Neighborhood-**A**ware Scalable **T**emporal Network Representation Learning

Yuhong Luo
University of Massachusetts

Pan Li
Purdue University

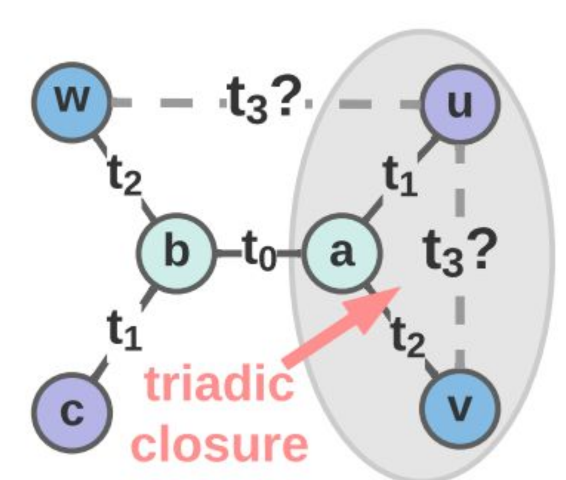
LOG CONFERENCE

Overview

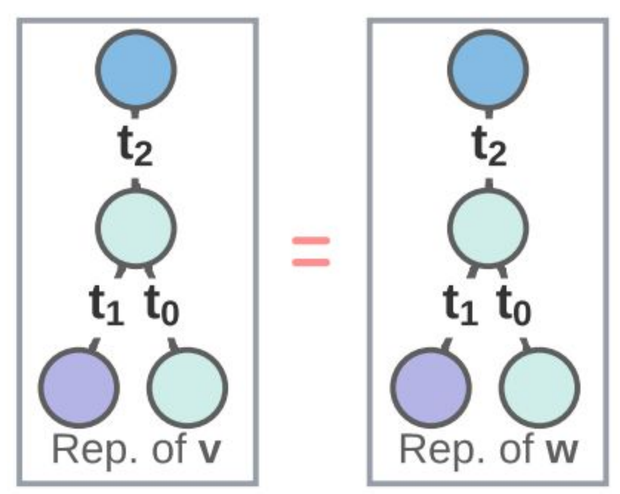
TL; DR: We provide a scalable neighborhood-aware framework NAT, that captures important structural feature in temporal network efficiently.

Motivation

Effectiveness: GNN-based temporal network representation learning cannot capture structural features that involve multiple nodes of interest.



Links arrive in the order of
$t_0 < t_1 < t_2 < t_3$



Traditional representations of **v** and **w** have the same structural contexts

u, **v** are more likely to connect than **u**, **w** at t_3.
GNN-type model will fail because node **w** and node **v** have the same computation graph.
Basically, they fail to capture the structural features in the neighborhood of **u,a,v** that indicates triadic closure.

Scalability: CAWN [Wang+ 2021] captures structural features, but it has serious computation issues.

- Need to **sample** random walks for queried node pairs
- Compute expensive relative positional encoding online

