DRUM: A Real Time Detector for Regime Shifts in Data Streams via an Unsupervised, Multivariate Framework

Adnan Bashir(✉) and Trilce Estrada

Computer Science Department, University of New Mexico, Albuquerque, NM, USA
{abashir,estrada}@cs.unm.edu

Abstract. In this work we present DRUM, an unsupervised approach that is based on statistical properties of multivariate data streams to identify regime shifts in real time. DRUM processes streams in small chunks, learns their statistical properties, and makes generalizations as time goes by. We show how this straightforward approach requires minimal computation and reaches state of the art accuracy, making it ideal for embedded and cyber physical systems.

Keywords: Multivariate data streams · Change point detection · Statistical analysis · Unsupervised · Real time · Online detection

1 Introduction

A regime shift refers to a sudden or significant change over time in the behavior of a system. It can occur in various natural and social systems, ranging from ecosystems and climate patterns to economic and social systems. Regime shifts in cyber physical systems can indicate potential problems or pattern changes that need to be dealt with in real time [3]. Regime Shift Detection (RSD), or Change Point Detection (CPD) refers to the identification of such changes in the underlying distribution of data streams [5]. Detection of regime shifts is of vital importance in many real-world problems such as weather monitoring, early detection of cyber security threats, medical monitoring, speech analysis, market analysis, human activity monitoring and many more [23]. For dynamic cyber physical systems, real time detection of regime shifts is essential for prevention and mitigation of system failures. Understanding when a regime shift is underway enables informed decision making, improved risk management, and accurate policy development, leading to increased resilience of the overall system.

In this paper we present DRUM (A real time **D**etector for **R**egime shifts in data streams via an **U**nsupervised and **M**ultivariate framework). DRUM has the following properties that make it suitable for solving regime shift detection in data streams: **Unsupervised** - Our method doesn't require labels or human intervention. **Real time responsiveness** - Due to very low computation, it

R. Wrembel et al. (Eds.): DaWaK 2023, LNCS 14148, pp. 294–302, 2023.
https://doi.org/10.1007/978-3-031-39831-5_27

offers minimum lag in detecting change points. **Online** - Our method processes data as a stream and does not require retraining or batch processing to adapt to changes over time. **General** - Our method has been tested on various univariate and multivariate datasets, and it was able to perform as well or with higher accuracy as other frameworks.

2 Related Work

Offline Methods - Offline CPD refers to those methods that require a complete data stream for detection [5]. Numerous offline CPD algorithms have been proposed in the past few decades. CUSUM method [21] was one of the earliest change point detectors that identified points in time series with their cumulative sum exceeding a threshold value. DRE-CUSUM [2] is built upon CUSUM which splits the data stream from an arbitrary point and compares the distribution on both sides to detect any change. Binary segmentation was introduced as an approximate change point detection method that greedily splits the data stream into disjointed segments based on a cost function. To reduce the higher time complexity of binary segmentation, pruning was added [14]. Wild Binary Segmentation extended the original binary segmentation method by applying CUSUM on subsets of data stream [10].

Online Methods - Online CPD refers to those methods that process the data as it arrives. They can be viewed as ϵ-real time algorithms [5]. Where ϵ is the delay in timestamps required by the algorithm to accurately detect a change point. To incorporate multivariate data streams and online requirement of CPD, Binary Online Changepoint Detection (BOCD) was proposed [1]. BOCD introduced a term called run-length, which denotes data stream since the most recent change point. BOCD approximates the probability distribution over run-length and compares it with previously approximated distribution for change point detection. Lately, several methods were proposed that were based on BOCD namely BOCPDMS [16] and RBOCPDMS [15]. We have compared all variants of BOCP due to its online detection ability.

Supervised Methods - Supervised learning algorithms have proven to be very effective when it comes to change point detection but they require labeled data and a training phase. Traditional machine learning methods like SVM [8], random forests [6], decision trees [22], hidden Markov models [20], nearest neighbors [19], and other binary classifiers [4] have been used to deal with this problem as a supervised classification problem. Among them, RFPOP [9] detects change points in an offline setting by engineering the cost function via dynamic programming algorithm. Another such approach is PELT [14] which minimizes the detection delay in univariate time series. Facebook has proposed PROPHET [25] which works as a regression model which requires domain expertise for tuning. This is not feasible for unsupervised and online applications.

Deep Learning and Neural Network-based Models - Neural Networks have been readily used for time series analysis. But they also suffer some limitations when it comes to online CPD. As stated by Zhang et al. [27], the network

structure, training method and sample data may affect the performance and accuracy of Neural Network based methods. Also training Neural Networks is a time-consuming process and usually requires a big chunk of data in training. For example, GAN (Generative Adversarial Networks [11]) based models require entire variable set concurrently to tune the hidden layer parameters (such as MAD-GAN [18]). Re-tuning these models is again a time-consuming process that compromises the real time response of these detectors. Hence a majority of Neural Network based CPD algorithms outperform when trained in an offline setting.

3 DRUM

In this section we describe our approach (DRUM) which consists of two modules: a scoring module, which computes statistics of the individual data streams; and a detection module, which quantifies the rate of change and determines if a regime shift has occurred.

Scoring Module: this module takes a multivariate data stream and calculates our Lobo Change Score (LCS), which is the weighted sum of change in mean (Δm), change in standard deviation (Δs), and change in fluctuations across the running mean (Δfrm) between two data windows of the data stream. Formally, the data stream is an infinite sequence of values generated by an underlying system and is denoted as $S = \{x_1, x_2, ..., x_{t-1}, x_t, x_{t+1}, ...\}$, where for a multivariate time-series with n variables, x_t is the n-dimensional data vector at time t [26]. LCS is computed for small chunks of data that we call windows. That is, given variable i of the stream, for a window of size $d+1$ starting in position j, our algorithm processes the sequence $w_{i,[j,j+d]} = \{x_{i,j}, x_{i,j+1}, x_{i,j+2}, ..., x_{i,j+d}\}$. LCS is composed of three components:

$$\Delta m_{i,j,k} = \left| m(w_{i,[j,j+d]}) - m(w_{i,[k,k+d]}) \right| \tag{1}$$

$$\Delta s_{i,j,k} = \left| s(w_{i,[j,j+d]}) - s(w_{i,[k,k+d]}) \right| \tag{2}$$

$$\Delta frm_{i,j,k} = \left| frm(w_{i,[j,j+d]}) - frm(w_{i,[k,k+d]}) \right| \tag{3}$$

where for two windows: $w_{i,[j,j+d]}$ starting at position j, and $w_{i,[k,k+d]}$ starting at position k; both of them corresponding to variable i, $\Delta m_{i,j,k}$ is the absolute difference between their mean; $\Delta s_{i,j,k}$ is the absolute difference in their standard deviation; and Δfrm is the absolute difference of fluctuation across the running mean. frm is computed by counting the number of intersections between the running mean and the signal itself within the current window. Then, LCS for windows $[j, j+d]$ and $[k, k+d]$ is the weighted sum of the three components described above, as shown by Eq. 4:

