Efficient Approximate Temporal Triangle Counting in Streaming with Predictions

Giorgio Venturin*

Ilie Sarpe*

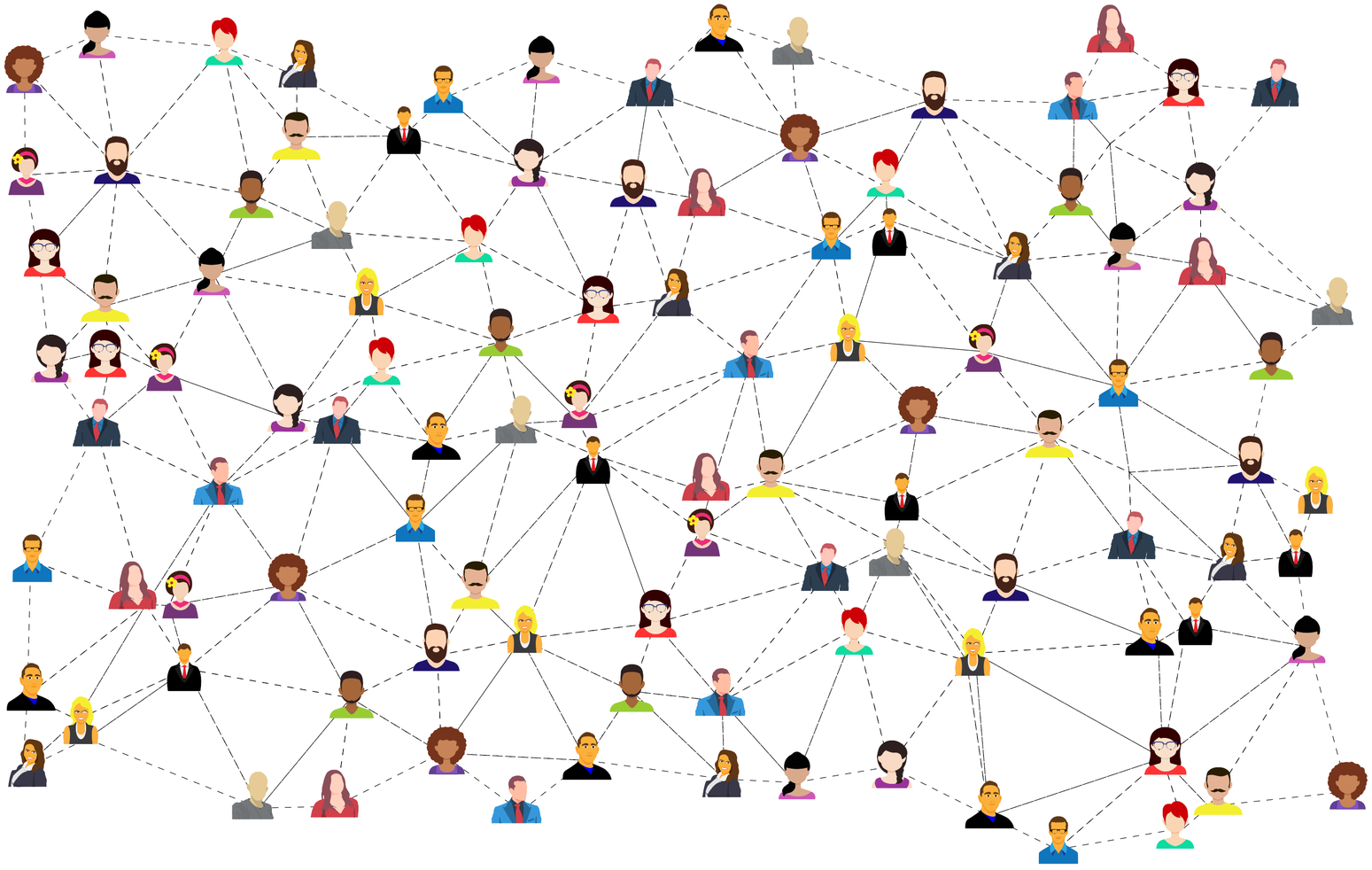
Fabio Vandin

*equal contribution

fabio.vandin@unipd.it

Temporal triangle counting

Counting triangles is a **key primitive** in graph analysis

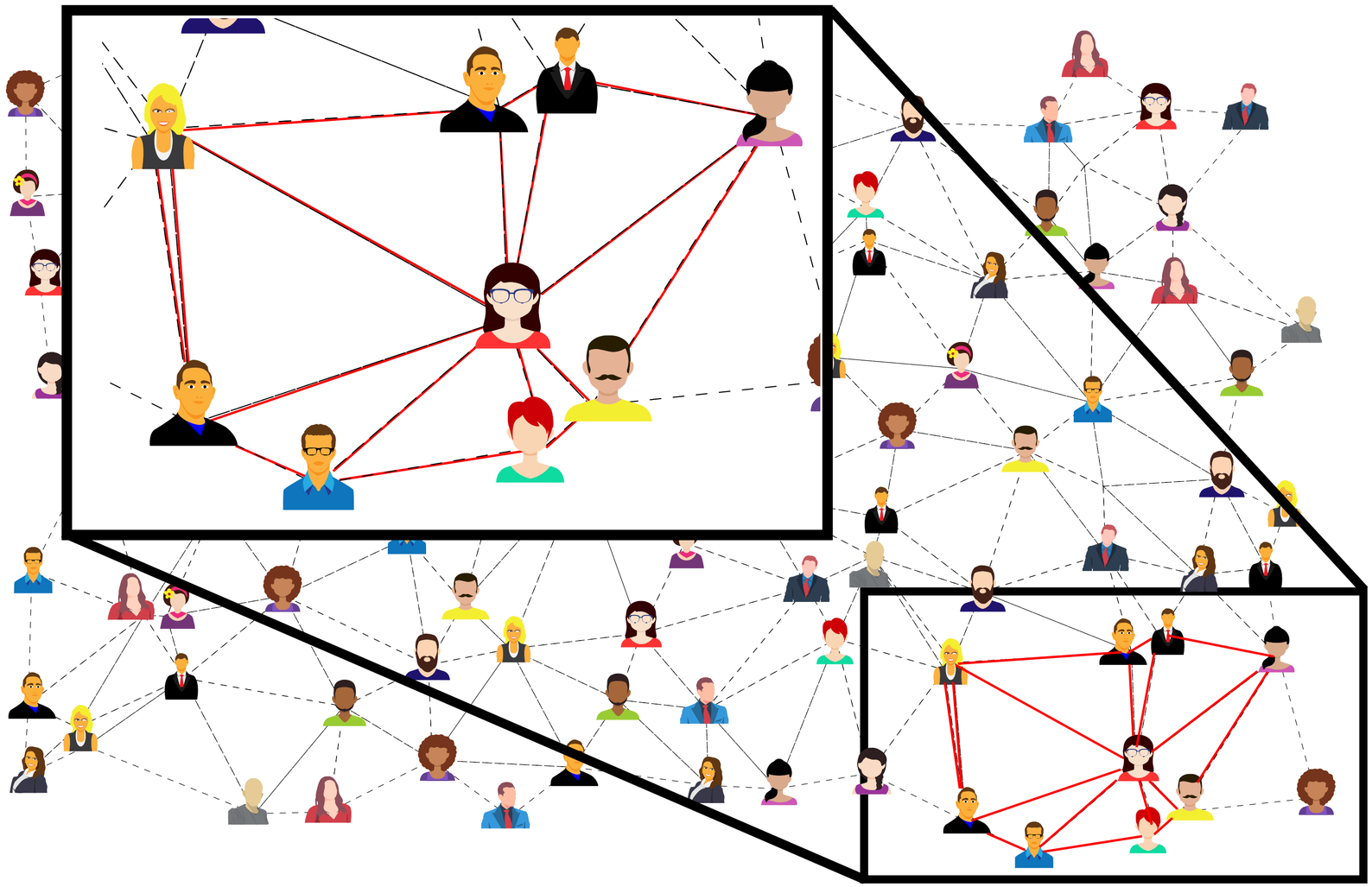


Temporal triangle counting

Counting triangles is a **key primitive** in graph analysis

Example: community detection in social networks

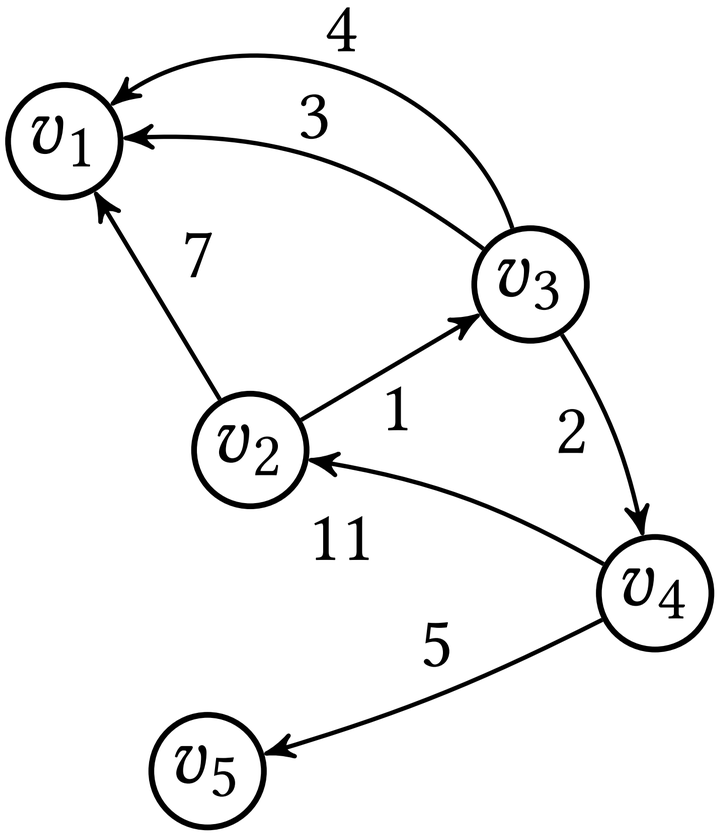
Other applications include anomaly detection, graph classification, biology and more

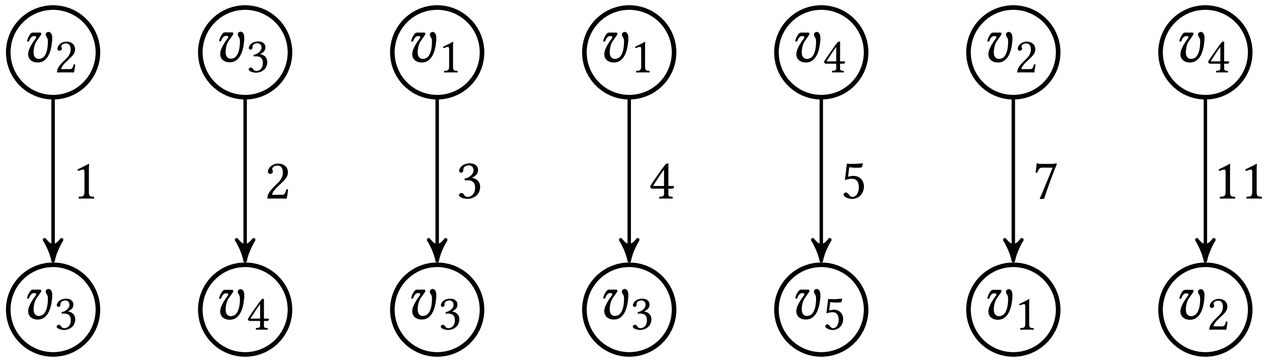


Temporal triangle counting

A temporal graph G is a pair (V, E) where V is the set of vertices and E is the set of edges.