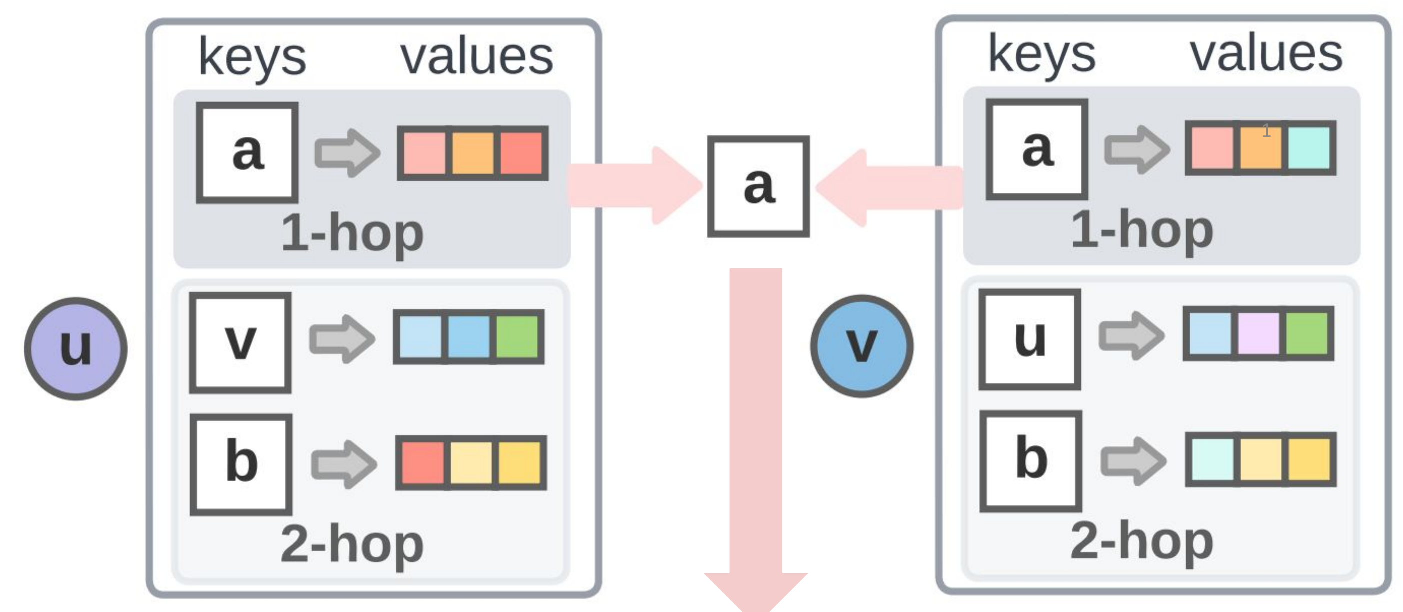
We present a framework that records *Dictionary-type Representations* for nodes, which

1. Constructs structural features efficiently.
2. Avoids online neighbor sampling.
3. Is maintained with ***Neighborhood Caches***.

Dictionary-type Node Representation

Insight: abandons long-vector representations and represents a node as a dictionary.

Keys: Down-sampled neighbors at 0 to k hops.
Values: Short vector representations-dim (2-8) for node pairs, e.g., (**u**, **a**), that summarize past interactions at the k hop between the pairs.



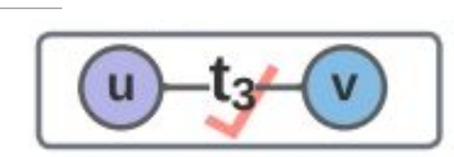
Inference: to predict if there is a link between **u, v**.

1. Construct joint neighborhood structural features
Relative position encoding on keys.
a: [(0,1,0), (0,1,0)].
Denote that this node is in first hop
It indicates that **a** is a common neighbor of **u**, **v**.

2. Aggregate short representations, i.e., the values
Values are aggregated based on the keys.
They work like traditional vector representations.

In parallel:
compute above 2 steps for all nodes (**a, u, v, b**) in the dictionaries of the node pair **(u, v)** of interests.

Make prediction:



Computation benefits
1. No sampling 2. Parallel representation construction

Neighborhood Cache (N-Cache)