$$LCS_{j,k} = \alpha \sum_{i=1}^{n} \Delta m_{i,j,k} + \beta \sum_{i=1}^{n} \Delta s_{i,j,k} + \gamma \sum_{i=1}^{n} \Delta frm_{i,j,k} \tag{4}$$

where α, β and γ are weight coefficients, used to differentiate the contribution of the three components. They can have values between 0 and 1, these values can be updated adaptively depending on the data characteristics, or can be kept constant to simplify hyperparameter selection. As shown in Algorithm 1, we use two types of windows for calculating LCS. LCS_D is calculated using disjointed windows, that is, windows of size $d+1$ that do not overlap, they track coarse changes over a period of time. While LCS_S uses consecutive sliding windows to detect with finer precision when a change occurs. Figure 1 visually explains the two types of windows.

Algorithm 1: DRUM

Input: Multivariate data stream S; window length $d+1$; α, β, γ
Output: LCS_D, LCS_S
$t \leftarrow 0$; $l \leftarrow t+1$; $j \leftarrow d+1$; $k \leftarrow 1$;
$LCS_D = \{\}$,$LCS_S = \{\}$
while $S \neq \{\}$ **do**
  while $t++ \leq d$ **do**
    $\forall_i w_{i,[k,k+d]} \leftarrow x_{i,t}$ /* gather data for window $[k, k+d]$ */
  end
  $t++$ /* use t as the timestamp counter */
  /* compute LCS_D for disjoint windows j and k */
  if $j \leq t \leq j+d$ **then**
    $\forall_i w_{i,[j,j+d]} \leftarrow x_{i,t}$ /* gather data for window $[j, j+d]$ */
  end
  if $t = j+d$ **then**
    $LCS_D \leftarrow \forall_i \alpha \Delta m_{i,j,k} + \beta \Delta s_{i,j,k} + \gamma \Delta frm_{i,j,k}$
    k $\leftarrow j$
    j $\leftarrow j+d+1$
  end
  /* compute LCS_S for consecutive sliding windows ending at t */
  update $\forall_i w_{i,[t-d,t-1]}$
  update $\forall_i w_{i,[t-d+1,t]}$
  $LCS_s \leftarrow \forall_i \alpha \Delta m_{i,t-d,t-d+1} + \beta \Delta s_{i,t-d,t-d+1} + \gamma \Delta frm_{i,t-d,t-d+1}$
  LCS_S.append(LCS_s)
  if *detectChange*(LCS_D) **then**
    return maxChange(LCS_S)
  end
end

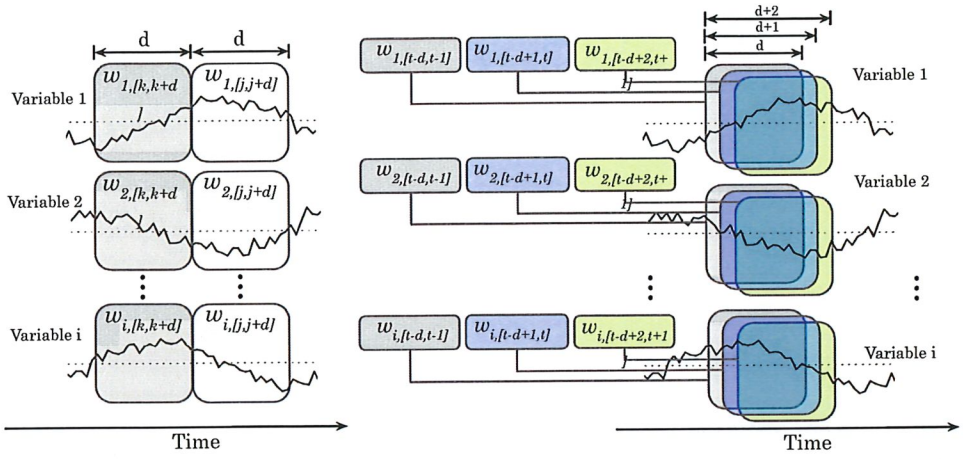


Fig. 1. Visualization of a) Disjointed Windows and b) Sliding Windows

Detection Module: Once LCS is calculated for disjointed (LCS_D) and sliding windows (LCS_S), it is evaluated by the detection module. The first step in detecting regime shift is to identify the window in which the change has occurred. If LCS_D has increased or decreased by 5% or more, the window is marked as a change point window. Once the change window is identified, we use LCS_S to further identify the specific timestamp with the maximum change score. This timestamp is returned as a change point.

4 Evaluation

To evaluate the accuracy and performance of our method, we compared its behavior on datasets labeled for CPD (see Table 1) with respect to other state of the art methods (see Table 2). Additional details regarding these methods were presented in Sect. 2. We used three metrics for comparison: NAB score, F1 score, and execution time. Below we present the results of such comparisons.

Table 1. Datasets and Benchmarks used for Evaluation

DATASET	Description	# of Variables	Length	# of Series
TCPD [7]	Real-world dataset for bench-marking CPD algorithms	1–5	15–991	42
TSSB [24]	Subset of UCR TS datasets	1	572–15970	66
SKAB [13]	Sensors on a testbed	8	666–1327	34
TEP [12]	Industrial chemical process	52	500–960	22
MDS	Mesa Del Sol microgrid	18	518960	1

Table 2. Properties of CPD frameworks used for evaluation

METHOD	Year	Online	No Training	Unsupervised	Multivariate
DRUM (ours)	2023	✔	✔	✔	✔
RFPOP [9]	2019	✖	✔	✖	✖
RBOCPDMS [16]	2018	✔	✔	✔	✔
PROPHET [25]	2018	✖	✖	✖	✖
BOCPDMS [15]	2018	✔	✔	✔	✔
WBS [10]	2012	✖	✔	✔	✖
PELT [14]	2012	✖	✔	✔	✖
BOCPD [1]	2007	✔	✔	✔	✔

NAB Score. NAB [17] was proposed in 2015 by Numenta[1]. It was the first anomaly and change point detection benchmark designed to evaluate unsupervised and real time algorithms. It rewards early detection and uses a scaled sigmoidal function to score false positives and false negatives. Figure 2 shows the NAB standard profile score. Our approach outperforms other methods in four out of five datasets (TCPBD, TEP, MDS, SKAB), and performs competitively in TSSB. It is important to note that NAB score allows the framework to perform a change point detection within a window (by default it is 5% of the length of time series).

