_{[\tau_b, \tau_l]}(s, u) = \langle e(s, u, \tau') \rangle$, *therefore* $TCV_{\tau_l}(s, u) = \{u\}$. *Since* $e(u, v, \tau) \in \mathcal{G}_t$, *according to Lemma 9,* $TCV_{\tau_l}(s, u) \cap TCV_{\tau_r}(v, t) = \emptyset$, *that is,* $u \notin TCV_{\tau_r}(v, t)$. *Then, based on the definition of* $TCV_{\tau_r}(v, t)$, *there exists a temporal simple path* $p^*_{[\tau_r, \tau_e]}(v, t)$ *that does not pass through s and u. Therefore, a temporal simple path* $p^*_{[\tau_b, \tau_e]}(s, t)$ *that includes* $e(u, v, \tau)$ *can be formed. We omit the proof for condition ii) as it is similar to that of condition i).*

APPENDIX B

BiDirSearch. With the proposed optimizations, our implementation of the bidirectional DFS is detailed in Algorithm 7. In Line 2, we determine whether to perform a forward or a backward search first. Then, Line 4 starts the *search* procedure with two flags F (resp. B), indicating whether a forward path (resp. backward path) has been found in the current search, initially set to $false$. The *search* procedure recursively searches for a temporal simple path in its given direction. Once a forward (resp. backward) path is completed, it sets the corresponding flag F (resp. B) and toggles the search direction in Lines 10-11. If the path in only one direction is completed, the *search* procedure will continue in the other direction and will terminate upon finding paths in both directions in Line 12. In Lines 16 and 21, the neighbor exploration order is determined based on the current search direction.

TABLE I: Statistics of datasets

Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{T}\rvert$	d	θ
D1 (email-Eu-core)	1,005	332,334	803	9,782	10
D2 (sx-mathoverflow)	88,581	506,550	2,350	5,931	20
D3 (sx-askubuntu)	159,316	964,437	2,613	8,729	20
D4 (sx-superuser)	194,085	1,443,339	2,773	26,996	20
D5 (wiki-ru)	457,018	2,282,055	4,715	188,103	25
D6 (wiki-de)	519,404	6,729,794	5,599	395,780	25
D7 (wiki-talk)	1,140,149	7,833,140	2,320	264,905	20
D8 (flickr)	2,302,926	33,140,017	196	34,174	10
D9 (sx-stackoverflow)	6,024,271	63,497,050	2,776	101,663	20
D10 (wikipedia)	2,166,670	86,337,879	3,787	218,465	25

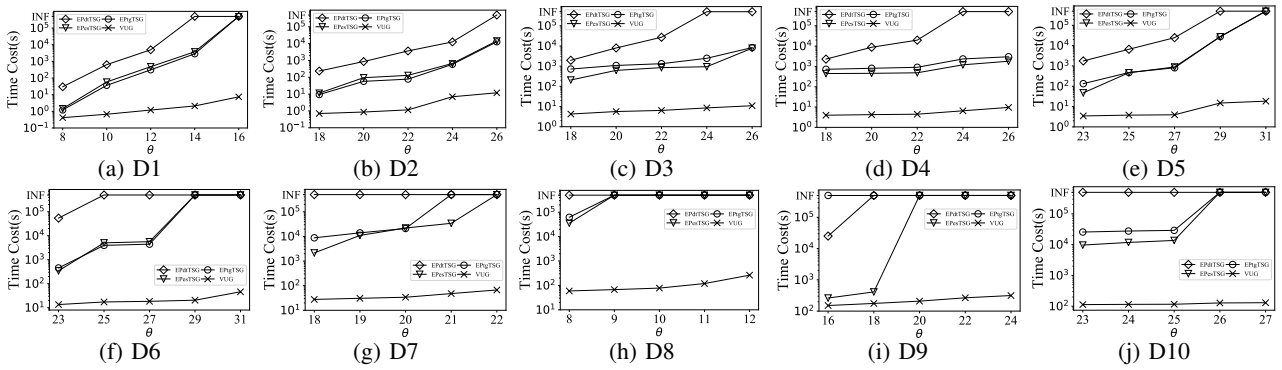
APPENDIX C

Datasets. We employ 10 real-world temporal graphs in our experiments. Details of these datasets are summarized in TABLE I, where d is the maximum degree in a dataset. Among these datasets, D5, D6, D8 and D10 are obtained from KONECT[2], the other six datasets are obtained from SNAP[3].

Supplement to Exp-2: Response time by varying parameter θ on the other datasets. In Fig. 14, we report the total response time of `VUG` for 1000 queries by varying θ, along with the three baseline algorithms as comparison. On each dataset, the response time of VUG grows modestly, which confirms the scalability of our proposed methods. For example, on the largest dataset D10, three baseline methods cannot finish queries within 12 hours when $\theta = 26$, while the total response time of VUG only increases by a factor of 1.2 when θ increases from 23 to 27.

