GNN Deployment with High Efficiency

Yucheng Wu

2024/4/12

Contents

- **GNN for online services**
  - Efficient User Sequence Learning for Online Services via Compressed Graph Neural Networks
- **GNN for edge computing**
  - GNN at the Edge: Cost-Efficient Graph Neural Network Processing over Distributed Edge Servers (IEEE Journal on Selected Areas in Communications, 2023)

GNN for online services

Efficient User Sequence Learning for Online Services via Compressed Graph Neural Networks

Anonymous Author(s)
Affiliation
Address
Email

***Abstract*—Learning representations of user behavior sequences is crucial for various online services, such as online fraudulent transaction detection mechanisms. Graph Neural Networks (GNNs) have been extensively applied to model sequence relationships, and extract information from similar sequences. While user behavior sequence data volume is usually huge for online applications, directly applying GNN models may lead to substantial computational overhead during both the training and inference stages and make it challenging to meet real-time requirements for online services. In this paper, we leverage graph compression techniques to alleviate the efficiency issue. Specifically, we propose a novel unified framework called *ECSeq*, to introduce graph compression techniques into relation modeling for user sequence representation learning. The key module of *ECSeq* is *sequence relation modeling*, which explores relationships among sequences to enhance sequence representation learning, and employs graph compression algorithms to achieve high efficiency and scalability. *ECSeq* also exhibits *plug-and-play* characteristics, seamlessly augmenting pre-trained sequence representation models without modifications. Empirical experiments on both sequence classification and regression tasks demonstrate the effectiveness of *ECSeq*. Specifically, with an additional training time of tens of seconds in total on 100,000+ sequences and inference time preserved within 10^{-4} seconds/sample, *ECSeq* improves the prediction R@P$_{0.9}$ of the widely used LSTM by $\sim 5\%$.**

***Index Terms*—Sequence Representation Learning, Graph Neural Network, Graph Compression, Online Inference**

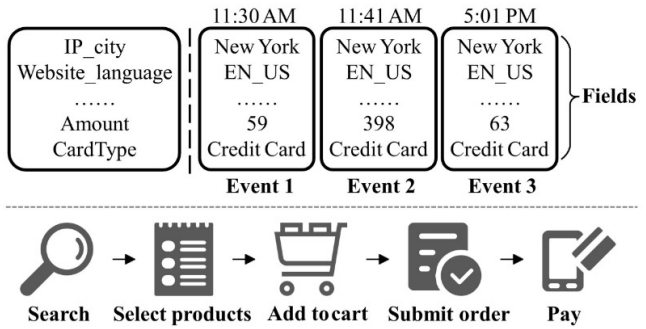


Fig. 1: **Up:** A user behavior sequence example consists of three events and each event has several fields. **Down**: An example of user behavior sequence for online shopping.

proposed to capture correlations among similar sequences for online user behavior modeling [9].

However, most GNN-based sequence representation methods still face challenges with efficiency and scalability [10] for online services. The number of user sequences produced by online applications is often immense, potentially escalating to the magnitude of millions [11], which results in relation graphs with millions of nodes and edges. For GNN training, such

Introduction

- Sequence representation learning
  - map user sequences into embedding vectors
  - conduct predictions

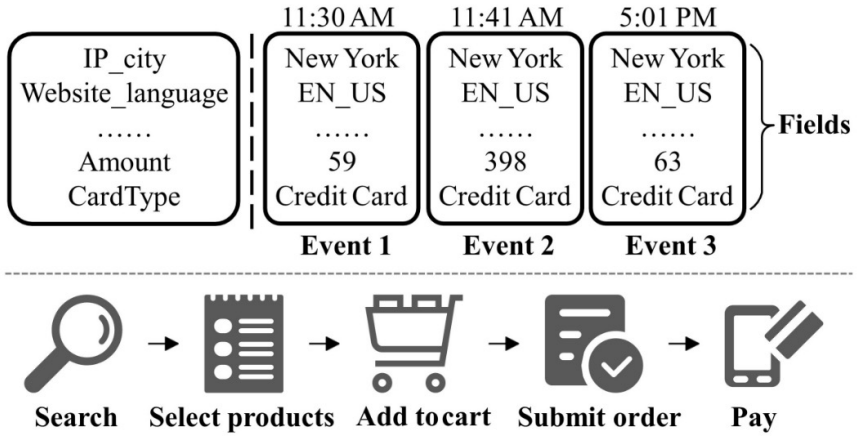


Fig. 1: **Up:** A user behavior sequence example consists of three events and each event has several fields. **Down**: An example of user behavior sequence for online shopping.

- GNN-based sequence representation learning
  - leverage similar sequences from other users and model their correlations

- Shortcomings when deploy GNN-based models to ***online services***:
  - **Training**: the number of user sequences produced by online applications is often immense, potentially escalating to the magnitude of millions → large-scale graphs incur substantial computational and memory burdens
  - **Inference**: online services typically require rapid response → puts stringent demands on the algorithms' inference efficiency.

Introduction

Basic idea

- we question the necessity of modeling all user sequences as nodes in GNNs
- → select a representative node subset
- **Solution**: compress the graph by reducing nodes and edges prior to GNN model training, benefiting computational efficiency

Other advantages

- Case-based Reasoning
  - Graph compression provides representative sequence prototypes, offering interpretable cases of model outputs.
- Sample Balancing
  - In biased distributions (e.g., fraud detection), compression can balance categories by setting similar compressed node counts.