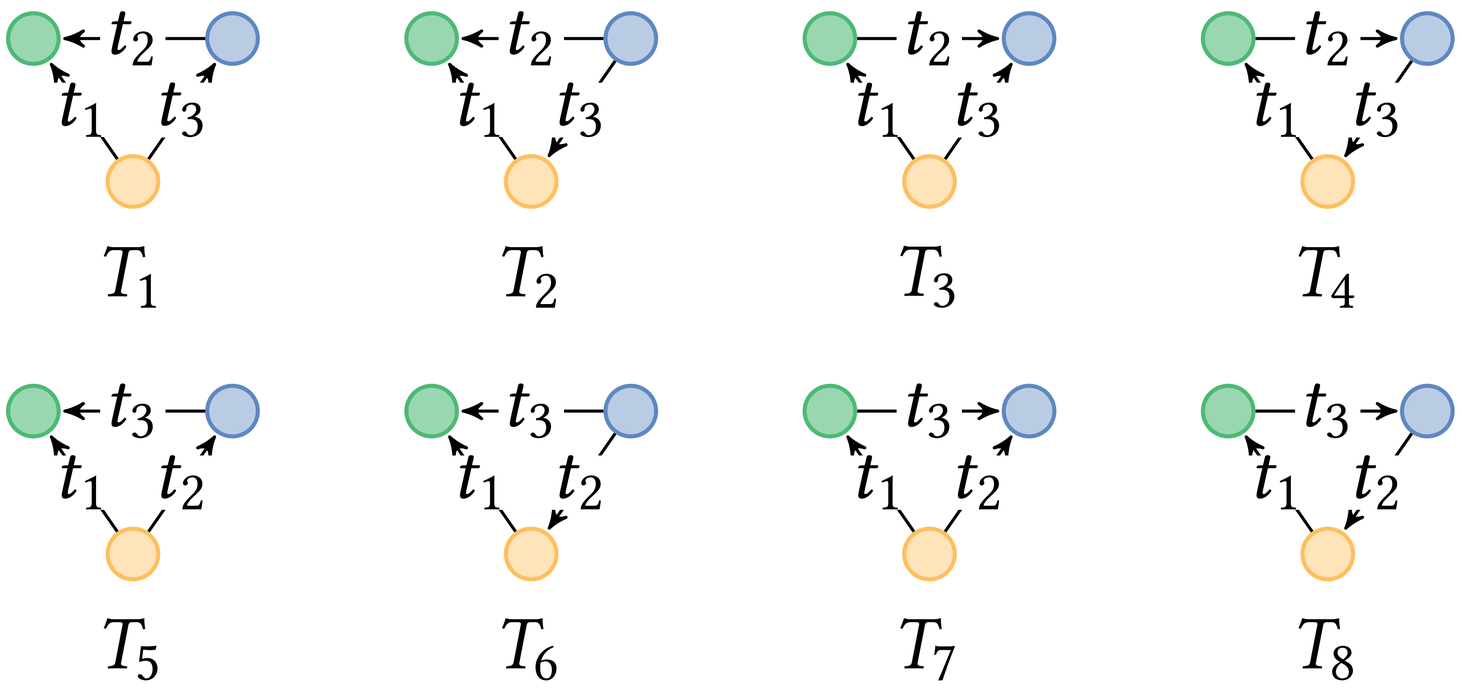
Each edge $e = (u, v, t)$ is **directed** from u to v, and t is the **time** at which the edge occured.





Temporal triangle counting

Informally. A temporal triangle T is a **triangle-shaped** subgraph with an **ordering** over its edges

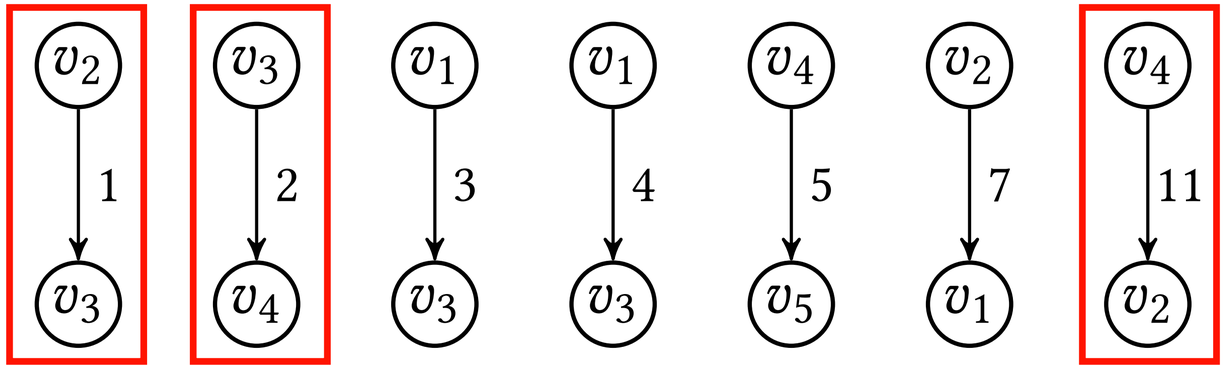


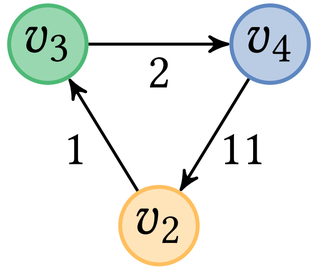
Temporal triangle counting

[Paranjape, Benson, Leskovec, WSDM 2017] Given a *time duration* $\delta \in \mathbb{R}^+$, a triplet of edges is a **δ-instance** of a temporal triangle if:

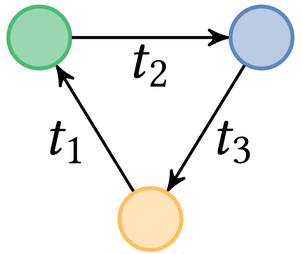
1. The triplet forms a temporal triangle

2. The duration of the triplet is at most δ





T_4 δ-instance
($\delta = 20$)

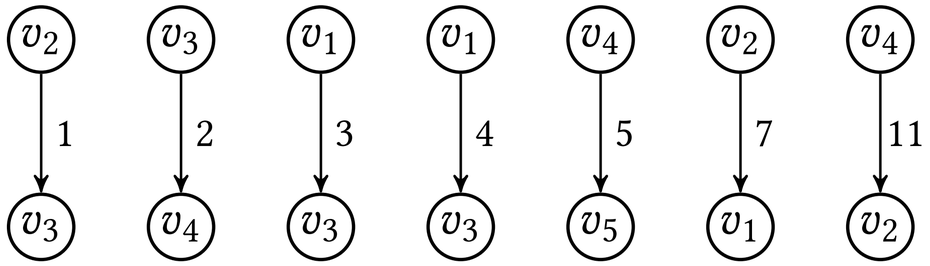


T_4: temporal triangle

Temporal triangle counting

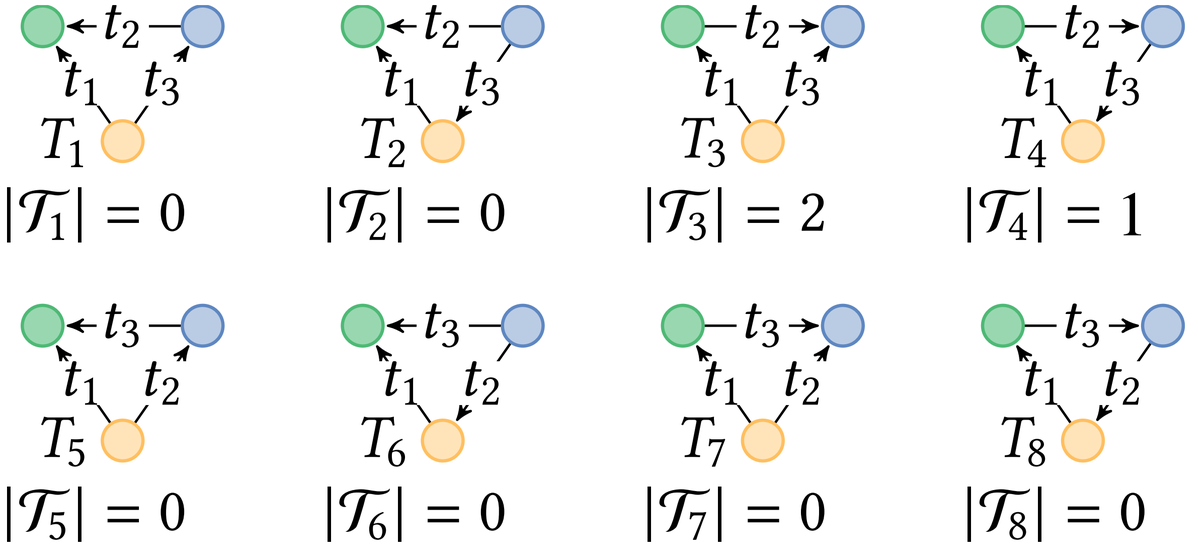
Temporal triangle counting in streaming: *Given a temporal graph* $G = (V, E)$ *processed in streaming and a time duration* δ*, output the number of* δ*-instances in* G *of each temporal triangle* T_i*, with* $i \in \{1,\ldots,8\}$.

Input



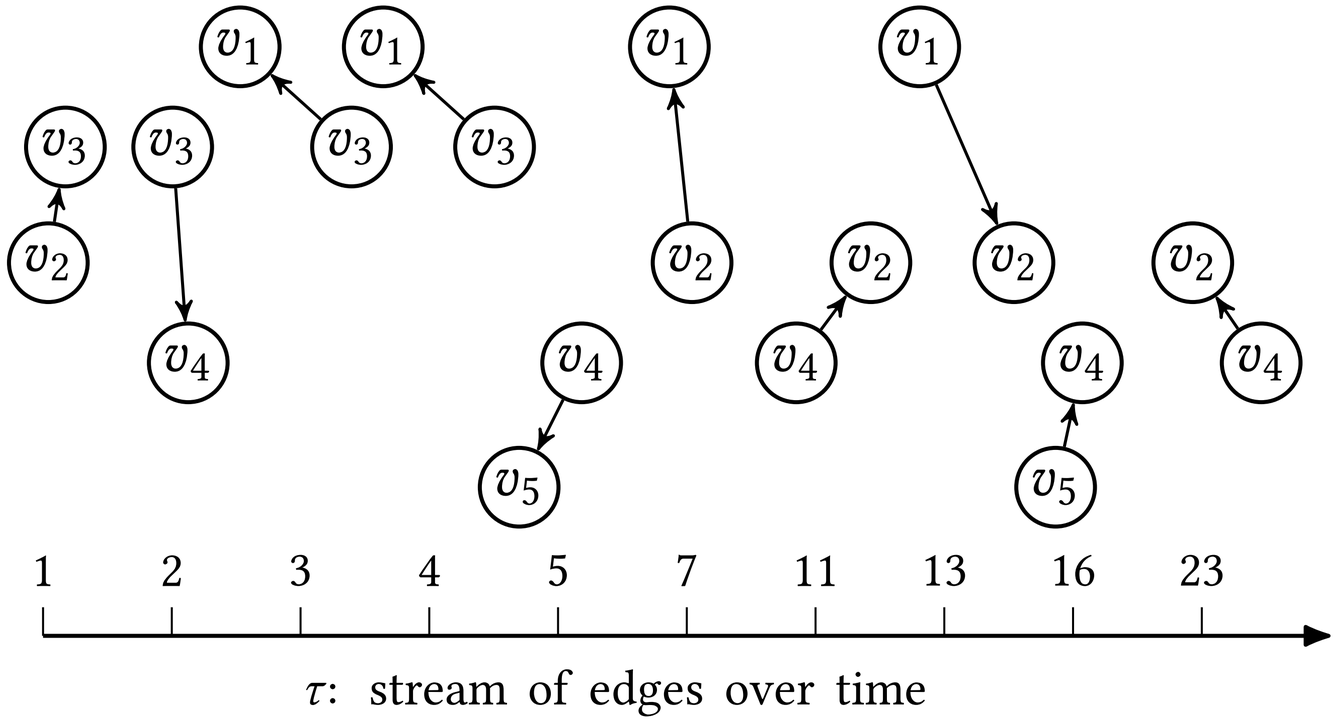
$\delta = 20$

Output



Streaming model

We can access only **one edge** at a time



Previous works

Exact:

