Student Research Abstract: Using Local Activity Encoding for Dynamic Graph Pooling in Stuctural-Dynamic Graphs

Silvia Beddar-Wiesing
University of Kassel
Kassel, Germany
s.beddarwiesing@uni-kassel.de

ABSTRACT

Graphs have recently become more popular for modeling real-world and theoretical problems, and several Machine Learning algorithms for learning on them have been developed. However, research is still at an early stage for graphs that change over time. This article proposes a framework **DynHeatmaP** that generates a constant-size graph sequence representation of graphs whose structure changes over time (structural-dynamic graph) in linear time. The framework uses a representation of additions and deletions of nodes and edges (events) to perform dynamic graph pooling. Our algorithm translates the **dyn**amic changes of the nodes and edges to a **heatmap** representation of the current node activity regarding additions and deletions in the neighborhood, respectively. The created activity heatmap is utilized as probabilities of the nodes to be sampled in the subsequent **p**ooling step. In addition, the neighborhood of selected nodes has also been sampled such that the ratio of additions and deletions is represented accordingly. The obtained down-sampled graph sequence can then be used as representative input to a GNN for learning on the original stream. Focusing on the active regions in a graph, the fast pooling process can reduce the training time significantly and addresses the problem of unbalanced data sets in real-world applications modeled as sparse graphs.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; • **Computing methodologies** → **Feature selection**; *Neural networks*; *Batch learning*;

KEYWORDS

structural-dynamic graph, dynamic graph representation, dynamic graph pooling, temporal feature selection

ACM Reference Format:
Silvia Beddar-Wiesing. 2022. Student Research Abstract: Using Local Activity Encoding for Dynamic Graph Pooling in Stuctural-Dynamic Graphs. In *The 37th ACM/SIGAPP Symposium on Applied Computing (SAC '22), April 25–29, 2022, Virtual Event, .* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3477314.3506969

SAC '22, April 25–29, 2022, Virtual Event,

ACM ISBN 978-1-4503-8713-2/22/04.
https://doi.org/10.1145/3477314.3506969

PROBLEM AND MOTIVATION

The goal is to design an efficient representation of structural-dynamic graphs to enable the learning of structural and temporal patterns and behavior. A **structural-dynamic graph** is a dynamic graph defined by a set of nodes $\mathcal{V} \subset \mathbb{N}$ and a set of edges $\mathcal{E}$[1] that change over time. In addition, either the nodes or the edges, or both, can have attributes that are assumed to be static in the structural-dynamic setting.

On the one hand, dynamic graphs can be represented as a sequence of T graphs $\mathcal{G} = (g_1, \ldots, g_T)$ called **graph snapshots** $g_t = (\mathcal{V}_t, \mathcal{E}_t)$ including a set of nodes and edges at each timestamp $t \in \{1, \ldots, T\} = [T]$ [4]. On the other hand, dynamic graphs can be available in form of **graph streams** (sometimes called continuous-time representation or event-based representation) $\mathcal{G} = (\mathcal{G}_0, O)$ consisting of a (possibly empty) start graph $\mathcal{G}_0 = (\mathcal{V}_0, \mathcal{E}_0)$ with start nodes and edges and a set of T events (or observations) $O = (o_1, \ldots, o_T)$ with corresponding timestamps $t \in [T]$ that can consist of additions or deletions of nodes or edges [11].

Especially the representation of deletions in a graph turned out to be challenging. The inconsistent dimension of the adjacency matrix is difficult to handle, and the neighborhoods and the employed information of the deleted nodes have to be appropriately updated.

In addition, the size of real-world graphs is challenging to process. The increasing mass of data leads to many nodes but possibly a relatively small number of edges resulting in a sparse graph. On the one hand, common representation approaches such as the adjacency matrix of a graph are insufficient due to the redundant storage of non-existing edges. On the other hand, in specific graph learning tasks such as link prediction, where the task is to predict the existing edges at the next timestamp, models suffer from unbalanced data. As a consequence of the sparseness, the graph information in a dynamic setting mostly lies in a few dense areas.