Method

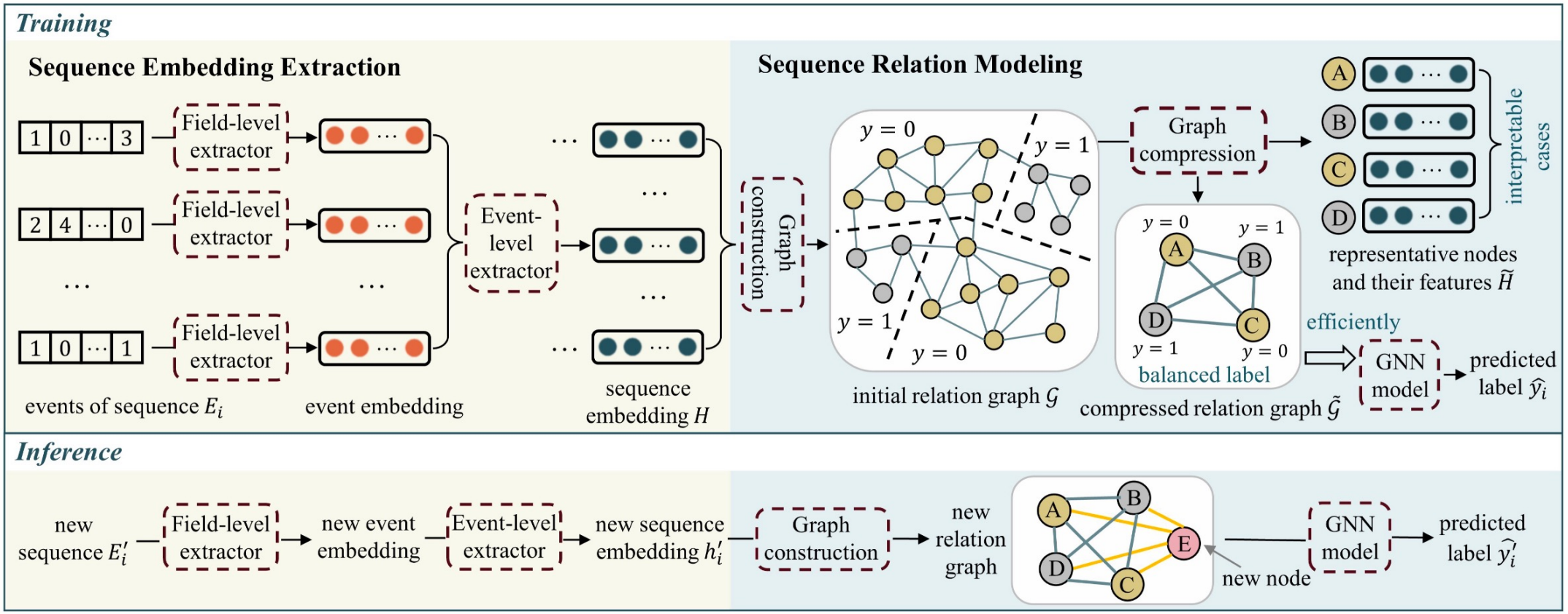


Fig. 2: Overview of *ECSeq*. Firstly, the *sequence embedding extraction* module meticulously transforms sequence information into a one-dimensional feature vector. Then, the *sequence relation modeling* module explores and leverages relationships among sequences to enhance the sequence representation, employing an appropriate graph compression technique to mitigate computational overhead and improve inference efficiency.

Method

- Graph Compression

TABLE I: Summary of typical graph compression methods. N: number of nodes, M: number of edges, D: dimension of node features, K: number of clusters/compressed nodes, c: some absolute constant. *Traceable*: whether the source of the compressed nodes is known; *Configurable*: whether the compression method can assign separate compressed node quantities for each category.

Category	**Methods**	**Input**	*Efficiency*		*Interpretability*	*Balancing*
			#Nodes↓	**#Edges↓**	**Traceable**	**Configurable**
Coreset Selection	k-means [27]	$\mathcal{X}$	✓	✓	✓	✓
	AGC [28]	$\mathcal{A}, \mathcal{X}$	✓	✓	✓	✗
	Grain [29]	$\mathcal{A}, \mathcal{X}$	✓	✓	✓	✓
	VNG [30]	$\mathcal{A}, \mathcal{X}$	✓	✓	✓	✗
Graph Coarsening	RSA [31]	$\mathcal{A}$	✓	✓	✓	✗
	REC [32]	$\mathcal{A}$	✓	✓	✓	✗
	GOREN [22]	$\mathcal{A}$	✓	✓	✓	✗
Graph Sparsification	ApproxCut [24]	$\mathcal{A}$	✗	✓	✓	✗

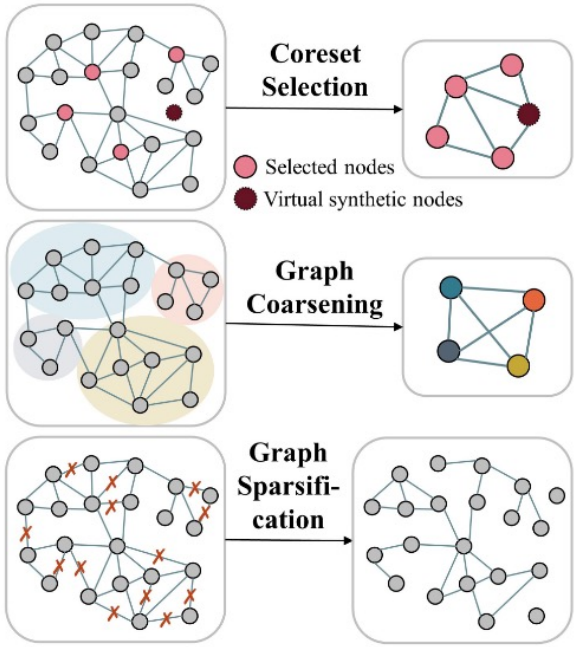


Fig. 3: Illustration of diverse graph compression methods.