Supplement to Exp-5: Evaluation of upper-bound graph generation of θ's impact on the other datasets. Fig. 15 reports the response time and the upper-bound ratio for `QuickUBG` and `TightUBG` on each dataset. As shown, the upper-bound ratio slightly decreases with increasing θ in most cases, i.e., D1 (97.5%-61.3%), D3 (75.4%-67.3%), D4 (92.2%-85.5%), D5 (97.6%-95.4%), D6 (94.4%-87.6%), D7 (90.8%-86.2%), D8 (99.5%-96.5%), D9 (98.5%-91.6%) within presented ranges of θ. However, in some cases, the ratio slightly increases, i.e., D2 (98.2%-98.4%) as θ increases from 18 to 20 and D10 (88.2%-99.6%) as θ increases from 23 to 25. Meanwhile, the overhead of `QuickUBG` and `TightUBG` remains stable across various settings of θ on all the datasets. For example, on the largest dataset D10, when θ increases from 23 to 27, the time overhead of VUG increases from 113 seconds to 130 seconds. Within this, the overhead of `QuickUBG` ranges from 105 seconds to 107 seconds, and that of `TightUBG` ranges from 4.3 seconds to 4.6 seconds.

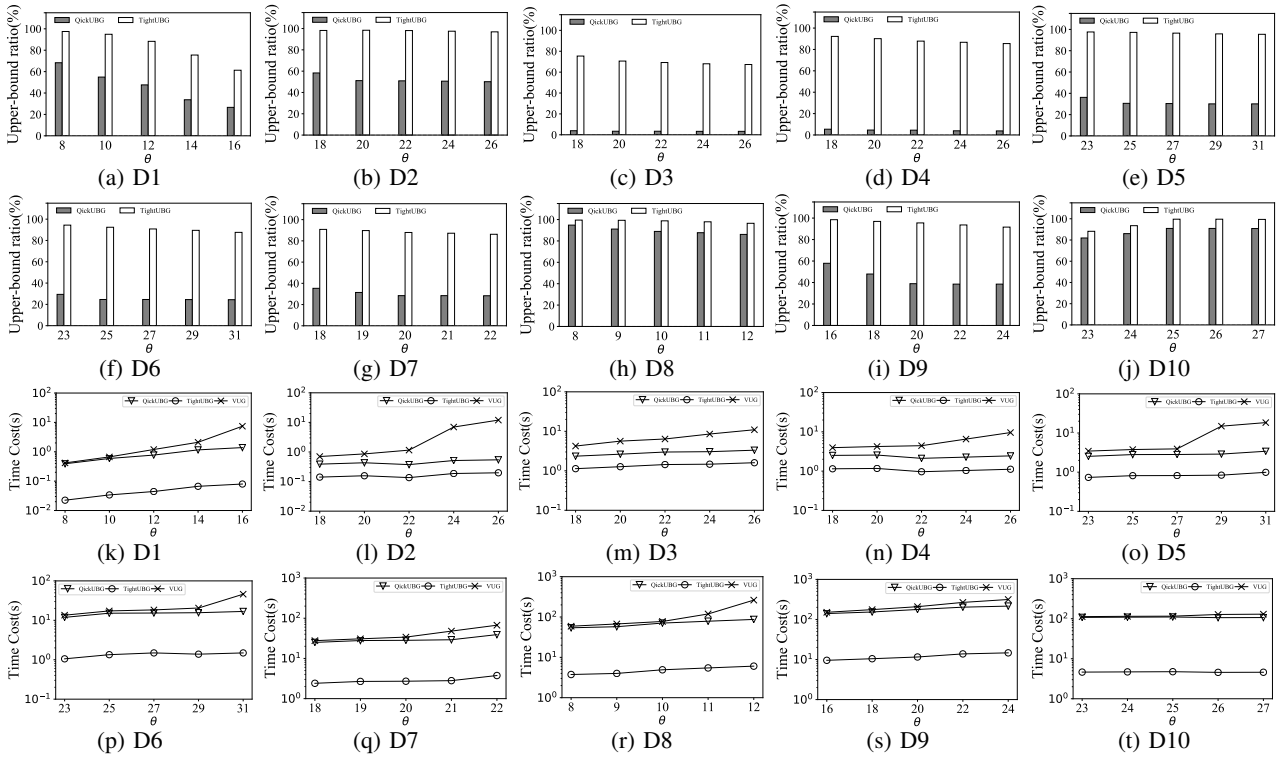
[2]http://konect.cc/networks/
[3]http://snap.stanford.edu