In our research, we address the representation of the events by introducing separate attributes for the additions and deletions in the local neighborhood of the nodes. The resulting heatmaps are utilized to select a preferential subset of active nodes via pooling for efficient Graph Neural Networks (GNNs) training to learn structural and temporal information. The entire procedure is illustrated in Fig. 1.

[1] Dependent on which type of graph is the basis for the dynamic graph, the edge set differs. For more information about the different graph types, see [22].

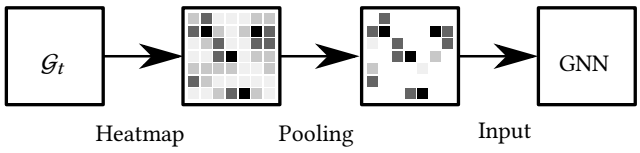

Figure 1: Structure of the DynHeatmaP module at timestamp t. Starting with the local activity encoding, the resulting heatmap comprises the dynamic of additions and deletions at a certain timestamp. Subsequently, the heatmap is utilized to sample preferential nodes and corresponding neighbors that can be used as input for a (static) GNN.

BACKGROUND AND RELATED WORK

Representing dynamic graphs for Machine Learning tasks is part of current research. Using Neural Networks for static graph problems has been investigated more intensely since the 1990s [2, 3, 19], whereas the focus on dynamic graphs started around 2017 [10, 23]. The problem is to model the structure and temporal information of dynamic graphs to enable learning of structural and temporal patterns. To encode structural information, a common approach uses the message-passing process that incorporates the information of neighboring nodes via convolution in the attributes of a node in a graph. The obtained representation can then be decoded by standard Neural Networks such as MLPs [17].

To learn the temporal behavior, several GNNs have been developed whereby different representation approaches are utilized depending on whether and how the nodes and edges change. For fixed nodes and a set number of changing edges between them, a convolutional GNN has been proposed [26]. For an arbitrary number of edges, a dynamic propagation scheme [5] or the combination of a Graph Convolutional and a Recurrent Neural Network [12] have been published. To learn on graph snapshots where the entire node set is given a Recurrent GNN limited to a sliding window has been proposed [21]. Furthermore, activity has been similarly defined for growing graphs as a node's influence in a communication network, where nodes and edges can only be added in an event [11]. Moreover, GNNs for graph streams allow for more efficient storage and model update by using, e.g., a Recurrent Update Unit [11] or Continual Learning [25].

However, most models cannot process changing node sets, especially the deletion of nodes, due to dimension inconsistencies of matrices in the network propagation. In addition, models that can address this problem, such as PINT [20] and the previous works [15, 24] improved in it, are hard to train since they usually utilize a Recurrent Neural Network to learn temporal dependencies, and dynamic graphs, especially in real-world applications, are large and sparse. In this article, we propose a preprocessing module **DynHeatmaP** that encodes the temporal activity of nodes in a structural-dynamic graph as a heatmap. The heatmap is used as an attribute representing the dynamic behavior close to the nodes regarding additions and deletions of nodes and edges separately for pooling nodes at a given timestamp.

Furthermore, the literature on graph pooling reveals that most graph pooling methods published yet are developed for static graphs and work globally or hierarchically [1, 7, 9, 14, 27]. To the best of our knowledge, only a few dynamic graph types were considered in recent works, none of them structural-dynamic. For the special case of spatio-temporal graphs representing positions and timestamps of events by snapshots connected over time, [18] provides a pooling strategy. The approach provided in [6] pools from the snapshots of a dynamic graph and remembers the positions of the discarded nodes to diffuse the graph information afterward. Thus, this model is limited to a fixed number of nodes per timestamp. Based on this model, the approach presented in [8] applies graph pooling on a graph with static structure but changing node attributes (graph signal). Therefore, to bridge the gap of pooling for graphs whose structure changes over time, this article aims to present a model that can both encode structural changes and serve as a pooling method for efficient learning on structural-dynamic graphs.

APPROACH AND UNIQUENESS

We first propose the local activity encoding module that fetches the events near all nodes at a timestamp and cumulates them in a four-dimensional attribute vector. Afterward, we present a pooling module that generates a preferential subset of nodes concerning the previously created heatmap, representing the dynamic behavior and the areas with high structural and temporal information density.