Method

- Overall Workflow

Algorithm 1: *ECSeq* Training Procedure

Input: Sequence set E and its label matrix Y

Output: Optimized sequence embedding extractor ($\mathcal{M}_f$, $\mathcal{M}_e$), optimized relation model ($\mathcal{M}_g$, $\mathcal{F}_{gnn}$), and the compressed graph $\tilde{\mathcal{G}}$.

Initialize parameters of sequence embedding extractor $\mathcal{M}_f$, $\mathcal{M}_e$, and $\mathcal{F}_{seq}$;
while *stopping condition is not met* **do**
  $H = \mathcal{M}_e(\mathcal{M}_f(E))$, $\hat{Y} = \mathcal{F}_{seq}(H)$;
  Compute the loss $\mathcal{L}_{seq}$ by Eq. 4;
  Update the parameters of $\mathcal{M}_f$, $\mathcal{M}_e$, and $\mathcal{F}_{seq}$;
end
Treat sequences as nodes with $\mathcal{X} = H$;
Construct node connections and get relation graph $\mathcal{G} = (\mathcal{A}, \mathcal{X})$;
Compress $\mathcal{G}$ to get the compressed graph $\tilde{\mathcal{G}} = (\tilde{\mathcal{A}}, \tilde{\mathcal{X}})$ and label $\tilde{\mathcal{Y}}$;
Initialize parameters of relation model $\mathcal{M}_g$ and $\mathcal{F}_{gnn}$;
while *stopping condition is not met* **do**
  $\hat{\mathcal{Y}} = \mathcal{F}_{gnn}(\mathcal{M}_g(\tilde{\mathcal{A}}, \tilde{\mathcal{X}}))$;
  Compute the loss $\mathcal{L}_{com}$ by Eq. 10;
  Update the parameters of $\mathcal{M}_g$ and $\mathcal{F}_{gnn}$;
end
Establish connections between $\mathcal{X}$ and $\tilde{\mathcal{X}}$, denoted as $\mathcal{A}'$;
while *stopping condition is not met* **do**
  $\hat{Y}' = \mathcal{F}_{gnn}(\mathcal{M}_g(\mathcal{A}', \mathcal{X} \cup \tilde{\mathcal{X}}))$;
  Compute the loss $\mathcal{L}_{cor}$ by Eq. 12;
  Update the parameters of $\mathcal{M}_g$ and $\mathcal{F}_{gnn}$;
end

Algorithm 2: *ECSeq* Inference Procedure

Input: Optimized sequence embedding extractor ($\mathcal{M}_f$, $\mathcal{M}_e$), optimized relation model ($\mathcal{M}_g$, $\mathcal{F}_{gnn}$), the compressed graph $\tilde{\mathcal{G}}$, and a set of new sequences E''

Output: The predicted label $\hat{Y}''$ of the new sequences.

Get new sequence embedding $H'' = \mathcal{M}_e(\mathcal{M}_f(E''))$;
Treat new sequences as nodes with $\mathcal{X}'' = H''$;
Establish connections between $\mathcal{X}''$ and $\tilde{\mathcal{X}}$, denoted as $\mathcal{A}''$;
Derive inference results $\hat{Y}'' = \mathcal{F}_{gnn}(\mathcal{M}_g(\mathcal{A}'', \mathcal{X}'' \cup \tilde{\mathcal{X}}))$;

Our *step-wise* approach has advantages over end-to-end training:

- Graph compression is only needed once, mitigating instability and inefficiency;
- Both modules' training processes incorporate label information supervision, ensuring the models' stability and optimality;
- Decoupled optimization enables flexibility in using diverse models without alignment concerns.

Experiments

Datasets:

TABLE II: Statistics of datasets.

Fraud Detection				
Datasets	#Fields	#Events	#Sequences	#Positive Samples
FD1	236	2,130,962	245,045	24,489 (9.99%)
FD2	178	275,322	15,366	777 (5.06%)
User Mobility				
Datasets	#Sensors	#Timesteps	Time Interval	Value Range
Bike	717	1,488	60 minutes	0.0 $\sim$ 108.0
Speed	325	1,488	60 minutes	3.1 $\sim$ 83.2

Baselines:

- *Methods with only features of the target event:* Regression and GBDT take features extracted by the field-level extractor of the target event as inputs to train a machine-learning classifier.
- *Methods with deep neural networks to extract historical information:* LSTM can capture long-term dependencies for sequential data, then we give the prediction by MLP layers, while R-Transformer combines RNNs and the multi-head attention mechanism.
- *Methods with GNN to capture sequence relationship:* GRASP enhances representation learning by leveraging knowledge extracted from similar users within the same batch, which is originally proposed for healthcare sequence classification problems.

Experiments

TABLE IV: Experimental results on fraud detection and user mobility tasks. The best results are highlighted in bold. While *R-Transformer* [48] and *GRASP* [7] are primarily intended for classification tasks, they do not show comparable performance on user mobility tasks.