- Pashanasangi and Seshadhri [1]: exact method based on the **degeneracy** of the static graph
- Gao et al. [2] and Li et al. [3]: **Fast-Tri** and **MoTTo**, two exact algorithms exploiting pruning techniques to improve efficiency
- Xia et al [5]: **OTTC** exact algorithm for counting all temporal triangles in streaming

[1] *Pashanasangi, Noujan, and C. Seshadhri. SIGKDD, 2021.*
[2] *Gao, Zhongqiang, et al. ICDE, 2022.*
[3] *Li, Jiantao, et al. CIKM, 2024.*
[4] *Wang, Jingjing, et al. CIKM, 2020.*
[5] *Xia, Yuyiang, et al. PACMMOD, 2025.*

Previous works

Exact:

- Pashanasangi and Seshadhri [1]: exact method based on the **degeneracy** of the static graph
- Gao et al. [2] and Li et al. [3]: **Fast-Tri** and **MoTTo**, two exact algorithms exploiting pruning techniques to improve efficiency
- Xia et al [5]: **OTTC** exact algorithm for counting all temporal triangles in streaming

Issues

- Graph **degeneracy** grows significantly in **large graphs**, becoming **impractical**
- Exact approaches are **memory-intensive** and most of them do not work in **streaming**

[1] *Pashanasangi, Noujan, and C. Seshadhri. SIGKDD, 2021.*
[2] *Gao, Zhongqiang, et al. ICDE, 2022.*
[3] *Li, Jiantao, et al. CIKM, 2024.*
[4] *Wang, Jingjing, et al. CIKM, 2020.*
[5] *Xia, Yuyiang, et al. PACMMOD, 2025.*

Previous works

Exact:

- Pashanasangi and Seshadhri [1]: exact method based on the **degeneracy** of the static graph
- Gao et al. [2] and Li et al. [3]: **Fast-Tri** and **MoTTo**, two exact algorithms exploiting pruning techniques to improve efficiency
- Xia et al [5]: **OTTC** exact algorithm for counting all temporal triangles in streaming

Issues

- Graph **degeneracy** grows significantly in **large graphs**, becoming **impractical**
- Exact approaches are **memory-intensive** and most of them do not work in **streaming**

Approximate:

- Wang et al. [4]: **EWS**, approximation streaming algorithm exploiting uniform edge and wedge sampling

[1] *Pashanasangi, Noujan, and C. Seshadhri. SIGKDD, 2021.*
[2] *Gao, Zhongqiang, et al. ICDE, 2022.*
[3] *Li, Jiantao, et al. CIKM, 2024.*
[4] *Wang, Jingjing, et al. CIKM, 2020.*
[5] *Xia, Yuyiang, et al. PACMMOD, 2025.*

Previous works

Exact:

- Pashanasangi and Seshadhri [1]: exact method based on the **degeneracy** of the static graph
- Gao et al. [2] and Li et al. [3]: **Fast-Tri** and **MoTTo**, two exact algorithms exploiting pruning techniques to improve efficiency
- Xia et al [5]: **OTTC** exact algorithm for counting all temporal triangles in streaming

Issues

- Graph **degeneracy** grows significantly in **large graphs**, becoming **impractical**
- Exact approaches are **memory-intensive** and most of them do not work in **streaming**

Approximate:

- Wang et al. [4]: **EWS**, approximation streaming algorithm exploiting uniform edge and wedge sampling

Issues:

- **Cannot scale** to massive temporal graphs and estimates only **one triangle** at a time

[1] *Pashanasangi, Noujan, and C. Seshadhri. SIGKDD, 2021.*
[2] *Gao, Zhongqiang, et al. ICDE, 2022.*
[3] *Li, Jiantao, et al. CIKM, 2024.*
[4] *Wang, Jingjing, et al. CIKM, 2020.*
[5] *Xia, Yuyiang, et al. PACMMOD, 2025.*

Challenges

Challenges

Solving our problem in **large** temporal graphs is hard:

- Extremely **resource-demanding** (both time and memory)
- **No random access** to the temporal graph
- **Minimize passes** on the stream with **small memory budget**
- When resorting to **approximations**, estimates have **high variance**

Challenges

Solving our problem in **large** temporal graphs is hard:

- Extremely **resource-demanding** (both time and memory)
- **No random access** to the temporal graph
- **Minimize passes** on the stream with **small memory budget**
- When resorting to **approximations**, estimates have **high variance**

Our goals:

- Compute estimates for **all** temporal triangles **efficiently**

- Provide highly **accurate** estimates

Challenges

Solving our problem in **large** temporal graphs is hard:

- Extremely **resource-demanding** (both time and memory)
- **No random access** to the temporal graph
- **Minimize passes** on the stream with **small memory budget**
- When resorting to **approximations**, estimates have **high variance**

Our goals:

- Compute estimates for **all** temporal triangles **efficiently**

→ design of a **one-pass** streaming algorithm to estimate all triangles **simultaneously**

- Provide highly **accurate** estimates

→ exploit **predictions** to improve estimates quality

Contributions

Contributions

Design of **STEP**, a randomized, single-pass streaming algorithm using **predictions** to estimate all temporal triangle counts **simultaneously**

Contributions

Design of **STEP**, a randomized, single-pass streaming algorithm using **predictions** to estimate all temporal triangle counts **simultaneously**

Theoretical **analysis** on STEP **estimates** quality (unbiasedness, variance), **memory** requirements under perfect and noisy **predictions**

Contributions

Design of **STEP**, a randomized, single-pass streaming algorithm using **predictions** to estimate all temporal triangle counts **simultaneously**

Theoretical **analysis** on STEP **estimates** quality (unbiasedness, variance), **memory** requirements under perfect and noisy **predictions**

Design of a **practical**, efficient and domain-agnostic **predictor**

Contributions

Design of **STEP**, a randomized, single-pass streaming algorithm using **predictions** to estimate all temporal triangle counts **simultaneously**

Theoretical **analysis** on STEP **estimates** quality (unbiasedness, variance), **memory** requirements under perfect and noisy **predictions**

Design of a **practical**, efficient and domain-agnostic **predictor**

Our experimental evaluation of STEP shows that it **outperforms** state-of-the-art approaches and **scales** to temporal graphs with billions of edges

STEP overview

STEP overview

STEP (***S***treaming-based temporal ***T***riangle ***E***stimation with ***P***redictions) is composed of two main phases:

- **Sampling edges of the graph stream**
- **Estimating temporal triangle counts**

STEP overview

STEP (***S***treaming-based temporal ***T***riangle ***E***stimation with ***P***redictions) is composed of two main phases:

- **Sampling edges of the graph stream**
- **Estimating temporal triangle counts**

Predictions are used to:

- Retain important **edges involved in many triangles** (*heavy edges*)
- **Reduce error of estimates** computed over retained edges

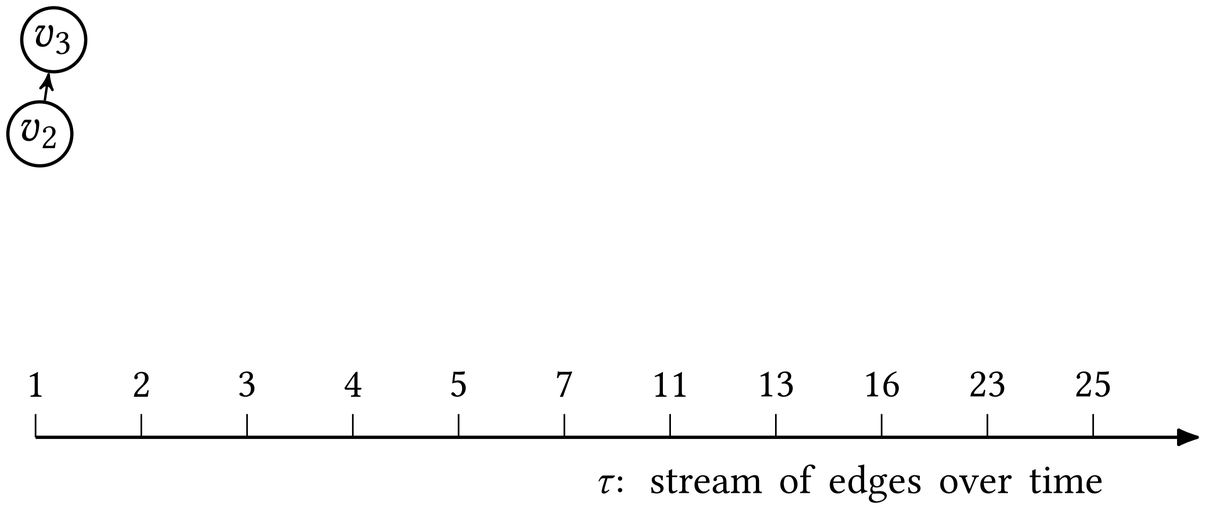
STEP overview

Sampling edges of the graph stream: for each incoming edge, STEP use the predictor to classify the edge: if **heavy**, it is kept in the sample deterministically; otherwise, it is **sampled** with fixed probability p

Sample

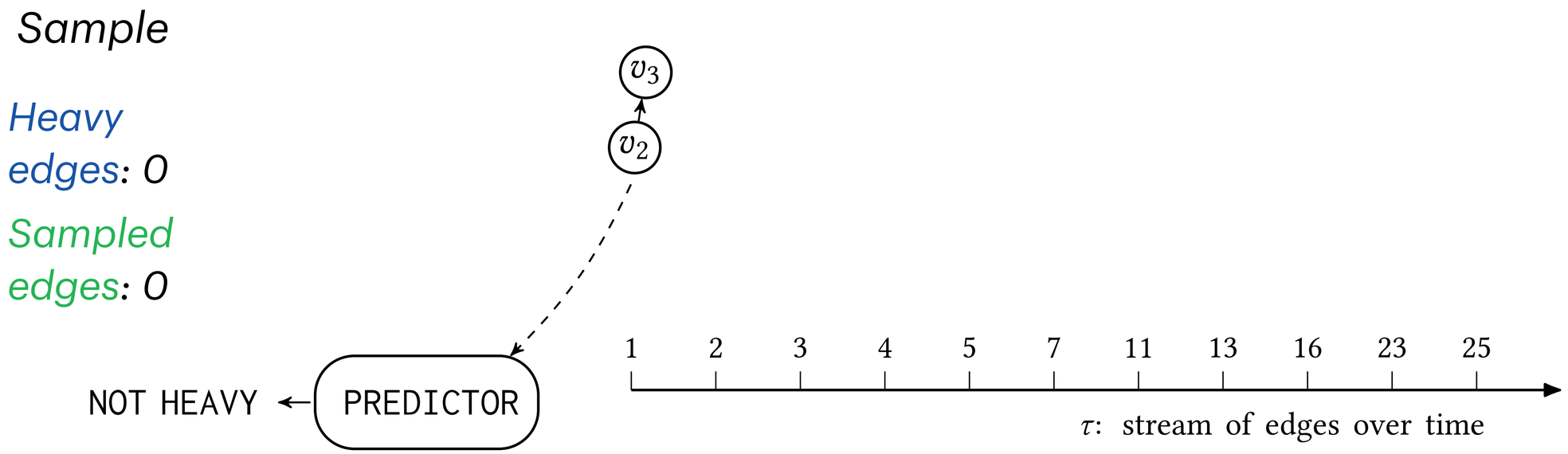
Heavy edges: 0

Sampled edges: 0



STEP overview

Sampling edges of the graph stream: for each incoming edge, STEP use the predictor to classify the edge: if **heavy**, it is kept in the sample deterministically; otherwise, it is **sampled** with fixed probability p



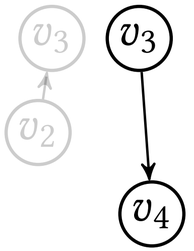
STEP overview

Sampling edges of the graph stream: for each incoming edge, STEP use the predictor to classify the edge: if **heavy**, it is kept in the sample deterministically; otherwise, it is **sampled** with fixed probability p