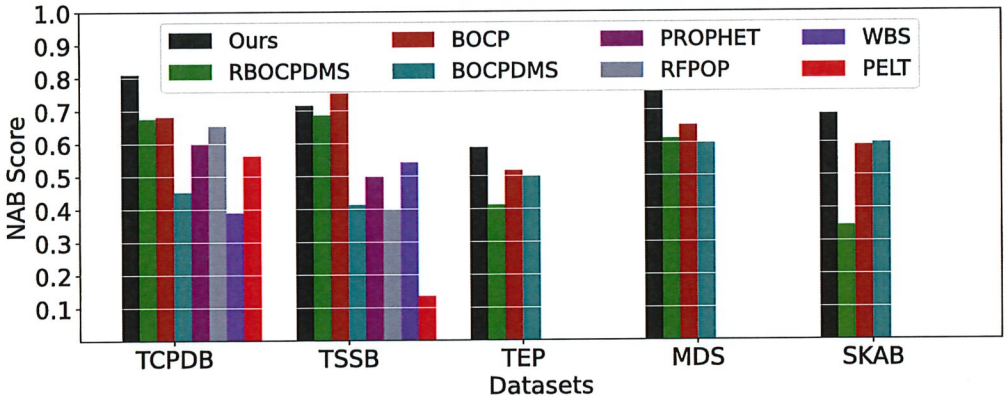


Fig. 2. NAB Standard Profile Score, Empty bars show that the framework is not multivariate capable

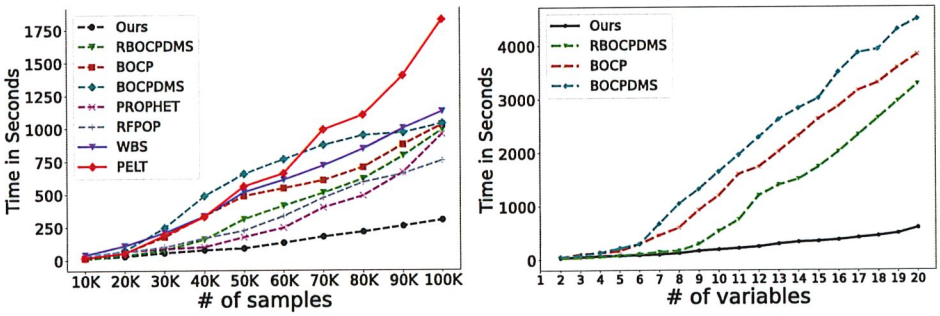
F1 Score. When a framework has a binary output (in this case change or no change), F1-Score has been readily used by the research community to evaluate such frameworks. F1 scores for the different methods are presented in Table 3 with the highest score per dataset in bold. Our approach outperforms other methods in four out of five datasets.

[1] https://numenta.com/.

Table 3. F1 Score. Empty fields mean that the framework is not designed for multivariate time series data

METHOD	TCPDB	TSSB	SKAB	TEP	MDS
DRUM (ours)	0.804	**0.715**	**0.634**	**0.664**	**0.714**
BOCPD [1]	**0.818**	0.345	0.568	0.608	0.640
BOCPDMS [15]	0.620	0.589	0.618	0.652	0.687
RBOCPDMS [16]	0.447	0.548	0.623	0.325	0.686
PELT [14]	0.787	0.563	-	-	-
PROPHET [25]	0.534	0.446	-	-	-
WBS [10]	0.533	0.214	-	-	-
RFPOP [9]	0.531	0.197	-	-	-

Execution Time. We also compared the runtime of CPD frameworks both on univariate and multivariate time series. All Experiments were run on an Intel Core i7-8750H @ 2.20 GHz with 6 cores and 32 GB of RAM. Implementation was done using Python 3.7 on Windows 11. Fig. 3a shows the execution time for univariate time series from 10,000 to 100,000 instances, sampled at intervals of 10,000. Our approach has the lowest execution time with approximately linear behavior, while others show an exponential response. Figure 3b shows the execution time for multivariate time series with a fixed length of 10,000 but with the number of features ranging from 2 to 20. Computation time for BOCPD and its variants depends on change point location: if a change point is not detected more samples are stored to approximate the distribution of the data stream. This results in longer execution over time. While our approach always used a constant amount of data and exhibits linear scalability.



(a) Univariate time series of increasing length

(b) Multivariate time series of length 10K with an increasing number of features

Fig. 3. Comparison of execution time

5 Conclusion

In this work, we present an unsupervised change point detector that works on streams and scales linearly with the number of variables being tracked. It requires minimal computations and can be applied toward detecting different events happening in a system in real time. Such events include system failures, concept drift, and state transitions. Since it is lightweight, it can be run on IoTs with low compute power.

Acknowledgments. This work was supported by National Science Foundation (NSF) EPSCoR grant number OIA-1757207.

References

1. Adams, R.P., MacKay, D.J.C.: Bayesian Online Changepoint Detection (2007). http://arxiv.org/abs/0710.3742
2. Adiga, S., Tandon, R.: Unsupervised change detection using dre-cusum. arXiv preprint arXiv:2201.11678 (2022)
3. Ahamed, R., Lavin, A., Purdy, S., Agha, Z.: Unsupervised real-time anomaly detection for streaming data. Neurocomputing **262**, 134–147 (2017). https://doi.org/10.1016/j.neucom.2017.04.070
4. Aminikhanghahi, S., Cook, D.J.: A survey of methods for time series change point detection. Knowl. Inf. Syst. **51**(2), 339–367 (2016). https://doi.org/10.1007/s10115-016-0987-z
5. Aminikhanghahi, S., Wang, T., Cook, D.J.: Real-time change point detection with application to smart home time series data. IEEE Trans. Knowl. Data Eng. **31**(5), 1010–1023 (2019). https://doi.org/10.1109/TKDE.2018.2850347
6. Athey, S., Tibshirani, J., Wager, S.: Generalized random forests (2019)
7. van den Burg, G.J., Williams, C.K.: An evaluation of change point detection algorithms. arXiv, pp. 1–33 (2020)
8. Camci, F.: Change point detection in time series data using support vectors. Int. J. Pattern Recognit. Artif. Intell. **24**(01), 73–95 (2010)
9. Fearnhead, P., Rigaill, G.: Changepoint detection in the presence of outliers. J. Am. Stat. Assoc. **114**(525), 169–183 (2019)
10. Fryzlewicz, P.: Wild binary segmentation for multiple change-point detection. Ann. Stat. **42**(6), 2243–2281 (2014)
11. Goodfellow, I., et al.: Generative adversarial nets. In: Advances in Neural Information Processing Systems, pp. 2672–2680 (2014)
12. Katser, I., Kozitsin, V., Lobachev, V., Maksimov, I.: Unsupervised offline changepoint detection ensembles. Appl. Sci. **11**(9), 1–19 (2021). https://doi.org/10.3390/app11094280
13. Katser, I.D., Kozitsin, V.O.: Skoltech anomaly benchmark (SKAB) (2020). https://www.kaggle.com/dsv/1693952. https://doi.org/10.34740/KAGGLE/DSV/1693952
14. Killick, R., Fearnhead, P., Eckley, I.A.: Optimal detection of changepoints with a linear computational cost. J. Am. Stat. Assoc. **107**(500), 1590–1598 (2012)
15. Knoblauch, J., Damoulas, T.: Spatio-temporal Bayesian on-line changepoint detection with model selection. In: International Conference on Machine Learning, pp. 2718–2727. PMLR (2018)