Local activity Encoding

Given a graph stream $\mathcal{G} = (\mathcal{G}_0, O)$ with events $O = (o_1, \ldots, o_T)$, and the total number of nodes n, the activity of a node at timestamp t is determined by nearby events $o_t \in O$ such as its own addition or deletion and current additions or deletions of incident edges. The pseudocode of the algorithm is given in Alg. 1. At the beginning, all existing nodes $v \in \mathcal{V}_0$ become fully active regarding additions, i.e., their activities $\alpha_+(v)$ are set to a maximum activity, while their activities regarding deletions $\alpha_-(v)$ are set to a constant inactivity value.

Algorithm 1 Local Activity Encoding

```
procedure LocalActivity(G = (V_0, E_0), O = (o_1, ..., o_T))
    for all v ∈ V_0 do
        // Start nodes are max. active
        update_actv(v, {add(v)})
    end for
    for t ∈ [T] do
        UpdateHeatmap(G_t, o_t)
    end for
end procedure
```

Then, the activity regarding additions or deletions is updated respectively, or both activities are down dated by a constant factor when no incident activities occur as illustrated in Alg. 2 and Fig. 2. Furthermore, a node can become *inactive* (regarding additions or deletions) when no incident events have occurred for some time and the activity value passes a threshold. Then, it is set to the inactivity value.

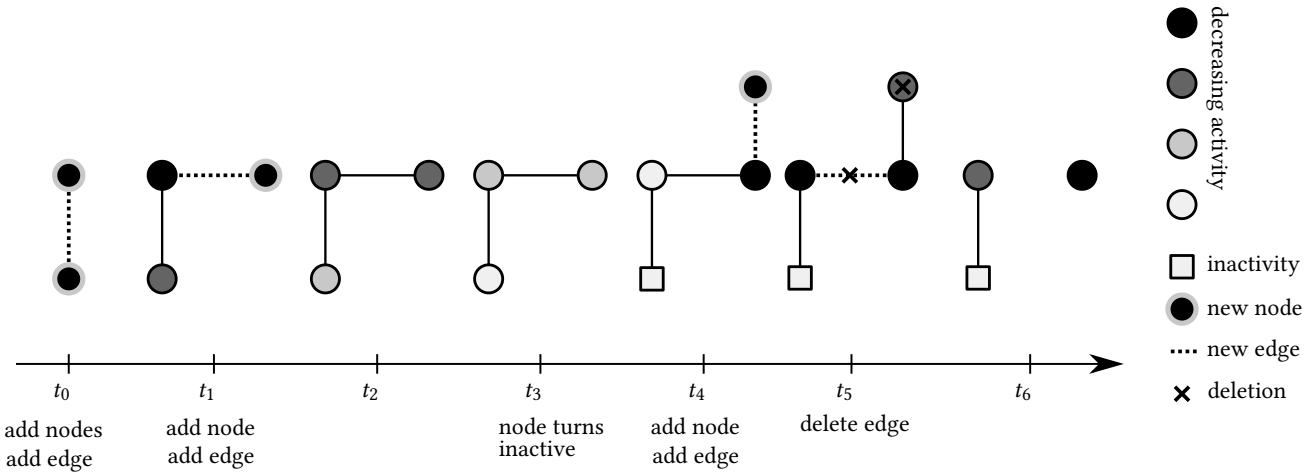

Figure 2: Local activity encoding for adding and deleting nodes and edges. At each timestamp, the corresponding activity is stored in the heatmap. Here, for simple illustration, the activity regarding additions and deletions is assumed to have the same values.

Algorithm 2 Update Heatmap

```
procedure UpdateHeatmap(G_t, o_t)
    for all v ∈ V_t do
        if v ∈ V_t(o_t) then
            // Set node activity to max.
            update_actv(v, o_t)
            // Update neighborhood activities reg. add. and del.
            update_neighborhood_actv(v)
        else
            // Downdate all nodes
            downdate_actv(v)
        end if
    end for
    return G_t
end procedure
```

Hence, nodes can have a high activity value for additions while having a low activity regarding deletions or vice versa.