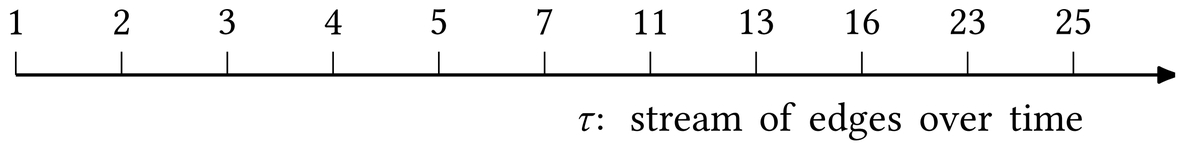
Sample

Heavy edges: 0

Sampled edges: 0

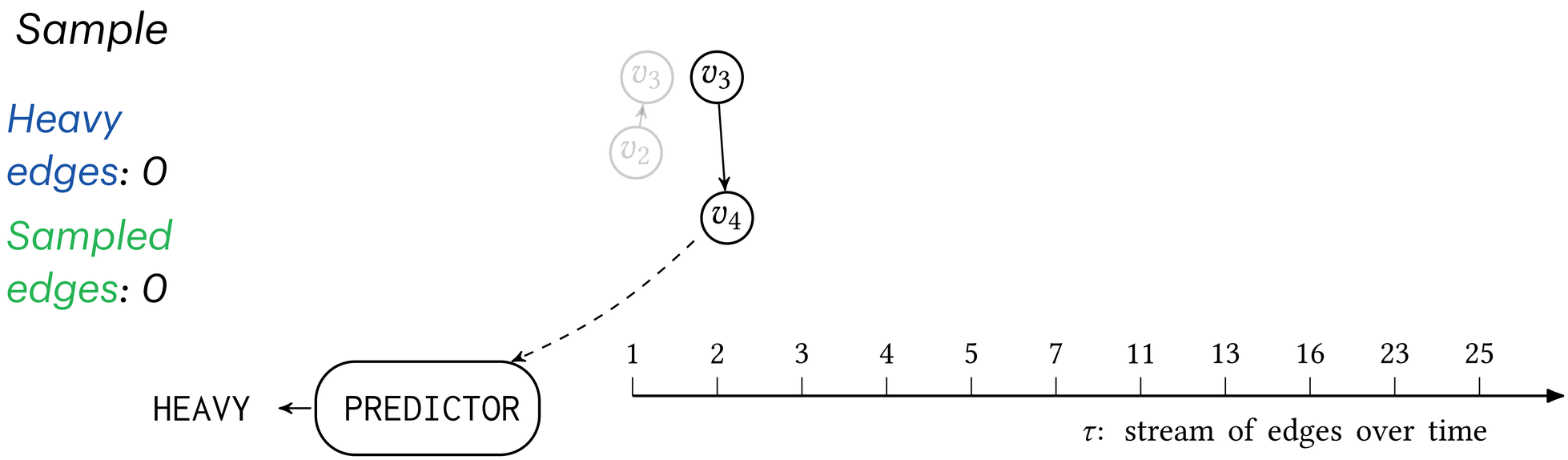


PREDICTOR



STEP overview

Sampling edges of the graph stream: for each incoming edge, STEP use the predictor to classify the edge: if **heavy**, it is kept in the sample deterministically; otherwise, it is **sampled** with fixed probability p



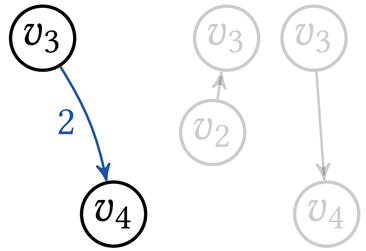
STEP overview

Sampling edges of the graph stream: for each incoming edge, STEP use the predictor to classify the edge: if **heavy**, it is kept in the sample deterministically; otherwise, it is **sampled** with fixed probability p

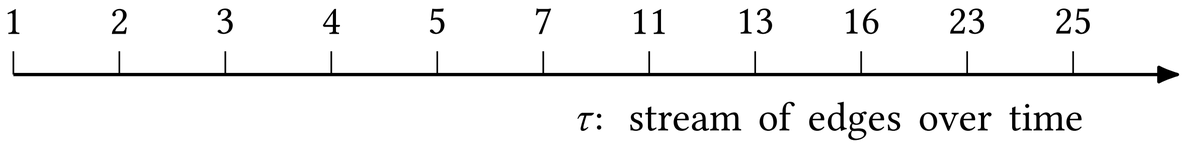
Sample

Heavy edges: 1

Sampled edges: 0

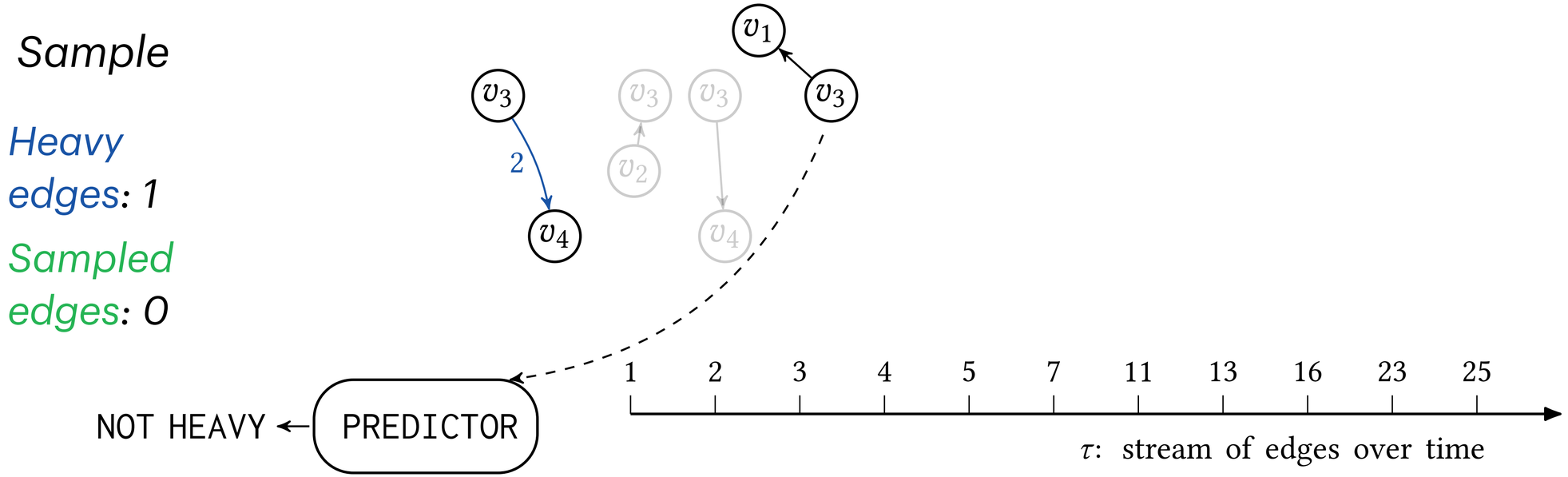


PREDICTOR



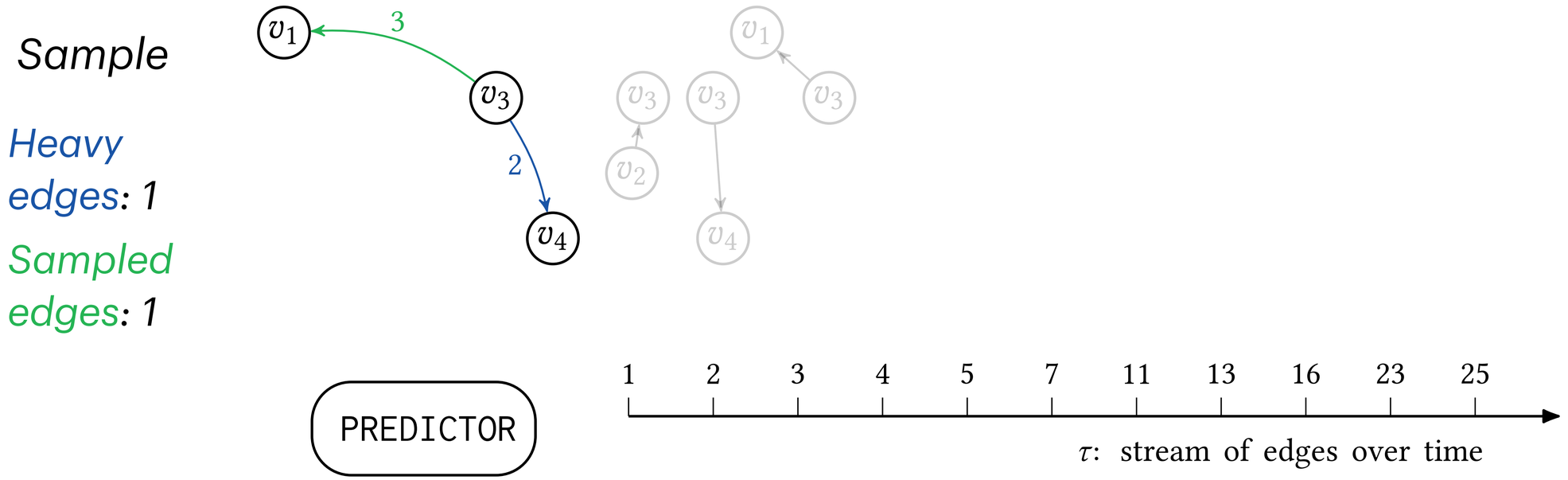
STEP overview

Sampling edges of the graph stream: for each incoming edge, STEP use the predictor to classify the edge: if **heavy**, it is kept in the sample deterministically; otherwise, it is **sampled** with fixed probability p



STEP overview

Sampling edges of the graph stream: for each incoming edge, STEP use the predictor to classify the edge: if **heavy**, it is kept in the sample deterministically; otherwise, it is **sampled** with fixed probability p



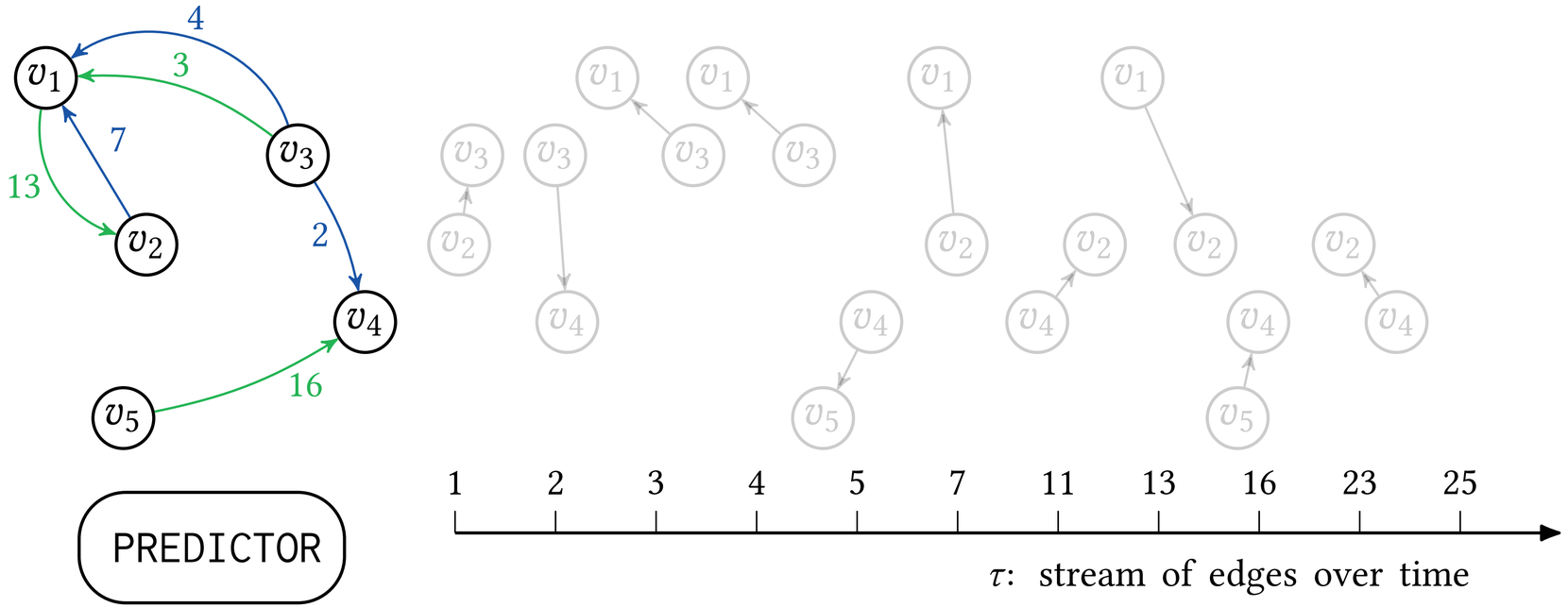
STEP overview

Sampling edges of the graph stream: for each incoming edge, STEP use the predictor to classify the edge: if **heavy**, it is kept in the sample deterministically; otherwise, it is **sampled** with fixed probability p