16. Knoblauch, J., Jewson, J.E., Damoulas, T.: Doubly robust Bayesian inference for non-stationary streaming data with divergences. In: Advances in Neural Information Processing Systems, vol. 31 (2018)
17. Lavin, A., Subutai, A.: Numenta anomaly benchmark. In: International Conference on Machine Learning and Applications, vol. 14 (2015)
18. Li, D., Chen, D., Shi, L., Jin, B., Goh, J., Ng, S.K.: MAD-GAN: multivariate anomaly detection for time series data with generative adversarial networks. arXiv, vol. 1, pp. 703–716 (2019)
19. Liu, Y.W., Chen, H.: A fast and efficient change-point detection framework based on approximate k-nearest neighbor graphs. arXiv preprint arXiv:2006.13450 (2020)
20. Miller, D.J., Ghalyan, N.F., Mondal, S., Ray, A.: Hmm conditional-likelihood based change detection with strict delay tolerance. Mech. Syst. Signal Process. **147**, 107109 (2021)
21. Page, E.S.: Continuous inspection schemes. Biometrika **41**(1/2), 100–115 (1954)
22. Reddy, S., Mun, M., Burke, J., Estrin, D., Hansen, M., Srivastava, M.: Using mobile phones to determine transportation modes. ACM Trans. Sen. Netw. **6**(2) (2010). https://doi.org/10.1145/1689239.1689243
23. Reeves, J., Chen, J., Wang, X.L., Lund, R., Lu, Q.Q.: A review and comparison of changepoint detection techniques for climate data. J. Appl. Meteorol. Climatol. **46**(6), 900–915 (2007)
24. Schäfer, P., Ermshaus, A., Leser, U.: ClaSP - time series segmentation. In: CIKM (2021)
25. Taylor, S.J., Letham, B.: Business time series forecasting at scale. PeerJ Preprints 5:e3190v2 **35**(8), 48–90 (2017)
26. Tran, D.H.: Automated change detection and reactive clustering in multivariate streaming data. In: 2019 IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF), pp. 1–6. IEEE (2019)
27. Zhang, G., Patuwo, B.E., Hu, M.Y.: Forecasting with artificial neural networks: the state of the art. Int. J. Forecast. **14**(1), 35–62 (1998). https://doi.org/10.1016/S0169-2070(97)00044-7

Hierarchical Graph Neural Network with Cross-Attention for Cross-Device User Matching

Ali Taghibakhshi[1,2(✉)], Mingyuan Ma[1], Ashwath Aithal[1], Onur Yilmaz[1], Haggai Maron[1], and Matthew West[2]

[1] NVIDIA, Santa Clara, USA
[2] Department of Mechanical Science and Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, USA
alit2@illinois.edu

Abstract. Cross-device user matching is a critical problem in numerous domains, including advertising, recommender systems, and cybersecurity. It involves identifying and linking different devices belonging to the same person, utilizing sequence logs. Previous data mining techniques have struggled to address the long-range dependencies and higher-order connections between the logs. Recently, researchers have modeled this problem as a graph problem and proposed a two-tier graph contextual embedding (TGCE) neural network architecture, which outperforms previous methods. In this paper, we propose a novel hierarchical graph neural network architecture (HGNN), which has a more computationally efficient second level design than TGCE. Furthermore, we introduce a cross-attention (Cross-Att) mechanism in our model, which improves performance by 5% compared to the state-of-the-art TGCE method.

Keywords: Graph neural network · User matching · Cross-attention

1 Introduction

Ensuring system security and effective data management are critical challenges in the modern day [3,4]. In this regard, data integration plays a vital role in facilitating data management, as it enables the integration of data from diverse sources to generate a unified view of the underlying domain. One of the primary challenges in data integration is the problem of entity resolution, which involves identifying and linking multiple data records that correspond to the same real-world entity. The problem of entity resolution arises in a wide range of domains, including healthcare, finance, social media, and e-commerce. Entity resolution is a challenging problem due to various factors, including the presence of noisy and ambiguous data, the lack of unique identifiers for entities, and the complexity of the relationships between different entities.

A. Taghibakhshi—This work was done while Ali Taghibakhshi was an intern at NVIDIA.

R. Wrembel et al. (Eds.): DaWaK 2023, LNCS 14148, pp. 303–315, 2023.
https://doi.org/10.1007/978-3-031-39831-5_28

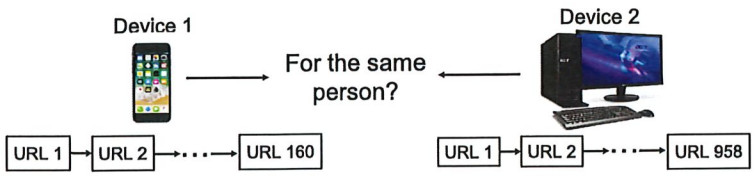


Fig. 1. Cross-device user matching problem: only based the URL visit logs of two different devices, determine whether or not they belong to the same real-world person.

Among entity resolution tasks, cross-device user matching is of significant importance. This task involves determining whether two separate devices belong to the same real-world person based on their sequential logs. The device sequential logs are time-stamped actions taken by the user over a relatively long period of time, say a few months. These actions are often in the form of browsing a Uniform Resource Locator (URL), and almost always, user identifications are not available due to privacy reasons. Refer to Fig. 1 for an illustration of the cross-device user matching task.

It is a common occurrence for users to engage in online activities across multiple devices. However, businesses and brands often struggle with having insufficient user identities to work with since users are perceived as different individuals across different devices due to their unique activities. The ability to automatically identify the same user across multiple devices is essential for gaining insights into human behavior patterns, which can aid in applications such as user profiling, online advertising, improving system security. In recent years, the has been a flourishing amount of studies focusing on cross-device user matching [9].

With the advent of machine learning-based methods for entity resolution, several studies have focused on learning distributed embeddings for the devices based on their URL logs [6,11,12]. The earlier studies focused on utilizing unsupervised feature learning techniques [7], developing handcrafted features for the device logs, or relied on co-occurrence of key attributes of URL logs in pairwise classification [12].