Methods	FD1		FD2		Bike		Speed	
	AUPRC (↑)	R@P$_{0.9}$ (↑)	AUPRC (↑)	R@P$_{0.9}$ (↑)	RMSE (↓)	sMAPE (↓)	RMSE (↓)	sMAPE (↓)
Non-Graph Methods								
Regression	0.7685±0.0000	0.4890±0.0000	0.5271±0.0000	0.3052±0.0000	2.8169±0.0000	0.2148±0.0000	6.3994±0.0000	0.0672±0.0000
GBDT	0.7742±0.0000	0.5244±0.0000	0.6147±0.0006	0.3766±0.0000	2.7596±0.0077	0.2063±0.0008	6.2964±0.0575	0.0632±0.0005
LSTM	0.8332±0.0047	0.5840±0.0132	0.7124±0.0076	0.6987±0.0078	1.5463±0.0504	0.1782±0.0040	4.8383±0.1600	0.0561±0.0011
R-Transformer	0.8338±0.0040	0.5847±0.0289	0.7064±0.0149	0.5403±0.1951	–	–	–	–
Graph Methods								
GRASP	0.8362±0.0037	0.6049±0.0230	0.7138±0.0353	0.6776±0.0420	–	–	–	–
ECSeq	**0.8383±0.0018**	**0.6153±0.0079**	**0.7249±0.0112**	**0.7039±0.0032**	**1.4832±0.0209**	**0.1766±0.0038**	**4.4362±0.1015**	**0.0542±0.0012**

Experiments

TABLE V: Fraud detection performance and computation time on *FD1*, whose training set contains 145,236 sequences, *i.e.*, 145,236 original nodes when modeling the relation. We train the GNN for 50 epochs.

Methods	#Nodes	AUPRC (↑)	R@$P_{0.9}$ (↑)	Compression Time (s) (↓)	GNN Training Time (s) (↓)	Inference Time (10^{-4} s/sample) (↓)	GPU Memory Usage (GB) (↓)
LSTM	–	0.8332±0.0047	0.5840±0.0132	–	–	**0.286**	–
ECSeq	100	0.8377±0.0023	0.6111±0.0077	**5.318**	**9.950**	0.614	**1.306**
	500	**0.8383±0.0018**	**0.6153±0.0079**	15.444	9.955	0.618	1.664
	1,000	0.8372±0.0021	0.6115±0.0083	36.686	10.480	0.622	2.283
	5,000	0.8343±0.0013	0.6056±0.0055	137.570	28.930	0.630	7.037
batch GNN	145,236 (1,000/batch)	0.8335±0.0017	0.5861±0.0157	–	10.675	2.243	6.414
full graph GNN	145,236			Out-of-Memory			

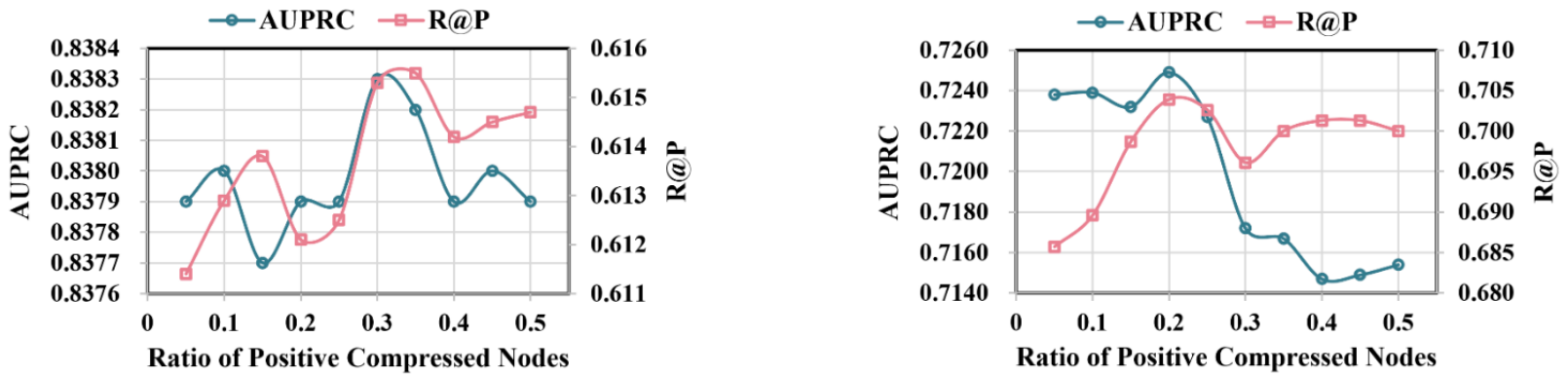
TABLE VI: Performance of *ECSeq* variants on *FD2* and *Bike*. *R-Transformer* cannot converge on *Bike*, so we do not report the results.

	Sequence Embedding Extractor	**Graph Compression Algorithm**	**GNN Model**	**FD2**		**Bike**	
				AUPRC (↑)	R@$P_{0.9}$ (↑)	RMSE (↓)	sMAPE (↓)
Sequence Model w.o. Graphs	LSTM	–	–	0.7124±0.0076	0.6987±0.0078	1.5463±0.0504	0.1782±0.0040
	R-Transformer	–	–	0.7064±0.0149	0.5403±0.1951	–	–
ECSeq	LSTM	k-means	GraphSAGE (Mean)	**0.7249±0.0112**	**0.7039±0.0032**	1.4832±0.0209	0.1766±0.0038
	R-Transformer	k-means	GraphSAGE (Mean)	0.7105±0.0184	0.6991±0.0031	–	–
	LSTM	AGC	GraphSAGE (Mean)	0.7232±0.0102	0.6935±0.0095	**1.4792±0.0218**	0.1764±0.0044
	LSTM	Grain	GraphSAGE (Mean)	0.7232±0.0102	0.6870±0.0095	1.4949±0.0219	0.1812±0.0050
	LSTM	RSA	GraphSAGE (Mean)	0.7201±0.0091	0.6922±0.0120	1.4812±0.0190	**0.1761±0.0040**
	LSTM	k-means	GraphSAGE (Max)	0.7162±0.0083	0.7026±0.0049	1.4846±0.0166	0.1768±0.0037
	LSTM	k-means	GCN	0.7212±0.0102	0.6987±0.0052	1.9226±0.0376	0.2139±0.0059
	LSTM	k-means	GAT	0.7200±0.0112	0.7026±0.0026	2.0056±0.0107	0.2383±0.0053

Experiments

TABLE VII: The new sequences of fraudulent behavior and their corresponding representative sequences with the closest relation (*i.e.*, the highest cosine similarity of sequence embedding). Events are arranged from left to right in chronological order, and the underlined elements are the target events. *RouterMac*: the mac address of users' devices; *Item*: the category of goods; *CardType*: the type of payment card (*e.g.* credit card); *Country*: the transaction location.