Sample

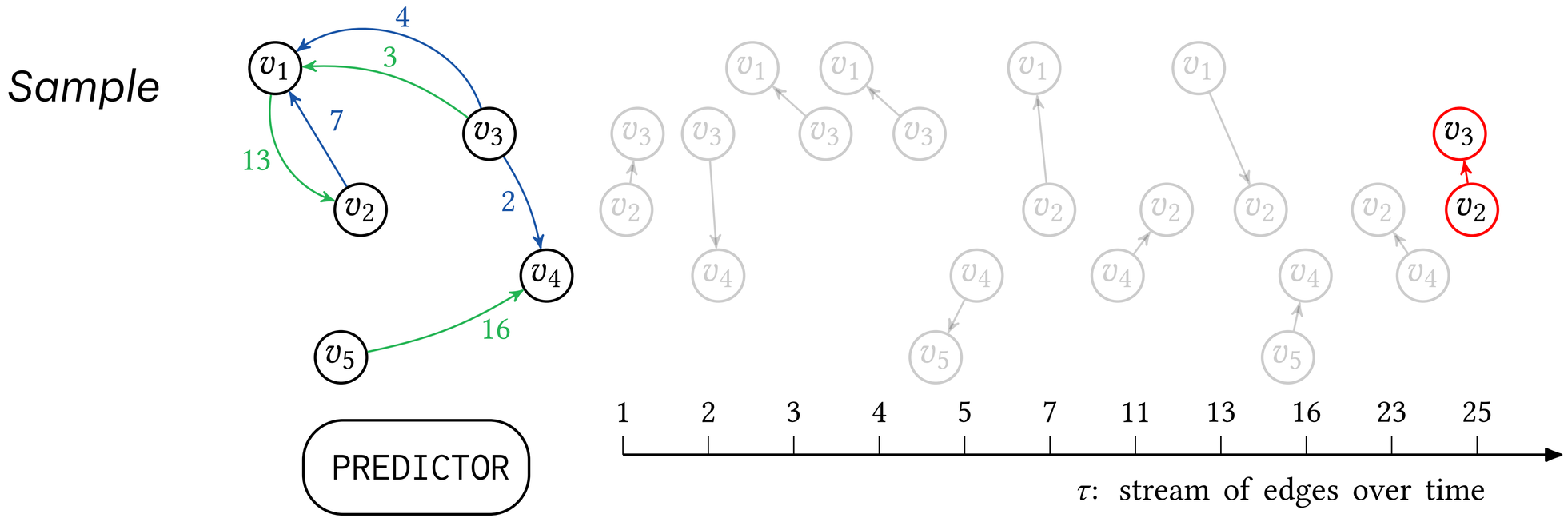
Heavy edges: 3

Sampled edges: 3



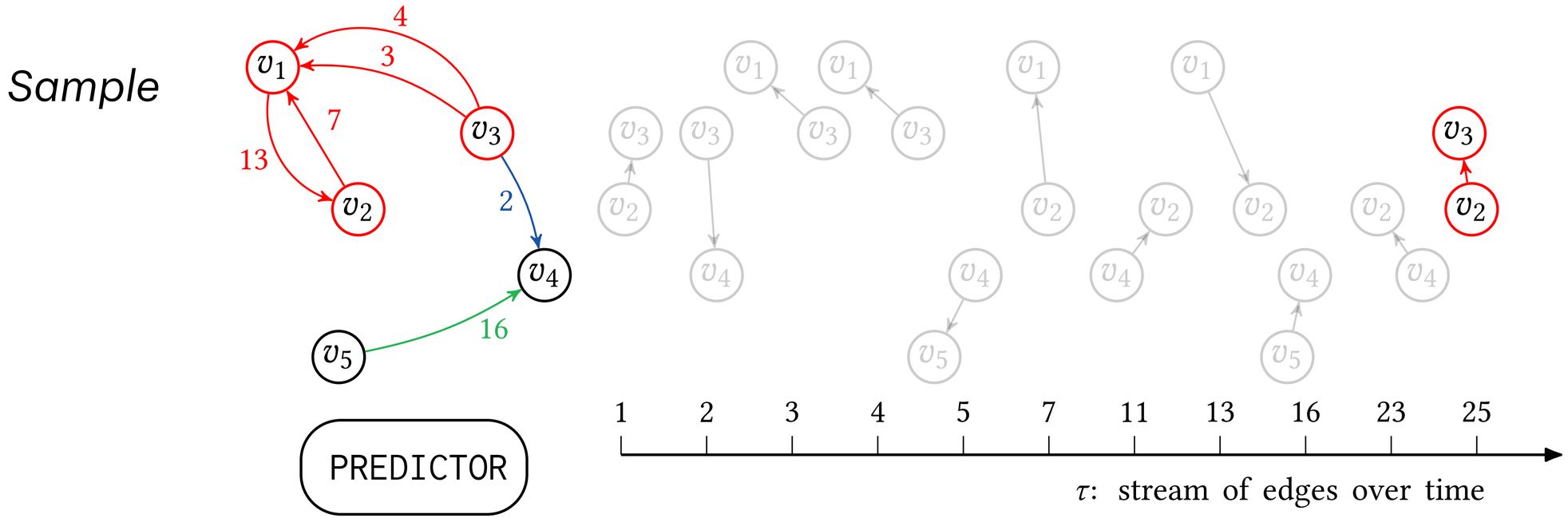
STEP overview

Estimating temporal triangle counts: for each incoming edge, STEP collects pairs of edges from its sample and checks for new temporal triangles instances



STEP overview

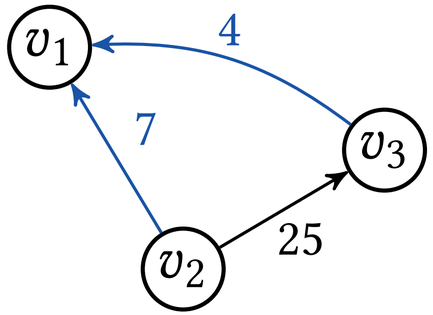
Estimating temporal triangle counts: for each incoming edge, STEP collects pairs of edges from its sample and checks for new temporal triangles instances

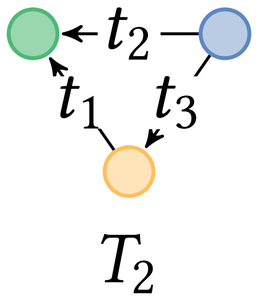


STEP overview

Estimating temporal triangle counts: for each incoming edge, STEP collects pairs of edges from its sample and checks for new temporal triangles instances

Estimate c_i increments by 1 if the triangle has 2 **heavy** edges





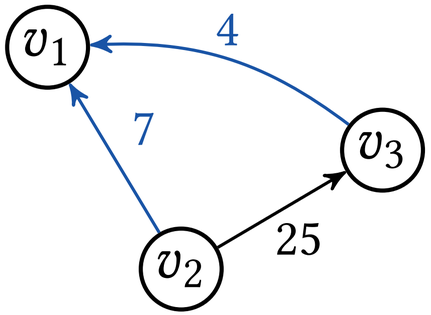
T_2

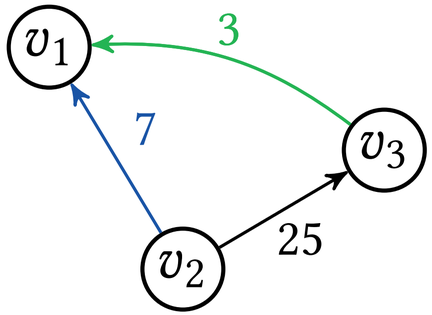
$c_2 \leftarrow 1$

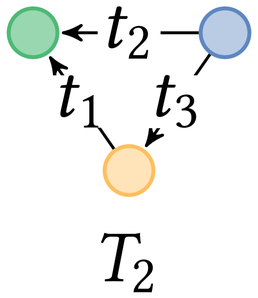
STEP overview

Estimating temporal triangle counts: for each incoming edge, STEP collects pairs of edges from its sample and checks for new temporal triangles instances

Estimate c_i increments by $1/p$ if the triangle has 1 **heavy** edge and 1 **sampled** edge





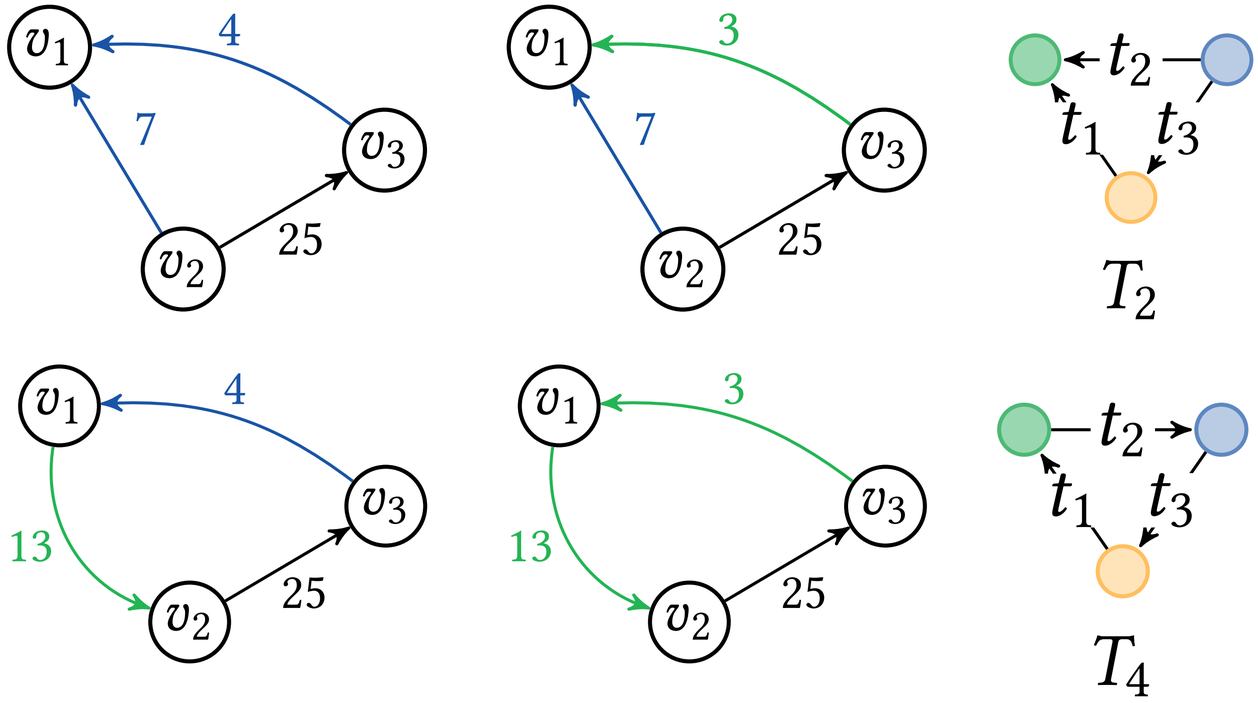


$$c_2 \leftarrow 1 + \frac{1}{p}$$

STEP overview

Estimating temporal triangle counts: for each incoming edge, STEP collects pairs of edges from its sample and checks for new temporal triangles instances