Methods that utilize deep learning have a greater ability to convey dense connections among the sequential device logs. For instance, researchers have utilized a 2D convolutional neural network (CNN) framework to encode sequential log representations to understand the relationship between two devices [16]. However, this model primarily captures local interactions within user sequence logs, limiting its ability to learn the entire sequence or a higher-level pattern. Recently, there has been further emphasis on the effectiveness of sequential models like recurrent neural networks (RNNs) and attention-based techniques in modeling sequence patterns and achieving promising results in numerous sequence modeling tasks [5,13,15]. Although these methods work well for sequence modeling, they are not specifically designed for user-matching tasks and may not be optimal for learning sequential log embeddings.

Recently, researchers proposed a two-tier graph contextual embedding (TGCE) network for the cross-device user matching [6] task. While previous methods for the task often failed at long-range information passing along the sequence logs, TGCE leverages a two-level structure that can facilitate information passing beyond the immediate neighborhood of a device log. This was specifically achieved by considering a random walk starting from every node in a device log, connecting all of the visited nodes to the original node, and performing a round of message passing using the newly generated shortcut edges.

Although the two-tier structure seems to enable long-range information sharing, we note two major limitations with the existing method. First, in the device graph, the random walk on the URL nodes may randomly connect two URLs that have been visited at two far-away time-stamps. Intuitively, two different URLs browsed by a device with weeks of gap in between share less information than two URLs visited in a shorter time frame. Second, at the end of the TGCE architecture, for the pairwise classification task, the generated graph embeddings for two devices are entry-wise multiplied and sent through a fully connected network to determine if they belong to the same person. However, there could be significant key features in the learned embeddings that may be shared between the devices, which can alternatively get lost if the architecture does not compare them across one another.

To address the above two issues, we propose a new hierarchical graph neural network (HGNN) inspired by the star graph architecture [10]. In the terminology of HGNN, we refer to the URL nodes as *fine* nodes, and in an unraveled sequence of URL logs, HGNN assigns a *coarse* node to every K consecutive fine nodes. The message passing between the coarse and fine nodes enables effective long-range message passing without the need to excessively add edges, as in the random walk method. Moreover, for the pairwise classification task, we utilize a cross-attention mechanism inspired by Li *et al.* [8], which enables entry-wise cross-encoding of the learned embeddings. The main contributions of this paper are summarized as follows:

- We model a given device log as a hierarchical heterogeneous graph, which is 6x faster than the previous state-of-the-art while keeping a competitive level of accuracy and performance.
- We employ a cross-attention mechanism for pairwise matching of the graphs associated with a device log, which improves the accuracy of the overall method by about 5%.

2 Related Work

The cross-device user matching task was first introduced in the CIKM Cup 2016[1] on the Data Centric Alliance (DCA) dataset, and the first proposed methods for the task mainly considered hand-crafted features. For instance, the runner-up solution [9] produces sub-categories based on the most significant URLs to

[1] http://cikm2016.cs.iupui.edu/cikm-cup/.

generate detailed features. Furthermore, the competition winner solution proposed by Tay *et al.* [14] utilizes "term frequency inverse document frequency" (TD-IDF) features of URLs and other related URL visit time features. However, their manually designed features did not fully investigate more intricate semantic details, such as the order of behavior sequences, which restricted their effectiveness. Aside from the hand-crafted features, the features that are developed from the structural information of the device URL visit data are also crucial for accomplishing the task of user matching. To further process sequential log information, studies have applied LSTM, 2D-CNN, and Doc2vec to generate semantic features for a sequence visited by a device [11,12,16].

Sequence-based machine learning models have also been employed for different entity resolution tasks; for instance, recurrent neural networks (RNN) have been utilized to encode behavior item sequential information [5]. Nevertheless, long-range dependencies and more advanced sequence features are not well obtained using sequence models [6]. With the advent of graph neural networks (GNN), studies have leveraged their power for many entity resolution tasks. By utilizing neighborhood-based aggregation, GNNs effectively capture and propagate structural information, which enables them to perform excellently in numerous tasks such as node and graph classification. In order to employ GNNs, researchers have modeled device logs as individual graphs where nodes and edges represent visited URLs and transitions between URLs. Each node and/or edge has an initial feature vector obtained from the underlying problem, and the layers of GNNs are then employed to update these features based on information passing in the local neighborhood of every node, such as the SR-GNN paper [17]. Another example is the LESSR [1] method for recommendation systems where the method is capable of long-range information capturing using an edge-order preserving architecture. However, these methods are specifically designed for the recommendation task and do not necessarily achieve desirable results on the cross-device user matching task.

Recently, researchers have proposed TGCE [6], a two-tier GNN for the cross-device user matching task. In the first tier, for every device log, each URL is considered as a node, and directional edges denote transitions between URLs. In the second tier, shortcut edges are formed by starting a random walk from every node and connecting all of the visited nodes to it. After a round of message passing in the first tier, the second tier is supposed to facilitate long-range information sharing in the device log. After the second tier, a position-aware graph attention layer is applied, followed by an attention pooling, which outputs the learned embedding for the whole graph. For the final pairwise classification, these learned embeddings for each of the devices are multiplied in an entry-wise manner and are sent to a fully connected deep neural network to determine whether they belong to the same user.

3 Hierarchical Graph Neural Network

In this section, we discuss how we employ a two-level heterogeneous graph neural network for the cross-device user matching problem.

3.1 Problem Definition

The aim of the cross-device user matching problem is to determine whether two devices belong to the same user, given only the URL visits of each device. Denote a sequence of visited URLs by a device v by $\mathcal{S}_v = (s_1, s_2, ..., s_n)$, where s_i denotes the i'th URL visit by the device (note that s_i and s_j are not necessarily different, for $i \neq j$). We build a hierarchical heterogeneous graph, G_v, based on the sequence $\mathcal{S}_v$ as follows: for a visited URL, s_i, consider a *fine* node in G_v and denote it by f_i. Note that if multiple URL visits (s_i) correspond to the same URL, we only consider one node for it in G_v. Then, we connect nodes corresponding to consecutively visited URLs by directed edges in the graph (preferred over undirected edges to emphasize on the temporal order of URL visits); we connect f_i and f_{i+1} by a directional edge (if f_i and f_{i+1} correspond to the same URL, the edge becomes a self-loop). Up to this point, we have defined the fine-level graph, and we are ready to construct the second level, which we call the *coarse* level.

To construct the second level, we partition the sequence $\mathcal{S}_v$ into non-overlapping subgroups of K URLs (where K is a hyperparameter), where each subgroup consists of consecutively visited URLs (the last subgroup may have less than K URLs). For every subgroup j, we consider a coarse node, c_j, and connect it to all of the fine nodes corresponding to the URLs in subgroup j via undirected edges.