	Fields	New Sequences (Fraud)					Representative Sequences with the Closest Relation				
(a)	**RouterMac**	MAC1	MAC1	MAC1	MAC1	MAC2	MAC3	MAC3	MAC4	MAC3	MAC2
	Amount	242	484	242	242	5634	429	529	6	129	5478
(b)	**Item**	I1	I1	I1	I1	I3	I2	I1	I1	I1	I3
	Time	7:43 PM	6:21 PM	8:10 PM	10:41 PM	3:05 AM	8:57 PM	5:21 PM	5:34 PM	9:01 PM	2:06 AM
(c)	**Country**	C1	C2	C1	C1	C3	C1	C1	C4	C1	C3
	CardType	TYPE1	TYPE1	TYPE1	TYPE2	TYPE3	TYPE1	TYPE2	TYPE1	TYPE1	TYPE3



(a) *FD1* (b) *FD2*

Fig. 4: Fraud detection performance of *ECSeq* under varying settings of positive compressed node ratio.

GNN for edge computing

中国计算机学会推荐国际学术刊物
（● 计算机网络）

A类

序号	刊物名称	刊物全称	出版社	地址
1	JSAC	IEEE Journal on Selected Areas in Communications	IEEE	http://dblp.uni-trier.de/db/journals/jsac/

GNN at the Edge: Cost-Efficient Graph Neural Network Processing over Distributed Edge Servers

Liekang Zeng, Chongyu Yang, Peng Huang, Zhi Zhou, Shuai Yu, and Xu Chen

***Abstract*—Edge intelligence has arisen as a promising computing paradigm for supporting miscellaneous smart applications that rely on machine learning techniques. While the community has extensively investigated multi-tier edge deployment for traditional deep learning models (*e.g.* CNNs, RNNs), the emerging Graph Neural Networks (GNNs) are still under exploration, presenting a stark disparity to its broad edge adoptions such as traffic flow forecasting and location-based social recommendation. To bridge this gap, this paper formally studies the cost optimization for distributed GNN processing over a multi-tier heterogeneous edge network. We build a comprehensive modeling framework that can capture a variety of different cost factors, based on which we formulate a cost-efficient graph layout optimization problem that is proved to be NP-hard. Instead of trivially applying traditional data placement wisdom, we theoretically reveal the structural property of quadratic submodularity implicated in GNN's unique computing pattern, which motivates our design of an efficient iterative solution exploiting graph cuts. Rigorous analysis shows that it provides parameterized constant approximation ratio, guaranteed convergence, and exact feasibility. To tackle potential graph topological evolution in GNN processing, we further devise an incremental update strategy and an adaptive scheduling algorithm for lightweight dynamic layout optimization. Evaluations with real-world datasets and various GNN benchmarks demonstrate that our approach achieves superior performance over *de facto* baselines with more than 95.8% cost reduction in a fast convergence speed.**

***Index Terms*—Edge intelligence, Graph Neural Networks, cost optimization, distributed edge computing.**

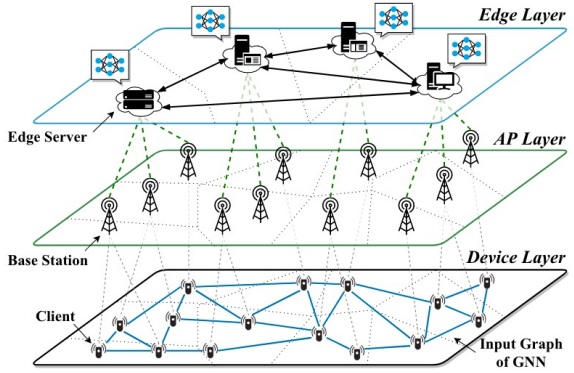


Fig. 1. A multi-tier edge network architecture comprising device layer, access point (AP) layer, and edge layer. The clients with connections form a data graph (*e.g.*, social web), while each node in the edge network covers a range of users. The edge servers perform distributed GNN processing by parallelizing model execution and exchanging data mutually, in order to infer targeted graph properties (*e.g.*, predicting potential social relationships).

techniques with convolution to collectively aggregate information from nodes and their dependencies, enabling capturing hierarchical patterns from subgraphs of variable sizes. Benefited from such advanced ability in modeling graph structures, GNNs have been recently employed in miscellaneous graph-

Introduction

- While traditional deep learning models (e.g. CNNs, RNNs) accept inputs from each individual client independently, processing GNNs at the network edge can *span over distributed edge servers geographically*, due to the dispersed nature of graph data.
- When a GNN inference query is raised, each edge server
  - first aggregates a subset of the graph data via APs in certain areas
  - next launches a **distributed runtime** to compute the embeddings through the given GNN model
    (During the runtime, the edge servers are orchestrated in a collaborative manner, exchanging necessary graph data with each other)