Estimate c_i increments by $1/p^2$ if the triangle has 2 **sampled** edges



$$c_2 \leftarrow 1 + \frac{1}{p}$$

$$c_4 \leftarrow \frac{1}{p} + \frac{1}{p^2}$$

Our predictor

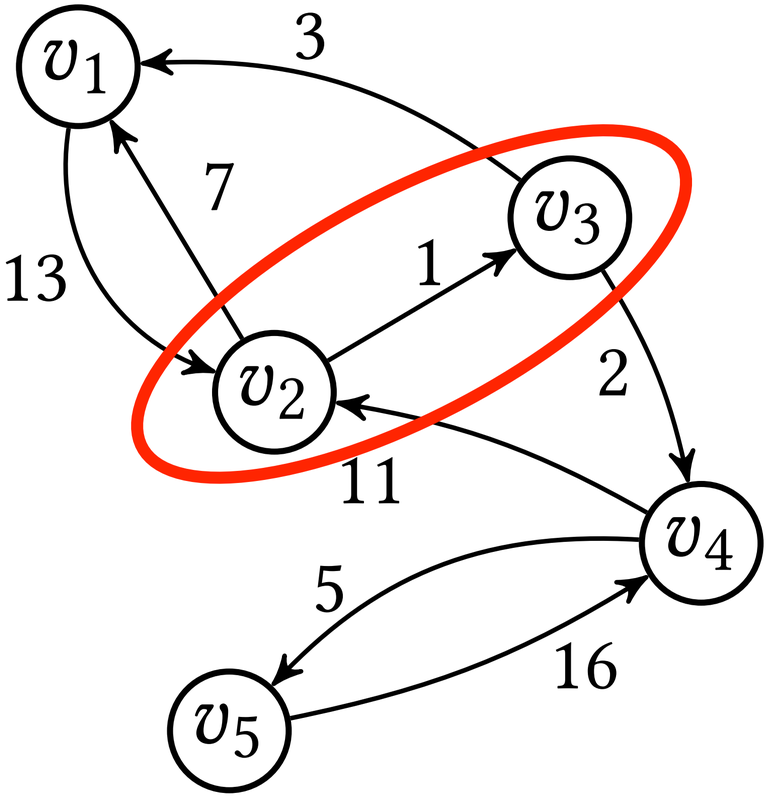
Our predictor

Practical predictor exploiting **node degrees** and **time** information

Design goal: identify **edges** contained in **many** δ -instances — avoid exact computation of temporal triangle counts

Idea: **edge** is likely to form triangles if its nodes have a high number of incident edges in a δ time window

Can be built efficiently in **one-pass** over the temporal graph stream!



Our predictor: Details

Temporal degree of a node $u \in V$ within a time interval $[t_a, t_b]$:

$$d(u, t_a, t_b) = |\{(x, y, t') \in E : (x = u \vee y = u), t' \in [t_a, t_b]\}|$$

Temporal minimum degree weight of an edge $e = (u, v, t) \in E$ for a given time duration δ:

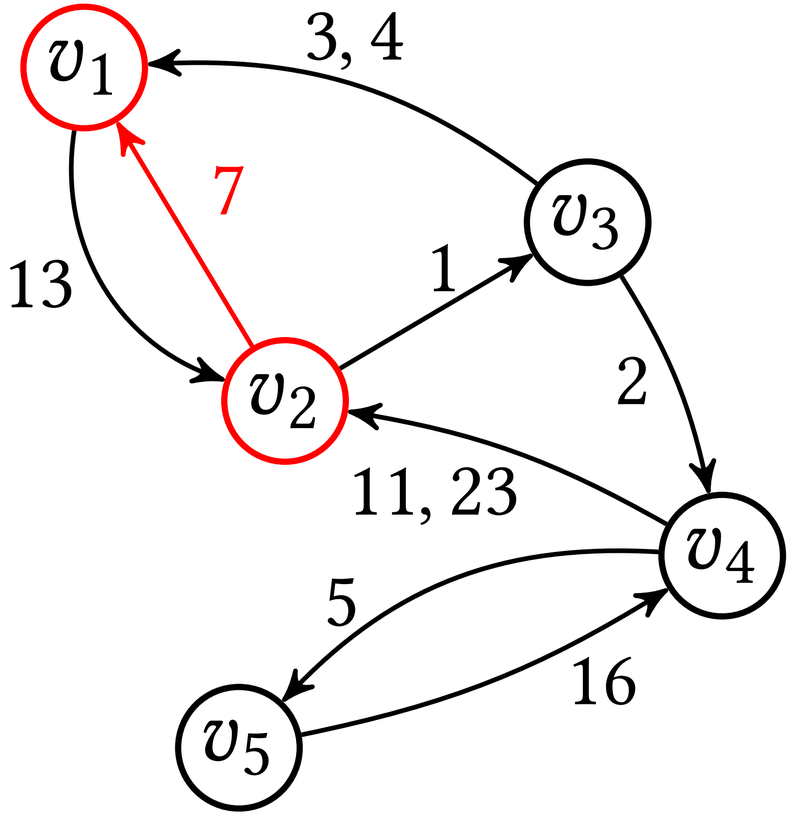
$$w_{m-d}(e) = min\{d(u, t - \delta, t + \delta), d(v, t - \delta, t + \delta)\}$$

The **temporal min-degree predictor** classifies an edge as heavy if the edge is among the top K edges ordered by the temporal minimum degree weight

Our predictor: Example

Example: at time $t = 7$ and for time duration $\delta = 5$:

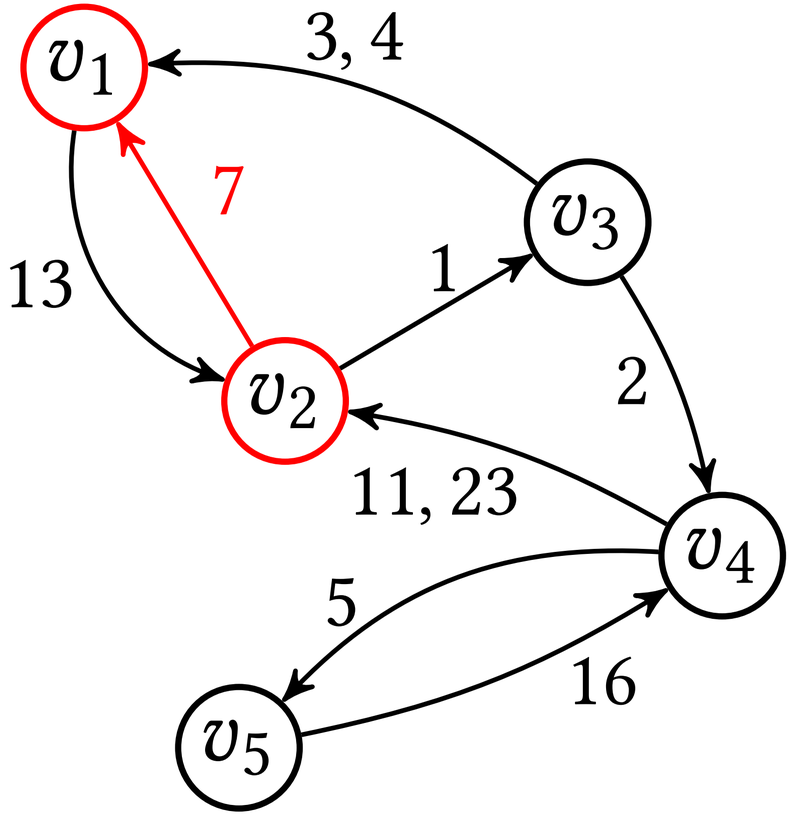
- node v_2 has a temporal degree of 2 within the time interval [2, 12]
- node v_1 has a temporal degree of 3 within the time interval [2, 12]
- edge $(v_2, v_1, 7)$ has temporal min-degree weight equal to 2



Our predictor: Example

Example: at time $t = 7$ and for time duration $\delta = 5$:

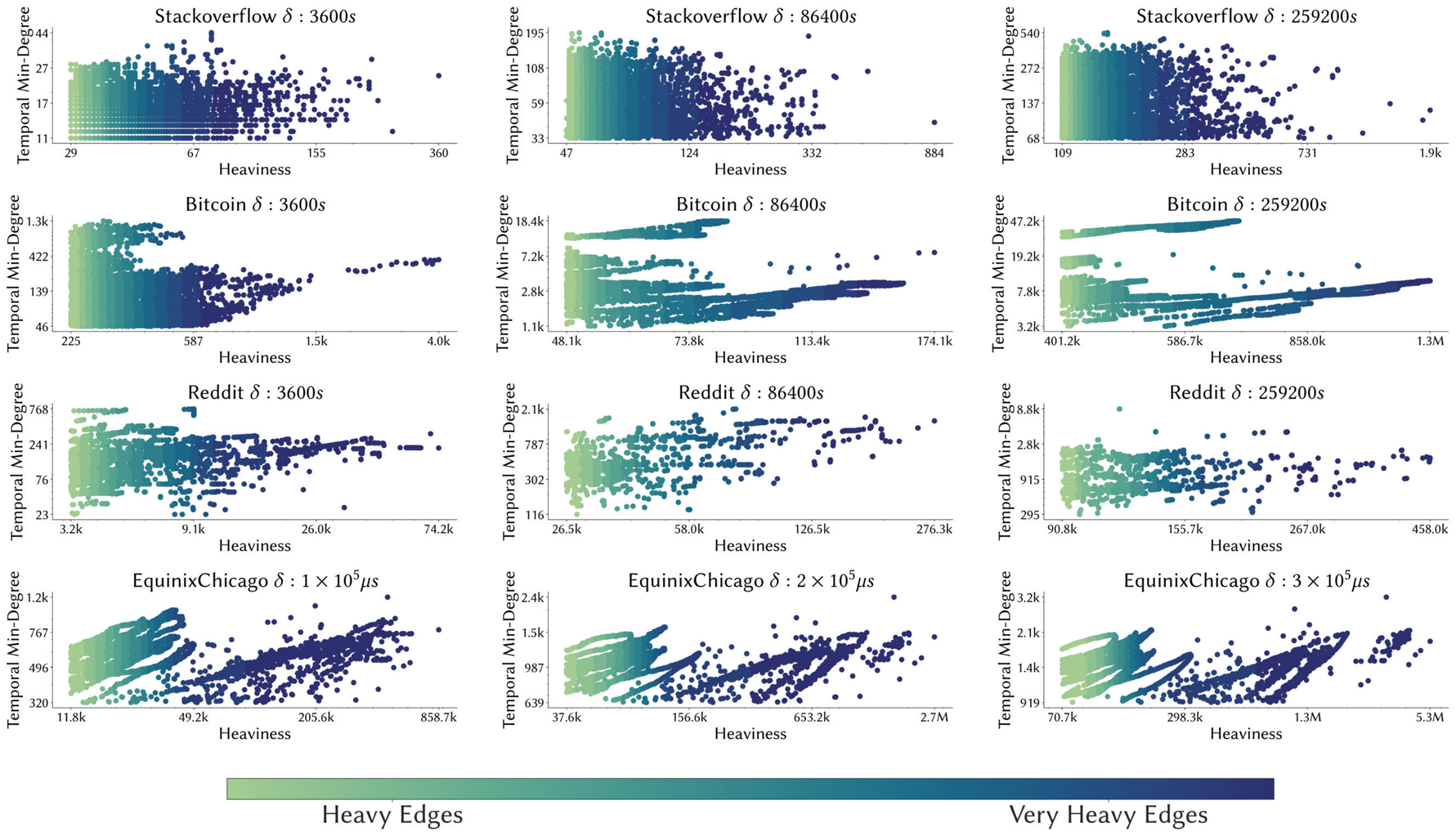
- node v_2 has a temporal degree of 2 within the time interval [2, 12]
- node v_1 has a temporal degree of 3 within the time interval [2, 12]
- edge $(v_2, v_1, 7)$ has temporal min-degree weight equal to 2



Our predictor vs perfect predictor

Correlation between the perfect predictor and the temporal min-degree predictor

Each point represents a temporal edge with a given heaviness (x-axis) and a corresponding minimum temporal degree (y-axis)



STEP analysis

STEP analysis: perfect predictor

Variance (informal): under a perfect predictor, the variance of estimates $c_i, i \in \{1,\ldots,8\}$ is:

$$Var[c_i] \in \mathcal{O}(p^{-2} m_\delta |\mathcal{T}_i|)$$

where m_δ is the maximum number of edges in a δ time-window and $|\mathcal{T}_i|$ is the number of δ-instances of triangle T_i.