To exploit the local activity of a node, the activity of the neighborhood is taken into account. For this, the neighbors' activities are normalized by the maximal activity and cumulated using a given aggregation function. When a neighbor is inactive, the constant inactivity value is added. Hence, the neighborhood activities of nodes with many highly active neighbors are high, while it is lower for nodes with few (active) neighbors or many inactive neighbors.

Dynamic Graph Pooling

Based on the obtained heatmap for the node set $\mathcal{V}_t$ of the input graph stream $\mathcal{G}$ at a timestamp t, a sequence of N_p pooling processes is performed in two steps each: First, a subset of preferential nodes $\mathcal{V}_p$ of size $N_n \in \mathbb{N}$ is sampled without lay back from the current node set $\mathcal{V}_t$ in pooling process $p \in [N_p]$. The sampling probability depends on the activity vector $\alpha(v)$ of each node $v \in \mathcal{V}_t$.

Simultaneously, the corresponding neighborhoods are collected in $\mathcal{N}$. Second, a subset of nodes is sampled from the neighborhood $\mathcal{N}$ without layback and added to the preferential nodes $\mathcal{V}_p$. The neighborhood is limited in average by the neighborhood sampling size $\lambda \in \mathbb{N}$, i.e., on average, $\lambda \cdot N_n$ many neighbors are drawn for each preferential node. Furthermore, the ratio $\rho \in [0, 1]$ controls the proportion of neighbors sampled w.r.t. the acivities of additions α_+ and deletions α_-. Moreover, the iterated pooling enables batching of the graph stream. I.e., by applying the pooling N_b times, a set of pooled graphs at timestamp t can be used as batch input into a GNN. The entire procedure is presented in Alg. 3.

The pooling process is easily extendable to graph streams with node or edge attributes or both. Furthermore, extensions to different neighborhood definitions and cumulation functions and other reductions and augmentations of the activities in the occurrence or absence of events are already intended.

Runtime Analysis

Let N be the total number of nodes in the entire graph stream $\mathcal{G} = (\mathcal{G}_0 = (\mathcal{V}_0, \mathcal{E}_0), O)$, i.e., the number of nodes occurring in the entire stream. The node activities are stored in hash maps to keep the memory low. Thus, the access to one node has a constant runtime, so the activity up- and down-dates can be done in $O(N)$ for all nodes in the worst-case. Updating the neighborhood activity for one node, however, again has a worst-case runtime of $O(N)$. In total, the generation of the heatmaps has a worst-case runtime of $O(N^2)$ for each timestamp.

Note that limiting the neighborhood in the update step to $k << N$ neighbors reduces the runtime to linear time $O(kN) \in O(N)$. Furthermore, inactive nodes can be excluded in the update process for further runtime reduction using a second hash map. So when there exists an upper bound $N_n << N$ for the active nodes for each timestamp, the runtime further reduces to $O(kN_n) \in O(1)$.

Algorithm 3 Dynamic Graph Pooling

procedure DynHeatmaP($\mathcal{G}$, N_n, N_p, λ, ρ)
  for all $t \in [T]$ **do**
    // Update node activities
    $\mathcal{G}_t$ = UpdateHeatmap($\mathcal{G}_t, o_t$)
    $pool\_t \leftarrow \emptyset$
    // For each pooling run
    for all $p \in [N_p]$ **do**
      // Initialize nodes and neighborhood
      $\mathcal{V}_p \leftarrow \emptyset$, $\mathcal{N} \leftarrow \emptyset$
      for all $i \in [N_n]$ **do**
        // Sample node from $\mathcal{G}_t$ w.r.t. heatmap without lay back
        $v \leftarrow sample(\mathcal{V}_t, \alpha(\mathcal{V}_t), \rho)$
        $\mathcal{V}_p.append(v)$
        $\mathcal{N}.append(neighborhood(v))$
      end for
      // Sample $\lambda \cdot N_n$ neighbors without lay back
      // addition/deletion ratio of ρ
      for all $j \in [\lambda \cdot N_n]$ **do**
        $w \leftarrow sample(\mathcal{N})$
        $\mathcal{V}_p.append(w)$
      end for
      $pool\_t.append(\mathcal{V}_p)$
    end for
  end for
end procedure