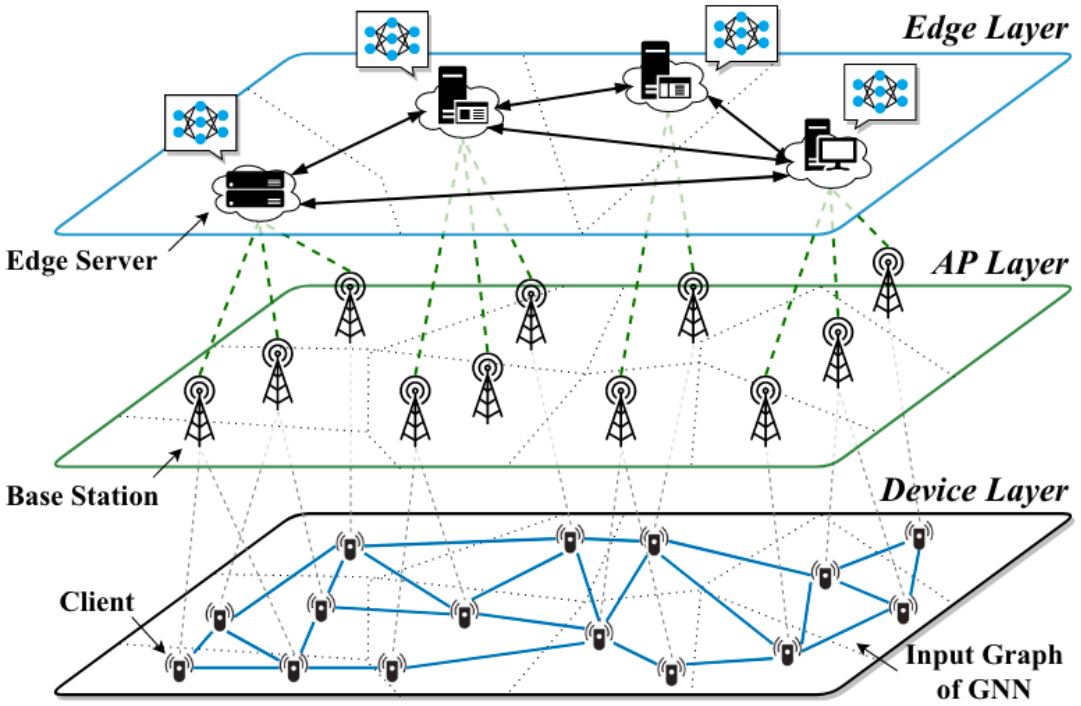


Fig. 1. A multi-tier edge network architecture comprising device layer, access point (AP) layer, and edge layer. The clients with connections form a data graph (*e.g.*, social web), while each node in the edge network covers a range of users. The edge servers perform distributed GNN processing by parallelizing model execution and exchanging data mutually, in order to infer targeted graph properties (*e.g.*, predicting potential social relationships).

Introduction

- Existing works on edge-enabled distributed intelligence:
  - offload computation with external infrastructures
  - partition model execution across edge servers
- **Shortcoming:** assume single-point input and an independent serving style between one client and one edge server, which fits CNNs and RNNs but is inapplicable for GNNs.

Contribution

- Formulate a **system cost optimization problem**
  - for *Distributed GNN Processing over heterogeneous Edge servers* (DGPE)
  - by building a novel modeling framework that generalizes to a wide variety of cost factors
- Theoretically reveal that the optimization problem's objective function is pseudo-boolean quadratic and submodular, based on which we propose an efficient algorithm leveraging **graph-cuts techniques**
- Develop an incremental graph layout improvement strategy to address the potential **dynamic evolution** of GNN's input data graph, and further design an adaptive scheduling algorithm to well balance the tradeoff between graph layout update overhead and system performance

Related work

- Distributed GNN systems.
- **Federated learning for GNN.**
  - DGPE significantly diverges from the FL category in that:
  - 1) DGPE considers distributed GNN inference processing services with a variety of **cost optimization** objectives, allowing attainable data sharing across edge servers, while FL focuses only on the GNN model training and particularly stresses **user privacy** by physical data isolation.
  - 2) DGPE targets a technically distinct execution mechanism from FL, where DGPE only touches the distributed clients and edge servers, and processes GNN inference workload in parallel without the orchestration of **centralized cloud servers** (as required by FL).
- Collaborative edge intelligence.
- Graph data placement.

System model

- **System overview**
  - the ***edge network*** that hosts distributed model execution
  - the data graph formed by clients' associated data, which feeds the GNN model as the ***input graph***
  - We bridge them by further defining ***graph layout***

- **Cost Factors**
  - Data collection
  - GNN computation
  - Cross-edge traffic
  - Edge server maintenance

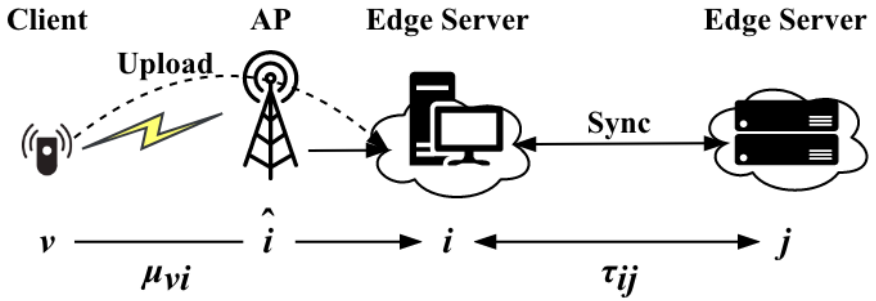


Fig. 3. Illustration of the data collection cost and cross-edge traffic cost for a single client's data.