STEP analysis: perfect predictor

Variance (informal): under a perfect predictor, the variance of estimates $c_i, i \in \{1,\ldots,8\}$ is:

$$Var[c_i] \in \mathcal{O}(p^{-2} m_\delta |\mathcal{T}_i|)$$

where m_δ is the maximum number of edges in a δ time-window and $|\mathcal{T}_i|$ is the number of δ-instances of triangle T_i. **Without** a predictor, the variance is:

$$Var[c_i] \in \mathcal{O}(p^{-2} m_\delta |\mathcal{T}_i|^2)$$

STEP analysis: perfect predictor

Variance (informal): under a perfect predictor, the variance of estimates $c_i, i \in \{1,\ldots,8\}$ is:

$$Var[c_i] \in \mathcal{O}(p^{-2} m_\delta |\mathcal{T}_i|)$$

where m_δ is the maximum number of edges in a δ time-window and $|\mathcal{T}_i|$ is the number of δ-instances of triangle T_i. **Without** a predictor, the variance is:

$$Var[c_i] \in \mathcal{O}(p^{-2} m_\delta |\mathcal{T}_i|^2)$$

Memory (informal): if STEP retains $o(m_\delta)$ edges flagged heavy by the predictor and p is large enough, then STEP obtains accurate relative ε-approximation of all temporal triangles simultaneously using expected **sublinear memory** in m_δ

STEP analysis: noisy ranking predictor

Perfect ranking predictor: Given an integer value $K > 0$, a ranking predictor classifies an edge as heavy if it is among the first K heaviest edges, where the heaviness (or weight) of an edge is defined as

$$W(e) = \sum_{i \in \{1,\ldots,8\}} |\{\Delta : e \in \Delta, \Delta \in \mathcal{T}_i\}|$$

STEP analysis: noisy ranking predictor

Perfect ranking predictor: Given an integer value $K > 0$, a ranking predictor classifies an edge as heavy if it is among the first K heaviest edges, where the heaviness (or weight) of an edge is defined as

$$W(e) = \sum_{i \in \{1,\ldots,8\}} |\{\Delta : e \in \Delta, \Delta \in \mathcal{T}_i\}|$$

Noisy ranking predictor: Given $K > 0$ and $0 \leq \alpha \leq min\{m - K + 1, K - 1\}$, an α-noisy K-ranking predictor:

- Classifies correctly the top $K - \alpha - 1$ heaviest edges
- Classifies correctly the lightest edges (i.e., edges with low weight $W(e)$) from position $K + \alpha + 1$ onwards
- Predictions of edges in position $K - \alpha, \ldots, K + \alpha$ can be arbitrarily wrong

STEP analysis: noisy ranking predictor

Perfect ranking predictor: Given an integer value $K > 0$, a ranking predictor classifies an edge as heavy if it is among the first K heaviest edges, where the heaviness (or weight) of an edge is defined as

$$W(e) = \sum_{i \in \{1,\ldots,8\}} |\{\Delta : e \in \Delta, \Delta \in \mathcal{T}_i\}|$$

Noisy ranking predictor: Given $K > 0$ and $0 \leq \alpha \leq min\{m - K + 1, K - 1\}$, an α-noisy K-ranking predictor:

- Classifies correctly the top $K - \alpha - 1$ heaviest edges
- Classifies correctly the lightest edges (i.e., edges with low weight $W(e)$) from position $K + \alpha + 1$ onwards
- Predictions of edges in position $K - \alpha, \ldots, K + \alpha$ can be arbitrarily wrong

We also denote with $\mathbf{W}$ the list of edges in E ordered by non-increasing values of their weights $W(e)$

STEP analysis: noisy ranking predictor

Theorem: Consider an execution of STEP, with $\epsilon > 0$ and $K = o(m_\delta)$. There exist constants $C > 1, \gamma \in \left(0, \frac{1}{2}\right)$ such that:

1. When $\mathcal{Q}(\cdot)_K$ is a ranking predictor and $p \geq \frac{C\sqrt{m_\delta}}{\epsilon\,|\mathcal{T}_i|^{1/2-\gamma}}$, STEP uses $\mathcal{O}\left(\frac{\epsilon^{-1} m_\delta^{3/2}}{|\mathcal{T}_i|^{1/2-\gamma}}\right)$ memory in expectation;

2. When $\mathcal{Q}(\cdot)_K$ is an α-noisy K-ranking predictor for $\alpha \geq 1$ and $p \geq \frac{C\sqrt{m_\delta \nabla_\alpha}}{\epsilon\,|\mathcal{T}_i|^{1/2-\gamma}}$, STEP uses $\mathcal{O}\left(\frac{\epsilon^{-1} m_\delta^{3/2}\sqrt{\nabla_\alpha}}{|\mathcal{T}_i|^{1/2-\gamma}}\right)$ memory in expectation, where ∇_α is the maximum difference between the weights of two edges that are adjacents in $\mathbf{W}$ multiplied by $(\alpha + 1)$

In both cases, it holds $\mathbb{P}[\exists i \in \{1,\ldots,8\} : |c_i - |\mathcal{T}_i|| \geq \epsilon\,|\mathcal{T}_i|] \leq 1/3$ using sublinear memory.

Experimental evaluation

Experimental evaluation

Our experimental evaluation answers the following points:

1. How does STEP compare to state-of-the-art methods in terms of **accuracy** and **computational resources**?

Experimental evaluation

Our experimental evaluation answers the following points:

1. How does STEP compare to state-of-the-art methods in terms of **accuracy** and **computational resources**?

2. What is the **impact of the predictor** on STEP estimates?

Experimental evaluation

Our experimental evaluation answers the following points:

1. How does STEP compare to state-of-the-art methods in terms of **accuracy** and **computational resources**?

2. What is the **impact of the predictor** on STEP estimates?

3. How does STEP performs in an **online** setting (predictor from historical data)?

Experimental evaluation

Six baselines to compare STEP with:

- **Degeneracy [1]**: exact algorithm to count all temporal triangles based on degeneracy ordering
- **Fast-Tri [2]**: exact algorithm tailored to temporal triangles
- **MoTTo [3]**: exact algorithm for counting 3-nodes 3-edges motifs, including triangles
- **EWS [4]**: state-of-the-art sampling algorithm, provides an approximate solution to the temporal triangle counting problem
- **OTTC [5]**: exact algorithm for counting all temporal triangles in streaming
- **NAIVE-S**: exploits the sampling approach of STEP without any predictor

[1] *Pashanasangi, Noujan, and C. Seshadhri. SIGKDD, 2021.*
[2] *Gao, Zhongqiang, et al. ICDE, 2022.*
[3] *Li, Jiantao, et al. CIKM, 2024.*
[4] *Wang, Jingjing, et al. arXiv preprint, 2022.*
[5] *Xia, Yuyiang, et al. PACMMOD, 2025.*

Experimental evaluation

Two predictors for STEP:

Experimental evaluation

Two predictors for STEP:

- **Perfect predictor**: exactly knows how many triangles an edge is involved in; STEP$_P$ denotes STEP coupled with such predictor

Experimental evaluation

Two predictors for STEP:

- **Perfect predictor**: exactly knows how many triangles an edge is involved in; STEP$_P$ denotes STEP coupled with such predictor
- **Our practical predictor**: exploits degrees and time information to classify important edges; STEP$_{TMD}$ denotes STEP coupled with such predictor

Experimental evaluation

Four of the **largest** available temporal graphs

Dataset	n	m	*precision*	*timespan*
Stackoverflow (SO)	$2.6\,M$	$63.5\,M$	sec	7.60 years
Bitcoin (BI)	$48.1\,M$	$113.1\,M$	sec	7.08 years
Reddit (RE)	$8.4\,M$	$636.3\,M$	sec	10.06 years
EquinixChicago (EC)	$11.1\,M$	$3.3\,B$	μ-sec	62.00 mins

We tested three values of δ (small, medium, large)

Experimental evaluation

System: Ubuntu 20.04, 2.20 GHz Intel Xeon CPU, 200 GB RAM memory limit

Implementation: C++17, gcc 9.4.0, optimization flags

STEP requires **less memory** (up to 10x times) w.r.t state-of-the-art approaches

Memory in GBs:

Dataset	NAIVE-S	STEP_{P}	STEP_{TMD}	EWS	Degeneracy	FAST-Tri	MoTTo	OTTC
SO	0.78	0.74	0.74	8.04	5.10	5.81	12.07	47.30
BI	1.70	1.81	1.74	27.05	15.40	17.43	34.08	94.10
RE	14.68	14.74	15.53	103.75	71.30	79.59	159.74	148.71
EC	63.46	62.70	63.47	*OOM*	*OOM*	*OOM*	*OOM*	*OOM*

Experimental evaluation

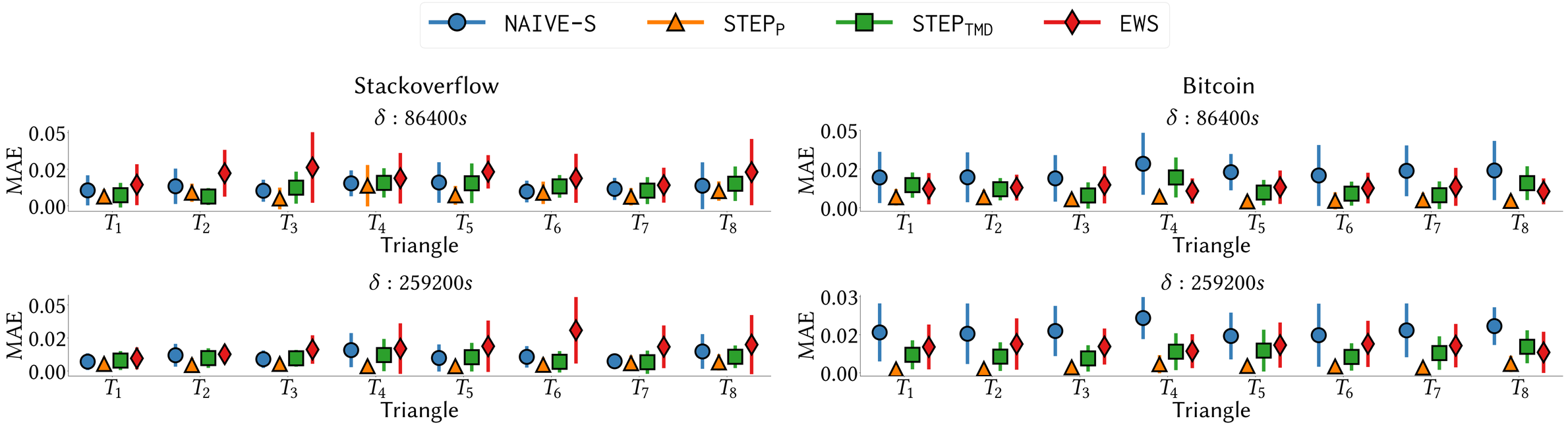
STEP is **orders of magnitude faster** than existing methods