Each pooling run consists of sampling a fixed number of $N_n << N$ preferential nodes considering the heatmap and $\lambda \cdot N_n$ nodes from their neighbors, which in total yields in a runtime of $O((1+\lambda)N_n) \in O(1)$ for the sampling. Together with the heatmap update, the total worst-case runtime of DynHeatmaP is

$$O(N^2 + N), \text{ or } O(N),$$

using advanced implementation for each timestamp respectively. If there exists an upper bound of the active nodes for all timestamps, then the algorithm runs in $O(1)$.

APPLICATION

On the one hand, many datasets contain graph streams in which only one event occurs at any given time. These include, e.g., many dynamic networks from [16]. Hence, applying the algorithm for dynamic graph pooling using limited neighborhoods can be done in constant time for each timestamp here.

On the other hand, the proposed method implements a preprocessing step for handling structural-dynamic graphs. Since the inconsistent input dimensions complicate the processing of the input graph, many models omit the dynamics of the node sets, as stated in the related work. However, the DynHeatmaP algorithm produces a graph whose node set is static while the node attributes change over time. The obtained attribute-dynamic graph can be further processed for several graph learning applications, such as graph embedding and generation [13].

RESULTS AND CONTRIBUTIONS

Our representation and pooling module DynHeatmaP enables fast encoding of local activities of structural-dynamic graphs in node attributes. The module is applied at each timestamp to generate a sequence of equal-sized graphs in linear time to tackle the problem of inconsistent matrix dimensions and significant processing times. The selection of both structural and temporal features at the same time condenses the graph information. It creates a focus on highly active regions that enhances the balance in the dataset and reduces redundant information.

Due to the separate consideration of additions and deletions, our approach supports faster learning of all types of events (additions and deletions of nodes and edges). It provides dynamic structural features for all kinds of subsequent graph learning applications.

Future research will investigate the theoretical and practical effort of DynHeatmaP together with a static GNN compared to a dynamic GNN. The performance will be examined concerning different neighborhood considerations and activity value determinations in the heatmap and pooling steps. This future work will also integrate the weighting in the local activity encoding and pooling of weighted edges into the procedure to provide an approach that can be used for broader applications. Furthermore, the effects of the sampled neighborhood proportionate to the occurrence of additions and deletions will be studied.

REFERENCES

[1] Bianchi, Filippo Maria and Grattarola, Daniele and Alippi, Cesare. 2020. Spectral Clustering with Graph Neural Networks for Graph Pooling. In *International Conference on Machine Learning*. PMLR, 874–883.

[2] Bienenstock, Elie and von der Malsburg, Christoph. 1987. A Neural Network for Invariant Pattern Recognition. *EPL (Europhysics Letters)* 4, 1 (1987), 121.

[3] Den Bout, Van and T. K. Miller. 1989. Graph Partitioning Using Annealed Neural Networks. In *International 1989 Joint Conference on Neural Networks*. IEEE, 521–528.

[4] Fathy, Ahmed and Li, Kan. 2020. TemporalGAT: Attention-Based Dynamic Graph Representation Learning. *Advances in Knowledge Discovery and Data Mining* 12084 (2020), 413.

[5] Fu, Dongqi and He, Jingrui. 2021. SDG: A Simplified and Dynamic Graph Neural Network. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2273–2277.

[6] Fernando Gama, Antonio G Marques, Geert Leus, and Alejandro Ribeiro. 2018. Convolutional Neural Network Architectures for Signals Supported on Graphs. *IEEE Transactions on Signal Processing* 67, 4 (2018), 1034–1049.

[7] Gao, Hongyang and Liu, Yi and Ji, Shuiwang. 2021. Topology-Aware Graph Pooling Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

[8] Elvin Isufi and Gabriele Mazzola. 2021. Graph-Time Convolutional Neural Networks. In *2021 IEEE Data Science and Learning Workshop (DSLW)*. IEEE, 1–6.