- **Cost-Efficient Graph Layout Optimization Problem**

$$\mathcal{P}: \quad \min_{\pi} \mathcal{C}(\pi|\mathcal{T},\mathcal{G}), \tag{10}$$

$$\text{s.t.} \sum_{i\in\mathcal{D}} x_{vi} = 1, \forall v \in \mathcal{V}, \tag{10a}$$

$$x_{vi} \in \{0,1\}, \forall v \in \mathcal{V}, \forall i \in \mathcal{D}, \tag{10b}$$

$$\pi = \{x_{vi} | v \in \mathcal{V}, i \in \mathcal{D}\}. \tag{10c}$$

Approximation algorithm design for static input graphs

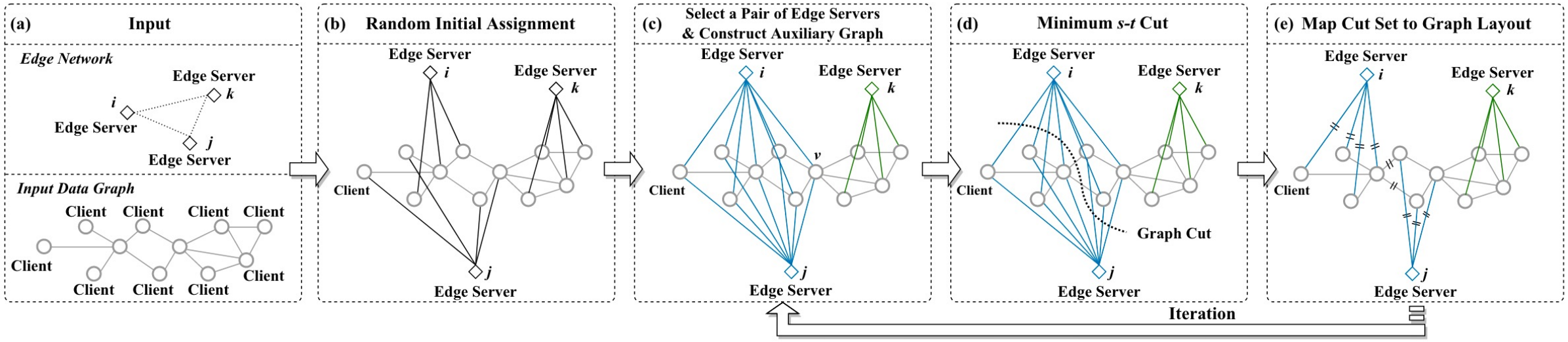


Fig. 4. Instance of graph-cut based graph layout optimization. Given an edge network and an input data graph (a), GLAD-S first initializes a graph layout via random assignments (b). Next, it selects an arbitrary pair of edge servers and constructs an augment graph by connecting both edges to every vertex that has been associated with one of them (c). Minimum *s-t* cut is then performed on the augment graph and obtained a set of graph cuts (d). Based on this cut set, it builds a graph layout for the edge servers and the client inside the augment graph, where marked links indicate assignments (e). GLAD-S will pass the obtained result to step (b) as the next iteration for finding a better graph layout until convergence.

Approximation algorithm design for static input graphs

Algorithm 1 GLAD-S: GLAD for Static input data graph

Input:

$\mathcal{T}$: The edge network $(\mathcal{D}, \mathcal{W})$
$\mathcal{G}$: The data graph $(\mathcal{V}, \mathcal{E})$
$C(\pi)$: The system cost function for π
R: The measurement of convergence

Output:

π: Optimized graph layout $\{x_{vi} | v \in \mathcal{V}, i \in \mathcal{D}\}$

Initialize a randomized graph layout π
$r \leftarrow 0$
while $r \leq R$ **do**
  Select a pair of connected edge servers $\langle i, j \rangle$ with the minimum visited times
  Construct an auxiliary graph $\mathcal{A}(i, j)$ *w.r.t.* $\langle i, j \rangle$
  Solve the minimum s-t cut of $\mathcal{A}(i, j)$
  Build a graph layout π' according to Eq. (15) from the obtained minimum cut set
  if $C(\pi') < C(\pi)$ **then**
    $\pi \leftarrow \pi'$
    $r \leftarrow 0$
  else
    $r \leftarrow r + 1$
  end if
end while
return π

Adaptive algorithm design for evolved input graphs

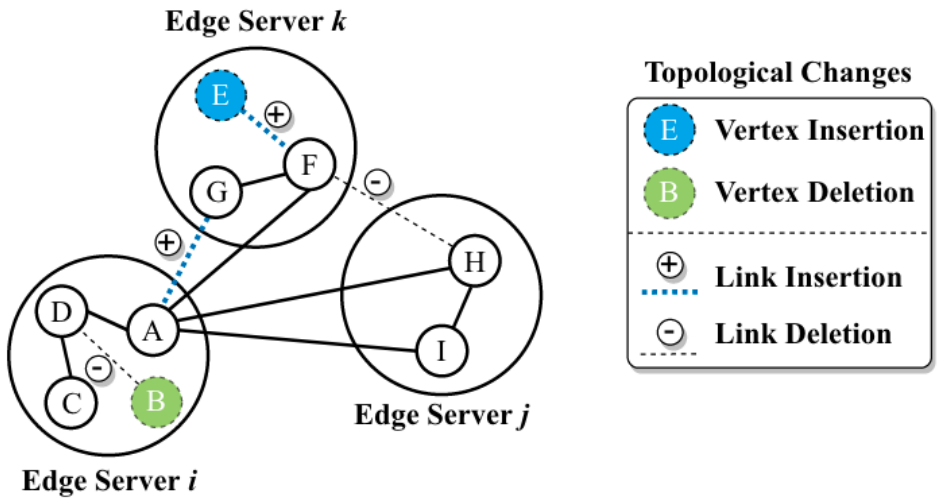


Fig. 5. Possible topological graphs on a data graph, where the clients (vertices) have been placed to three edge servers. The changes include vertex insertion, vertex deletion, link insertion, and link deletion.