3.2 Fine Level

In the fine level of the graph G_v, for every node f_i, we order the nodes corresponding to the URLs that have an incoming edge to f_i according to their position in $\mathcal{S}_v$. We denote this ordered sequence of nodes by $N_i = (f_{j_1}, f_{j_2}, ..., f_{j_\kappa})$. Also, we denote the feature vector of the fine node f_i by x_i. The l-th round of message passing in the fine-level graph updates the node features according to the following update methods:

$$M_i^{(l)} = \Phi^{(l)}([x_{j_1}, x_{j_2}, ..., x_{j_\kappa}, x_i]), \tag{1}$$

$$x_i^{(l+1)} = \Psi^{(l)}(x_i^{(l)}, M_i^{(l)}), \tag{2}$$

where $\Phi^{(l)}$ is a sequence aggregation function (such as sum, max, GRU, LSTM), for which we use GRU [2] (to leverage temporal order of URLs in encoding), and $\Psi^{(l)}$ is a function for updating the feature vector (e.g., a neural network), for which we use a simple mean.

3.3 Coarse Level

In every round of heterogeneous message passing between fine and coarse level nodes, we update both the fine and coarse node features. Consider the coarse node c_j, and denote its feature by $\tilde{x}_j$. Also, denote the fine neighbor nodes of c_j

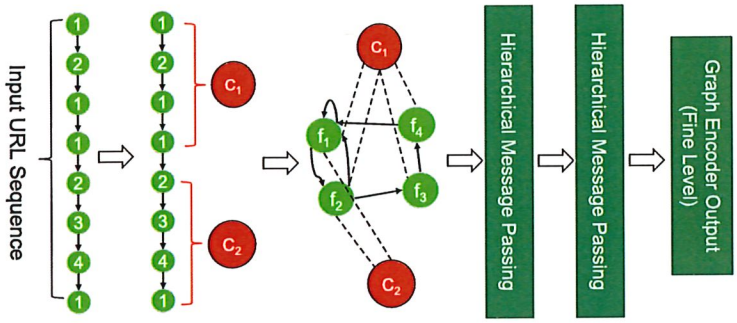


Fig. 2. From left to right: heterogeneous (fine and coarse) graph modeling from a given URL sequence. The hierarchical message passing blocks consist of message passing on the fine nodes with a GRU aggregation function. Next, the coarse node features are updated using a mean aggregation function. Finally, the fine node features are updated using their previous feature vector as well as an aggregated message from their associated coarse nodes obtained via an attention mechanism between coarse and fine level nodes.

by $\tilde{\mathcal{N}}(c_j)$. In the l-th layer of heterogeneous message passing, the coarse node feature update is as follows:

$$\tilde{x}_j^{(l+1)} = \underset{i \in \tilde{\mathcal{N}}(c_j)}{\square} (W_1^{(i)} x_i), \tag{3}$$

where $W_1^{(l)}$ is a learnable matrix and $\square$ is an aggregation function (such as mean, max, sum), for which we use mean (which was the most effective in our case). Denote by $\mathcal{N}(f_i)$ the set of coarse nodes connected to the fine node f_i. We first learn attention weights for the heterogeneous edges, and then we update fine nodes accordingly. In the l-th round of heterogeneous message passing, the fine node features are updated as follows:

$$e_{i,j}^{(l)} = \phi(W_2^{(l)} x_i^{(l)}, W_3^{(l)} \tilde{x}_j^{(l)}), \tag{4}$$

$$\alpha_{i,j}^{(l)} = \frac{\exp(e_{i,j}^{(l)})}{\sum_{j \in \mathcal{N}(f_i)} \exp(e_{i,j}^{(l)})}, \tag{5}$$

$$x_i^{(l+1)} = \xi(x_i^{(l)}, \sum_{j \in \mathcal{N}(f_i)} \alpha_{i,j}^{(l)} \tilde{x}_j^{(l)}), \tag{6}$$

where $W_2^{(l)}$ and $W_3^{(l)}$ are learnable matrices, and ξ and ϕ are update functions (such as a fully connected network). Figure 2 shows the overall architecture of fine and coarse level message passing.

3.4 Cross Attention

After the message passing rounds in the fine level and long-range information sharing between fine and coarse nodes, we extract the learned fine node embeddings and proceed to cross encoding and feature filtering, inspired by the GraphER architecture [8]. We consider two different device logs v and w, and their learned fine node embeddings as a sequence, ignoring the underlying graph structure. We denote the learned fine node embeddings for device logs v and w as $X_v \in \mathbb{R}^{m_v \times d}$ and $X_w \in \mathbb{R}^{m_w \times d}$, where m_v and m_w are the number of nodes in the fine level of G_v and G_w, respectively. We learn two matrices for cross-encoding X_v into X_w and vice-versa. Consider the i-th and j-th rows of X_v and X_w, respectively, and denote them by $x_{v,i}$ and $x_{w,j}$. The entries $\hat{\alpha}_{i,j}$ of the matrix $A_{v,w}$ for cross-encoding X_v into X_w are obtained using an attention mechanism (and similarly for $A_{w,v}$):

$$\hat{e}_{i,j} = \zeta(W_3 x_{v,i}, W_3 x_{w,j}), \tag{7}$$

$$\hat{\alpha}_{i,j} = \frac{\exp(\hat{e}_{i,j})}{\sum_{k=1}^{m_w} \exp(\hat{e}_{i,k})}, \tag{8}$$

where ζ is an update function (such as a neural network), for which we use a simple mean. After obtaining the cross-encoding weights, we apply feature filtering, a self-attention mechanism that filters important features. The filtering vector is obtained as $\beta_v = \text{sigmoid}(W_4 \tanh(W_5 X_v^T))$, where W_4 and W_5 are learnable weights (β_w is obtained similarly). We apply the feature-filtering vector to the cross-encoding matrix as follows:

$$L_{v,w} = [\text{diag}(\beta_v)(A_{v,w}X_w - X_v)] \odot [\text{diag}(\beta_v)(A_{v,w}X_w - X_v)], \tag{9}$$

where $\odot$ denotes the Hadamard product ($L_{w,v}$ is also obtained similarly). The $L_{v,w} \in \mathbb{R}^{m_v \times d}$ and $L_{w,v} \in \mathbb{R}^{m_w \times d}$ matrices come from the Euclidean distance between the cross-encoding of X_v into X_w and X_w into X_v, and therefore are a measure of the closeness of the original sequence logs of v and w.

To obtain a size-independent comparison metric, we apply a multi-layer perceptron (MLP) along the feature dimension of L matrices (the second dimension, d), followed by a max-pooling operation along the first dimension. Finally, we apply a dropout and a ReLU nonlinearity. This yields vectors $r_{v,w}$ and $r_{w,v}$ that have a fixed size for any pair of v and w. For the final pairwise classification task, we concatenate $r_{v,w}$ and $r_{w,v}$ and pass it through an MLP followed by a sigmoid activation to determine if the two devices belong to the same user or not:

$$\hat{y} = \text{sigmoid}(\text{MLP}(r_{v,w} || r_{w,v})). \tag{10}$$

The overall cross attention architecture employed in this paper is illustrated in Fig. 3.