[9] Lee, Junhyun and Lee, Inyeop and Kang, Jaewoo. 2019. Self-Attention Graph Pooling. In *International Conference on Machine Learning*. PMLR, 3734–3743.

[10] Li, Yaguang and Yu, Rose and Shahabi, Cyrus and Liu, Yan. 2017. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. *arXiv preprint arXiv:1707.01926* (2017).

[11] Ma, Yao and Guo, Ziyi and Ren, Zhaocun and Tang, Jiliang and Yin, Dawei. 2020. Streaming Graph Neural Networks. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 719–728.

[12] Manessi, Franco and Rozza, Alessandro and Manzo, Mario. 2020. Dynamic Graph Convolutional Networks. *Pattern Recognition* 97 (2020), 107000.

[13] Alice Moallemy-Oureh. 2022. Student Research Abstract: Continuous-Time Generative Graph Neural Network for Attributed Dynamic Graphs. In *The 37th ACM/SIGAPP Symposium on Applied Computing (SAC '22), April 25–29, 2022, Virtual Event*. ACM, New York, NY, USA. https://doi.org/10.1145/3477314.3506967

[14] Pang, Yunsheng and Zhao, Yunxiang and Li, Dongsheng. 2021. Graph Pooling via Coarsened Graph Infomax. *arXiv preprint arXiv:2105.01275* (2021).

[15] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. *arXiv preprint arXiv:2006.10637* (2020).

[16] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. https://networkrepository.com

[17] Scarselli, Franco and Gori, Marco and Tsoi, Ah Chung and Hagenbuchner, Markus and Monfardini, Gabriele. 2008. The Graph Neural Network Model. *IEEE transactions on neural networks* 20, 1 (2008), 61–80.

[18] Simon Schaefer, Daniel Gehrig, and Davide Scaramuzza. 2022. AEGNN: Asynchronous Event-Based Graph Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12371–12381.

[19] Shrivastava, Yash and Dasgupta, Soura and Reddy, SM. 1990. Neural Network Solutions to a Graph Theoretic Problem. In *IEEE International Symposium on Circuits and Systems*. IEEE, 2528–2531.

[20] Amauri Souza, Diego Mesquita, Samuel Kaski, and Vikas Garg. 2022. Provably Expressive Temporal Graph Networks. *Advances in Neural Information Processing Systems* 35 (2022), 32257–32269.

[21] Taheri, Aynaz and Gimpel, Kevin and Berger-Wolf, Tanya. 2019. Learning to Represent the Evolution of Dynamic Graphs with Recurrent Models. In *Companion Proceedings of The 2019 World Wide Web Conference*. 301–307.

[22] Josephine M Thomas, Silvia Beddar-Wiesing, Alice Moallemy-Oureh, and Rüdiger Nather. 2021. A Note on the Modeling Power of Different Graph Types. *arXiv preprint arXiv:2109.10708* (2021).

[23] Trivedi, Rakshit and Dai, Hanjun and Wang, Yichen and Song, Le. 2017. Know-Evolve: Deep Temporal Reasoning for Dynamic Knowledge Graphs. In *international conference on machine learning*. PMLR, 3462–3471.

[24] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. 2021. Inductive Representation Learning in Temporal Networks via Causal Anonymous Walks. In *International Conference on Learning Representations (ICLR)*.

[25] Wang, Junshan and Song, Guojie and Wu, Yi and Wang, Liang. 2020. Streaming Graph Neural Networks via Continual Learning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1515–1524.

[26] Wang, Yue and Sun, Yongbin and Liu, Ziwei and Sarma, Sanjay E and Bronstein, Michael M and Solomon, Justin M. 2019. Dynamic Graph CNN for Learning on Point Clouds. *Acm Transactions On Graphics (tog)* 38, 5 (2019), 1–12.

[27] Ying, Rex and You, Jiaxuan and Morris, Christopher and Ren, Xiang and Hamilton, William L and Leskovec, Jure. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. *arXiv preprint arXiv:1806.08804* (2018).