(a) D1 (b) D2 (c) D3 (d) D4 (e) D5

(f) D6 (g) D7 (h) D8 (i) D9 (j) D10

Fig. 14: Response time by varying parameter θ on all other datasets



(a) D1 (b) D2 (c) D3 (d) D4 (e) D5

(f) D6 (g) D7 (h) D8 (i) D9 (j) D10

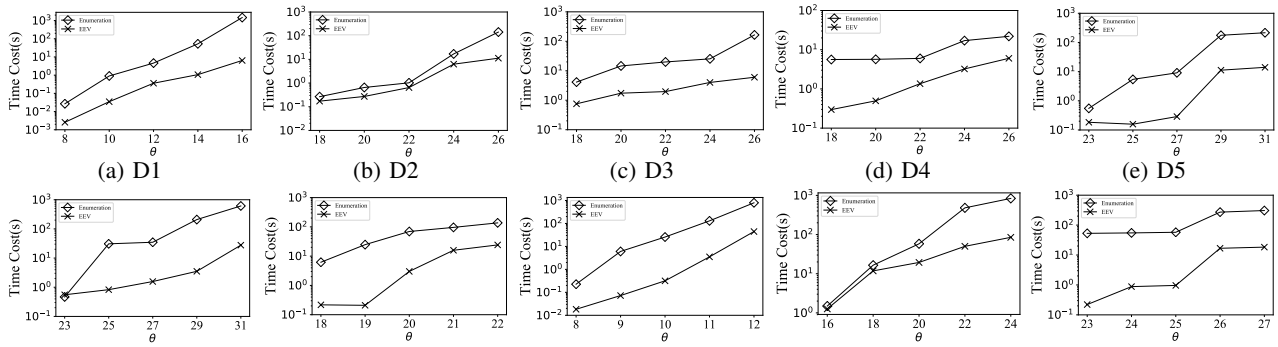
(k) D1 (l) D2 (m) D3 (n) D4 (o) D5

(p) D6 (q) D7 (r) D8 (s) D9 (t) D10

Fig. 15: Evaluation of upper-bound graph generation on all other datasets

Supplement to Exp-6: Evaluation of `EEV` on the other datasets. We compare `EEV` with path enumeration method by applying them separately on $\mathcal{G}_t$ and generating $tspG$. Fig. 16 reports the total response time of both methods for 1000 queries on each dataset. As observed, `EEV` accelerates the process of generating $tspG$ by at least an order of magnitude compared to path enumeration method in the most datasets across various setting of θ. For example, on the largest dataset D10, when θ is 23, enumerating paths to generate $tspG$ takes 53 seconds, while `EEV` only takes 0.2 seconds and when θ is 27, enumeration method takes 878 seconds, while `EEV` only takes 18 seconds.

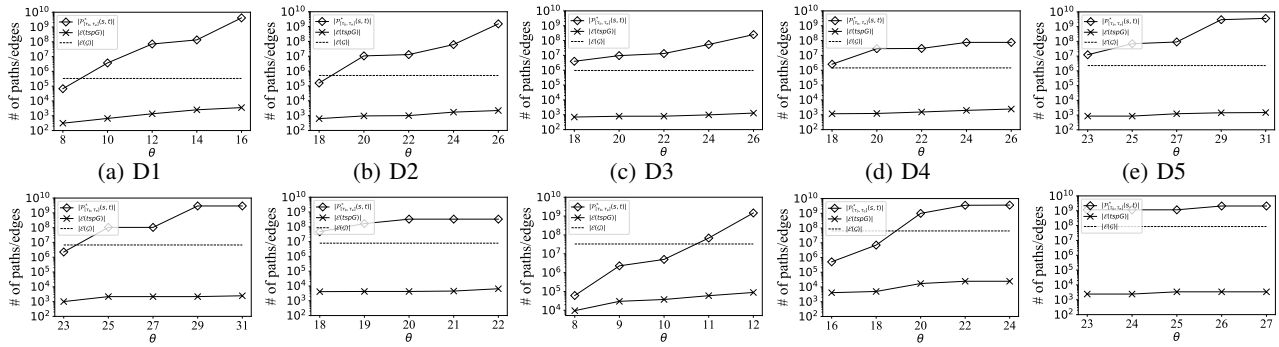
Supplement to Exp-7: Number of edges in $tspG$ on the other datasets. In this experiment, we report the number of edges in $tspG$ and the number of temporal simple paths it contains on each dataset by varying θ. Results are shown in Fig. 17. As we can see, the number of temporal simple paths in $tspG$ far exceeds the number of edges in $tspG$ on all the datasets. For example, when $\theta = 25$ on the largest dataset D10, the generated $tspG$ with 3442 edges contains more than 1.1 billion temporal simple paths.



(a) D1 (b) D2 (c) D3 (d) D4 (e) D5

(f) D6 (g) D7 (h) D8 (i) D9 (j) D10

Fig. 16: Evaluation of EEV on all other datasets



(a) D1 (b) D2 (c) D3 (d) D4 (e) D5

(f) D6 (g) D7 (h) D8 (i) D9 (j) D10

Fig. 17: Numbers of paths/edges in $tspG$ on all other datasets