Dataset	δ	STEP$_{TMD}$	EWS		Degeneracy		FAST-Tri		MoTTo		OTTC	
		Time	Time	SU	Time	SU	Time	SU	Time	SU	Time	SU
SO	3600	**4.9 ± 0.0**	30.4 ± 0.8	6.2×	348.4	71.1×	15.1	3.1×	174.5	35.6×	66.6	13.6×
	86400	**6.3 ± 0.1**	32.0 ± 0.5	5.1×	355.2	56.4×	41.8	6.6×	254.5	40.4×	76.5	12.1×
	259200	**7.5 ± 0.1**	35.6 ± 0.9	4.7×	356.3	47.5×	76.3	10.2×	378.9	50.5×	80.7	10.8×
BI	3600	**6.1 ± 0.1**	67.7 ± 5.1	11.1×	422.1	69.2×	189.0	31.0×	1113.7	182.6×	131.3	21.5×
	86400	**43.8 ± 0.4**	115.9 ± 1.7	2.6×	421.3	9.6×	4287.7	97.9×	18045.1	412.0×	146.0	3.3×
	259200	278.2 ± 5.5	198.5 ± 5.9	0.7×	424.8	1.5×	13804.3	49.6×	55494.2	199.5×	**151.5**	0.5×
RE	3600	**69.7 ± 2.0**	570.7 ± 50.3	8.2×	18656.8	267.7×	1708.7	24.5×	6406.0	92.0×	2827.4	40.6×
	86400	**103.5 ± 4.4**	943.1 ± 67.6	9.1×	18528.1	179.0×	7479.5	72.3×	23085.0	223.0×	3131.0	30.3×
	259200	**164.9 ± 2.1**	1121.8 ± 79.1	6.8×	18950.0	115.0×	11165.8	67.7×	29680.1	180.0×	3216.5	19.5×
EC	1×10^5	**425.6 ± 16.8**	✗	N/A	✗	N/A	✗	N/A	✗	N/A	✗	N/A
	2×10^5	**497.4 ± 8.6**	✗	N/A	✗	N/A	✗	N/A	✗	N/A	✗	N/A
	3×10^5	**580.3 ± 0.9**	✗	N/A	✗	N/A	✗	N/A	✗	N/A	✗	N/A

Experimental evaluation

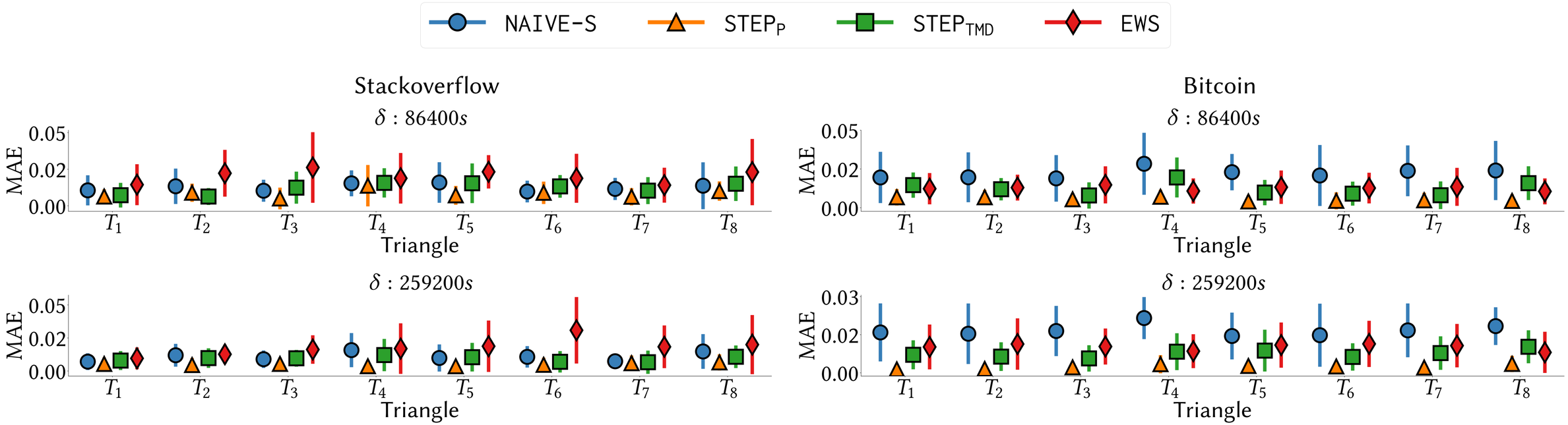
Evaluation of STEP **accuracy** w.r.t. state-of-the-art approximation algorithm EWS

Accuracy of estimates measured using Mean Absolute Error: $|c_i - |\mathscr{T}_i|| \;/\; |\mathscr{T}_i|$



Experimental evaluation

STEP obtains **accurate** estimates for all temporal triangles using less memory and time



Experimental evaluation

Online setting

Learn predictor on training stream $\boldsymbol{\tau}^{tr}$ and use it on test stream $\boldsymbol{\tau}^{ts}$

Experimental evaluation

Online setting

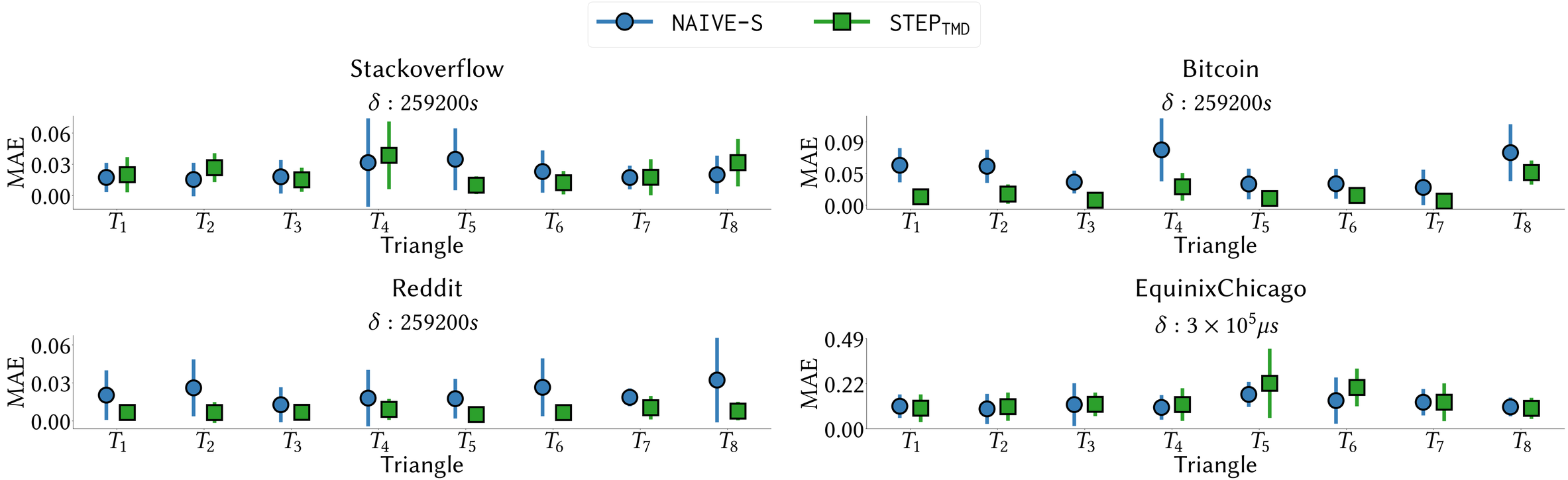
Learn predictor on training stream $\boldsymbol{\tau}^{tr}$ and use it on test stream $\boldsymbol{\tau}^{ts}$

In practice: $\boldsymbol{\tau}^{tr}$ consists of the first 75% edges of the stream $\boldsymbol{\tau}$ (test stream $\boldsymbol{\tau}^{ts}$ is the remaining 25%)

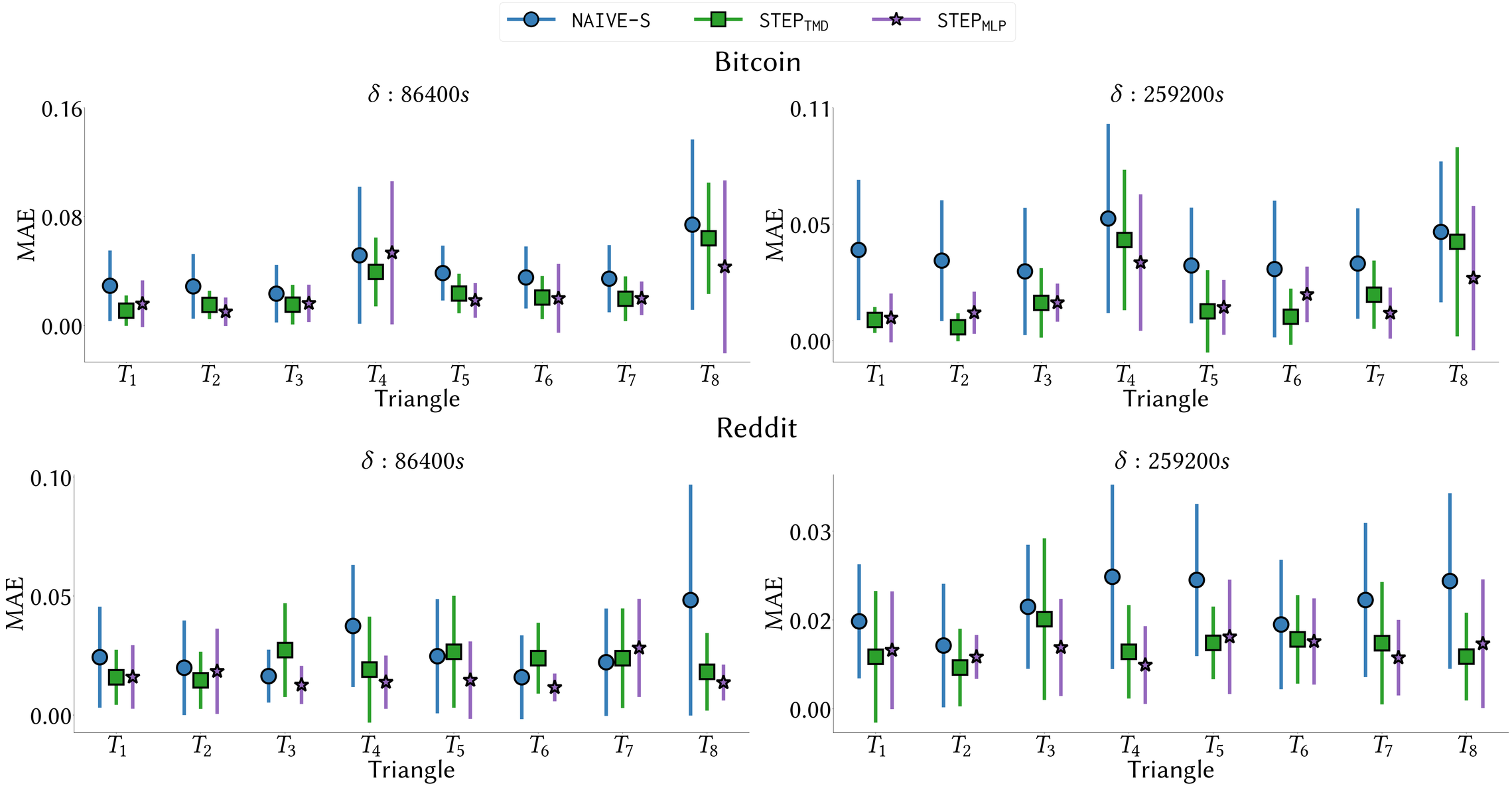
Experimental evaluation

Online setting

STEP outperforms NAIVE-S in most configurations in terms of MAE



Experimental evaluation: NN as predictor



Conclusion

We designed **STEP**, a single-pass streaming algorithm for accurately estimating all temporal triangle counts simultaneously;

We provided a theoretical **analysis** on STEP **estimates** quality (unbiasedness, variance), **memory** requirements under perfect and noisy **predictions**

Designed a **practical**, efficient and domain-agnostic **predictor**

In practice, STEP **outperforms** state-of-the-art methods and **scales** to billion-edges graphs.

Conclusion

G. Venturin, I. Sarpe, F. Vandin, "*Efficient Approximate Temporal Triangle Counting in Streaming with Predictions*", ECML-PKDD 2025.

Code: *https://github.com/VandinLab/STEP*

Paper extended version: *https://www.arxiv.org/pdf/2506.13173*

Contact: *fabio.vandin@unipd.it*

Thanks:

PNRR Project "Research initiatives for innovative technologies and pathways in the health and welfare sector, D.D. 931 of 06/06/2022,PNC0000002 DARE - Digital Lifelong Prevention CUP:B53C22006440001"

PRIN Project n. 2022TS4Y3N "EXPAND: scalable algorithms for EXPloratory Analyses of heterogeneous and dynamic Networked Data".

ERC Advanced Grant REBOUND (834862), and Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.