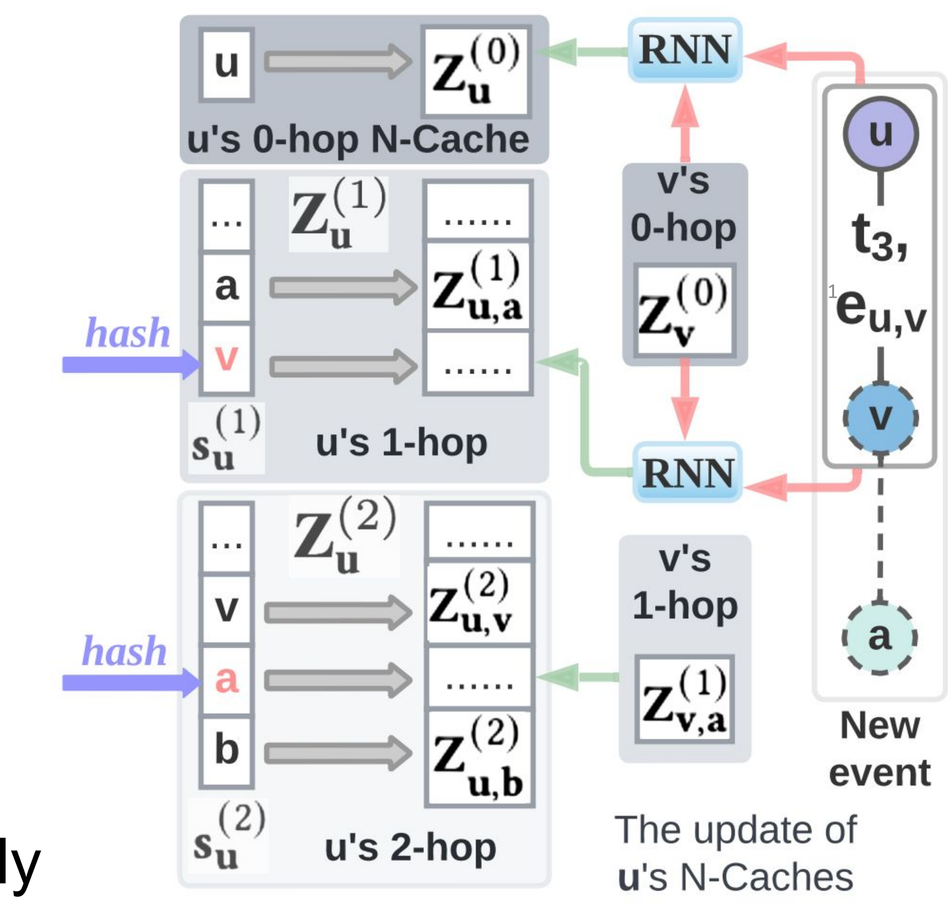
Stores the dictionaries with fixed-size GPU memory and maintains with parallel hashing.

In parallel:

- Encode new link with RNNs
- Hash with key to locate index
- Insert updated representation

OR

- Collision found, replace randomly



Experiments

NAT achieves SOTA prediction performance in both inductive and transductive learnings.