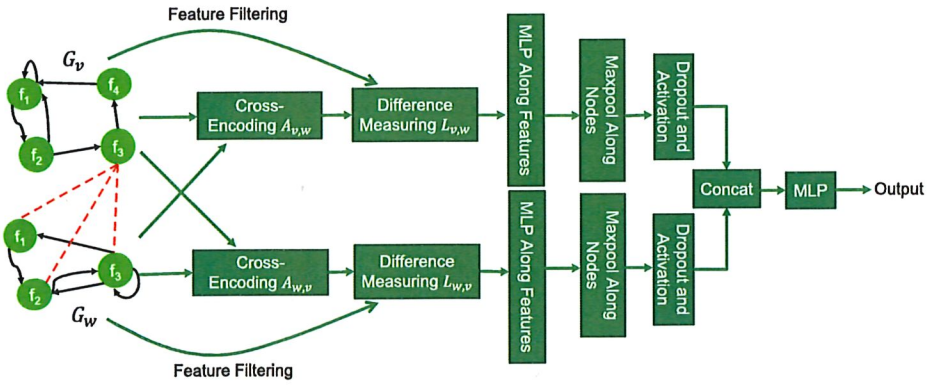


Fig. 3. Pairwise device graph matching: After the message passing, the two device graphs are cross-encoded via an attention mechanism followed by an attention-based feature filtering. The resulting matrix for each graph is then passed through an MLP layer, acting along the feature, followed by a maxpool operator along the nodes. Next, the obtained vectors pass through a dropout layer followed by an activation function. Finally, the resulting vectors of the two graphs are concatenated and passed through an MLP to obtain the final output.

4 Experiment

In this section, we will describe the dataset, training details, and discuss how our method outperforms all other baselines, including TGCE [6], the previous state-of-the-art.

4.1 Training Details

We studied the cross-device user matching dataset made publicly available by the Data Centric Alliance[2] for the CIKM Cup 2016 competition. The dataset consists of 14,148,535 anonymized URL logs of different devices with an average of 197 logs per device. The dataset is split into 50,146 and 48,122 training and test device logs, respectively. To obtain the initial embeddings of each URL, we applied the same data preprocessing methods as in [6,11]. We used a coarse-to-fine node ratio of $K = 6$, a batch size of 800 pairs of device logs, a learning rate of 10^{-3}, and trained the model for 20 epochs (the hyperparameters are fixed for optimal perfomance of the model using grid search). We used the binary cross-entropy (BCE) loss function for training our model. The training, evaluation, and test were all executed on an A100 NVIDIA GPU. The BCE loss during training as well as the validation F1 score are shown in Figs. 4 and 5, respectively.

[2] https://competitions.codalab.org/competitions/11171.

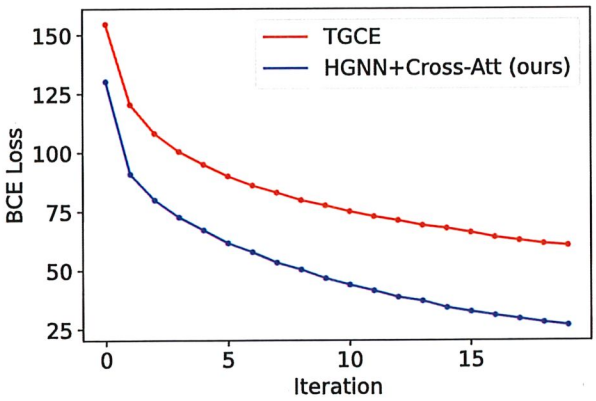


Fig. 4. Binary cross-entropy loss of our proposed method against that of TGCE. During training, our method obtains strictly better loss values.

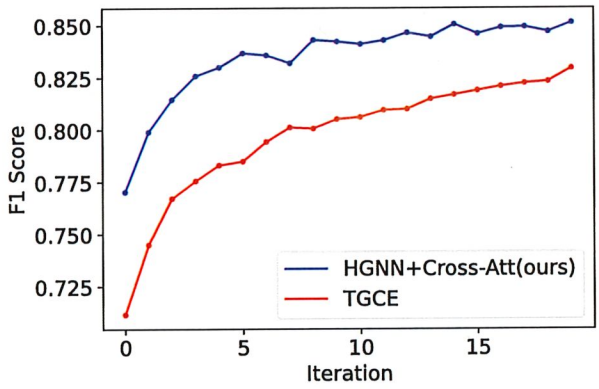


Fig. 5. Validation F1 score during training. Throughout the training, our method achieves strictly better F1 scores for the validation set compared to that of TGCE.

4.2 Results

In this section, we evaluate the precision, recall, and F1 score of our method on the test set and compare it to available baselines. All of the baselines have been obtained similarly as described in [6]. We present two variants of our method; the first one, which we label "HGNN", only differs from TGCE in the design of the second tier, i.e., we use the hierarchical structure presented in Subsects. 3.2 and 3.3, followed by the rest of the TGCE architecture. The second variant, which we label "HGNN+Cross-Att", uses the hierarchical structure in Subsects. 3.2 and 3.3, and also utilizes the cross-attention mechanism presented in Subsect. 3.4 after the hierarchical structure. As shown in Table 1, the "HGNN+Cross-Att" variant outperforms all of the baselines on the F1 score metric, including the second-best method (TGCE) by 5% on the test data.

Table 1. Precision, Recall, and F1 score of different methods for cross-device user matching on DCA dataset.

	Precision at Best F1 Score	Recall at Best F1 Score	Best F1 Score
TF-IDF	0.33	0.27	0.26
Doc2vec	0.29	0.21	0.24
SCEmNet	0.38	0.44	0.41
GRU	0.37	**0.49**	0.42
Transformer	0.39	0.47	0.43
SR-GNN	0.35	0.34	0.34
LESSER	0.41	**0.48**	0.44
TGCE	0.49	0.44	0.46
HGNN (ours)	0.48	0.43	0.45
HGNN+Cross-Att (ours)	**0.57**	**0.48**	**0.51**

We also compare the training time of the two variants of our method with that of TGCE. As shown in Table 2, our hierarchical structure is significantly more efficient than that of TGCE while keeping a competitive F1 score. Table 2 essentially indicates that by simply replacing the second-tier design of TGCE with our hierarchical structure (presented in Subsects. 3.2 and 3.3), the method becomes 6x faster while almost keeping the same performance. This is due to the large number of artificial edges generated in the random walk passes in the creation of the second tier of TGCE. Moreover, although including cross-attention slows down the model, we can still obtain the same training time as TGCE and achieve 5% better overall F1 score.

Table 2. Best F1 score and end-to-end training time of HGNN (without Cross-Att), HGNN+Cross-Att, and TGCE. The HGNN model is 6x faster than TGCE with a slight trade-off (about 1%) on the accuracy side. The HGNN+Cross-Att model has the same training time as TGCE while achieving 5% better F1 score.