Algorithm 2 GLAD-E: GLAD for Evolved input data graph

Input:

$\mathcal{T}$: The edge network

$\mathcal{G}(t-1)$, $\mathcal{G}(t)$: The data graph at time slot $t-1$ and t

$\pi(t-1)$: The graph layout at time slot $t-1$

$C(\pi)$: The total cost function for a given π

R: The measurement of convergence

Output:

$\pi(t)$: Optimized graph layout at time slot t

1: Filter the vertices that are newly added or have new neighbors at other edge servers from $\mathcal{G}(t-1)$ and $\mathcal{G}(t)$

2: Construct a graph $\mathcal{G}^+$ with the filtered vertices and their associated links

3: $\pi^+ \leftarrow$ GLAD-S$(\mathcal{T}, \mathcal{G}^+, C, R)$ ▷ Call Algorithm 1

4: Extract the graph layout π^- from existing layout $\pi(t-1)$ with respect to the unfiltered vertices

5: $\pi(t) \leftarrow \pi^+ \cup \pi^-$

6: **return** $\pi(t)$

Evaluation

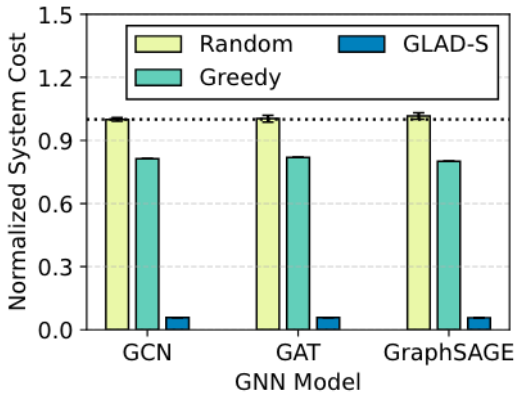


Fig. 8. The achieved system cost of different GNN models on SIoT.

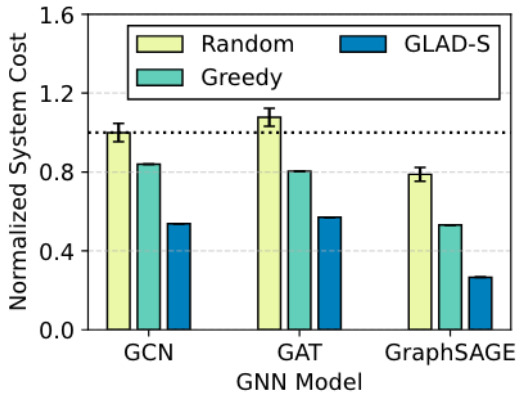


Fig. 9. The achieved system cost of different GNN models on Yelp.

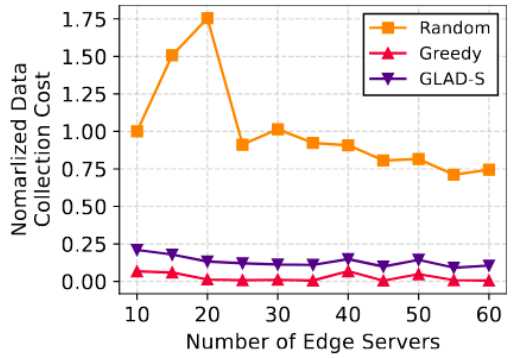


Fig. 10. Data collection cost on Yelp with varying number of edges.

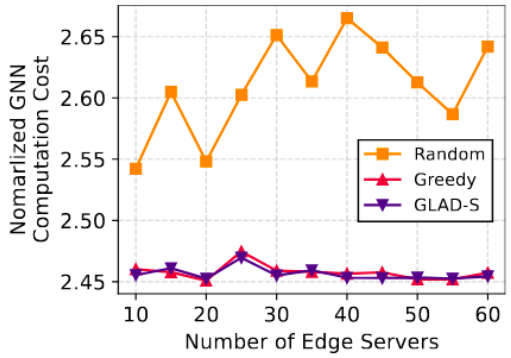
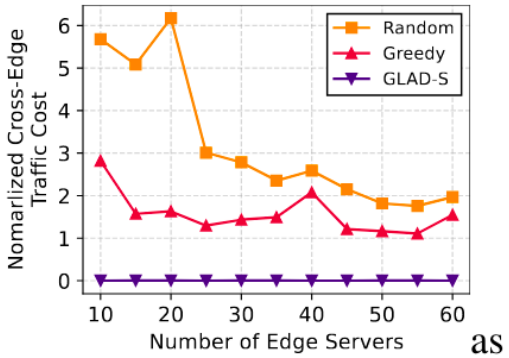


Fig. 11. GNN computation cost on Yelp with varying number of edges.



as

Fig. 12. Cross-edge traffic cost on Yelp with varying number of edges.

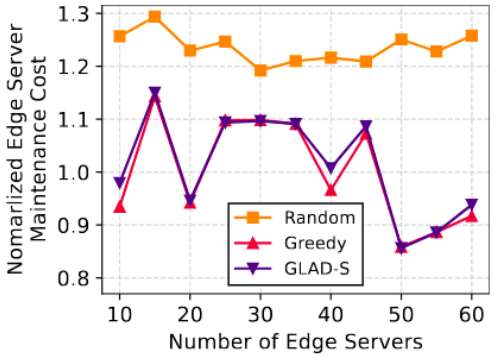


Fig. 13. Maintenance cost on Yelp with varying number of edges.

Thanks