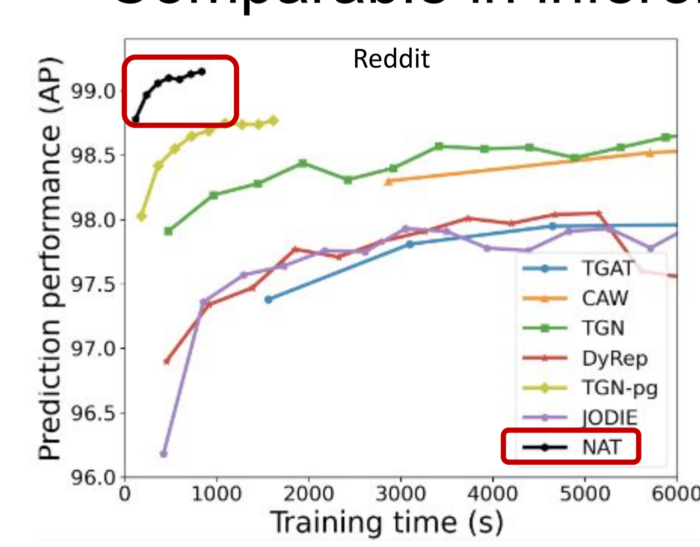
Task	Method	Wikipedia	Reddit	Social E. 1 m.	Social E.	Enron	UCI	Ubuntu	Wiki-talk
Inductive	CAWN	98.52 ± 0.04	98.19 ± 0.03	84.42 ± 1.89	87.71 ± 3.26	93.28 ± 0.01	**93.67 ± 0.65**	50.00 ± 0.00	80.21 ± 7.49
	JODIE	95.58 ± 0.37	95.96 ± 0.29	80.61 ± 1.55	81.13 ± 0.52	81.69 ± 2.21	86.13 ± 0.34	56.68 ± 0.49	65.89 ± 4.72
	DyRep	94.72 ± 0.14	97.04 ± 0.29	81.54 ± 1.81	52.68 ± 0.11	77.44 ± 2.28	68.38 ± 1.30	53.25 ± 0.03	51.87 ± 0.93
	TGN	98.01 ± 0.06	97.76 ± 0.05	86.00 ± 0.70	67.01 ± 10.3	75.72 ± 2.55	83.21 ± 1.16	62.14 ± 3.17	56.73 ± 2.88
	TGN-pg	94.91 ± 0.35	94.34 ± 3.22	63.44 ± 3.54	88.10 ± 4.81	69.55 ± 1.62	86.36 ± 3.60	79.44 ± 0.85	85.35 ± 2.96
	TGAT	97.25 ± 0.18	96.69 ± 0.11	54.66 ± 0.66	50.00 ± 0.00	57.09 ± 0.89	70.47 ± 0.59	54.73 ± 4..94	71.04 ± 3.59
	NAT	**98.55 ± 0.09**	**98.56 ± 0.21**	**91.82 ± 1.91**	**95.16 ± 0.66**	**94.94 ± 1.15**	92.58 ± 1.86	**90.35 ± 0.20**	**93.81 ± 1.16**
Transductive	CAWN	98.62 ± 0.05	98.66 ± 0.09	85.42 ± 0.19	92.81 ± 0.58	91.46 ± 0.35	94.18 ± 0.16	50.00 ± 0.00	85.50 ± 9.70
	JODIE	96.15 ± 0.36	97.29 ± 0.05	77.02 ± 1.11	69.30 ± 0.21	83.42 ± 2.63	91.09 ± 0.69	60.29 ± 2.66	75.00 ± 4.90
	DyRep	95.81 ± 0.15	98.00 ± 0.19	76.96 ± 4.05	51.14 ± 0.24	78.04 ± 2.08	72.25 ± 1.81	52.22 ± 0.02	62.07 ± 0.06
	TGN	98.57 ± 0.05	98.70 ± 0.03	88.72 ± 0.65	69.39 ± 10.50	80.87 ± 4.37	89.53 ± 1.49	53.80 ± 2.23	66.01 ± 4.79
	TGN-pg	97.26 ± 0.10	98.62 ± 0.07	66.39 ± 6.90	64.03 ± 8.97	80.85 ± 2.70	91.47 ± 0.29	90.56 ± 0.44	94.16 ± 0.09
	TGAT	96.65 ± 0.06	98.19 ± 0.08	58.10 ± 0.47	50.00 ± 0.00	61.25 ± 0.99	77.88 ± 0.31	55.46 ± 5.47	78.43 ± 2.15
	NAT	**98.68 ± 0.04**	**99.10 ± 0.09**	**90.20 ± 0.20**	**94.43 ± 1.67**	**92.42 ± 0.09**	**94.37 ± 0.21**	**93.50 ± 0.34**	**95.82 ± 0.31**

Table 2: Performance in average precision (AP) (mean in percentage ± 95% confidence level). **Bold font** and underline highlight the best performance and the second best performance on average.

NAT is fast on large datasets

- Train and converge faster.
- Comparable in inference.

Measurement	Ubuntu	Wiki-talk
nodes	159,316	1,140,149
temporal links	964,437	7,833,140
static links	596,933	3,309,592



	Method	Train	Test	Total	RAM	GPU	Epoch
Ubuntu	CAWN	1,066	222	5,385	38.9	17.4	1.0
	JODIE	6,670	2,860	76,220	35.3	18.7	5.5
	DyRep	2,195	2,857	39,148	38.5	16.6	1.0
	TGN	5,975	2,391	73,633	39	19.6	5.5
	TGN-pg	188.7	**36.5**	3,682	37.0	32.1	11.4
	TGAT	887	330	18,431	47.3	17.0	2.5
	NAT	**125.8**	41.2	**1,321**	28.9	10.1	5.4
Wiki-talk	CAWN	13,685	2,419	34,368	99.1	19.4	1.0
	JODIE	284,789	145,909	566,607	58.2	20.9	1.0
	DyRep	280,659	135,491	514,621	84.4	49.6	1.0
	TGN	281,267	136,780	534,827	77.9	24.1	1.0
	TGN-pg	1,236	311.5	12,761	60.9	59.0	5.1
	TGAT	6,164	2,451	186,513	65.0	17.6	16.0
	NAT	**833.1**	**280.1**	**7,802**	37.1	22.3	2.7