	Best F1 Score	End-to-end Training Time	Number of Epochs
TGCE	0.46	60 h	20
HGNN (ours)	0.45	**10 h**	20
HGNN+Cross-Att (ours)	**0.51**	60 h	**6**

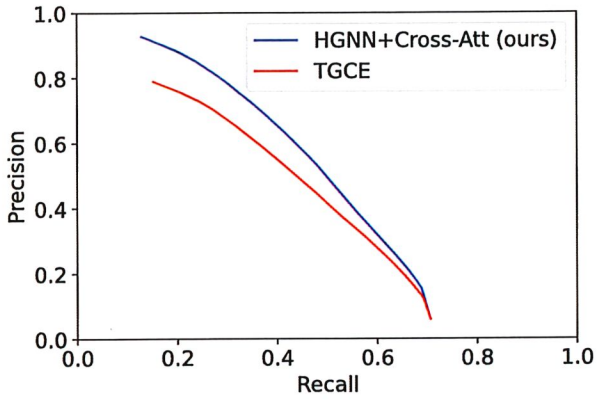


Fig. 6. Precision-Recall curve of the proposed method and that of TGCE on the test data.

Figure 6 shows the precision-recall curve of our method (the HGNN+Cross-Att variant, trained for 6 epochs) with that of TGCE (trained for 20 epochs). As shown in the figure, the precision-recall curve of our method is strictly better than that of TGCE. In other words, for every recall score, our method has a better precision. Additionally, we further trained the HGNN+Cross-Att variant for 20 epochs (the same number of epochs TGCE was trained for) to study if any further improvement is achieved on the test set. We also plot the F1 score with different thresholds (from 0 to 1 incremented by 0.01) for our model trained for 6 and 20 epochs and compare it to that of TGCE. As shown in Fig. 7, our model trained for 20 epochs strictly outperforms TGCE (also trained for 20 epochs) for every threshold for obtaining the F1 score. We note that for the full 20 epoch training, our model would require about three times more training time, and achieves F1 score of just about two percent higher than TGCE. Nevertheless, after training our model for six epochs, it achieves the highest overall F1 score, surpassing TGCE by a remarkable 5%. This achievement is particularly noteworthy considering that, as indicated in Table 2, our model requires the same amount of time to train as TGCE. Hence, we conclude that our model only needs 6 training epochs to achieve the best perfomance and training it for 20 epochs results in overfitting as discussed.

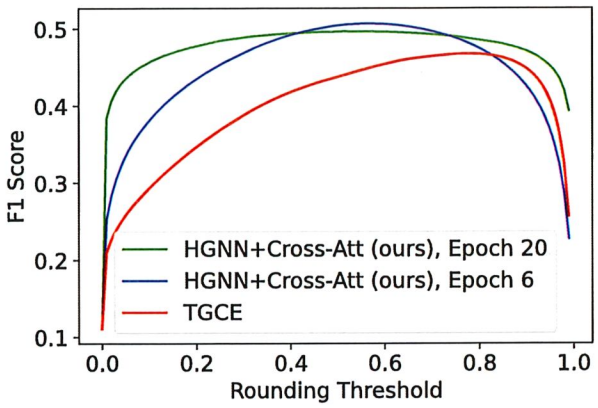


Fig. 7. F1 score against rounding threshold for our method (both for networks trained for 6 and 20 epochs) compared to that of TGCE (trained for 20 epochs).

5 Conclusions

In this paper, we present a novel graph neural network (GNN) architecture for a demanding entity resolution task: cross-device user matching, which determines if two devices belong to the same user based only on their anonymized internet logs. Our method comprises of designing an effective hierarchical structure for achieving long-range message passing in the graph obtained from device URL logs. After passing device logs through such a hierarchical GNN, we employ a cross-attention mechanism to effectively compare device logs against each other to determine if they belong to the same user. We demonstrate that our method outperforms available baselines by at least 5%, while having the same training time as the previous state-of-the-art method, establishing the effectiveness of our proposed method.

Acknowledgements. This research was supported by NVIDIA Corporation.

References

1. Chen, T., Wong, R.C.W.: Handling information loss of graph neural networks for session-based recommendation. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1172–1180 (2020)
2. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555 (2014)
3. Gharaibeh, A., et al.: Smart cities: a survey on data management, security, and enabling technologies. IEEE Commun. Surv. Tutor. **19**(4), 2456–2501 (2017)
4. Gholizadeh, N.: Iec 61850 standard and its capabilities in protection systems (2016)
5. Hidasi, B., Karatzoglou, A., Baltrunas, L., Tikk, D.: Session-based recommendations with recurrent neural networks. arXiv preprint arXiv:1511.06939 (2015)

6. Huang, H., et al.: Two-tier graph contextual embedding for cross-device user matching. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pp. 730–739 (2021)
7. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: International Conference on Machine Learning, pp. 1188–1196. PMLR (2014)
8. Li, B., Wang, W., Sun, Y., Zhang, L., Ali, M.A., Wang, Y.: Grapher: token-centric entity resolution with graph convolutional neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 8172–8179 (2020)
9. Lian, J., Xie, X.: Cross-device user matching based on massive browse logs: the runner-up solution for the 2016 cikm cup. arXiv preprint arXiv:1610.03928 (2016)
10. Pan, Z., Cai, F., Chen, W., Chen, H., De Rijke, M.: Star graph neural networks for session-based recommendation. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, pp. 1195–1204 (2020)
11. Phan, M.C., Sun, A., Tay, Y.: Cross-device user linking: url, session, visiting time, and device-log embedding. In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 933–936 (2017)
12. Phan, M.C., Tay, Y., Pham, T.A.N.: Cross device matching for online advertising with neural feature ensembles: first place solution at cikm cup 2016. arXiv preprint arXiv:1610.07119 (2016)
13. Qiu, R., Yin, H., Huang, Z., Chen, T.: Gag: global attributed graph neural network for streaming session-based recommendation. In: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 669–678 (2020)
14. Ramos, J., et al.: Using tf-idf to determine word relevance in document queries. In: Proceedings of the First Instructional Conference on Machine Learning, vol. 242, pp. 29–48. Citeseer (2003)
15. Sun, F., et al.: Bert4rec: sequential recommendation with bidirectional encoder representations from transformer. In: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, pp. 1441–1450 (2019)
16. Tanielian, U., Tousch, A.M., Vasile, F.: Siamese cookie embedding networks for cross-device user matching. In: Companion Proceedings of the the Web Conference 2018, pp. 85–86 (2018)
17. Wu, S., Tang, Y., Zhu, Y., Wang, L., Xie, X., Tan, T.: Session-based recommendation with graph neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 346–353 (2019)