A New Heterogeneous Graph Representation in a Social Media Platform: Steemit

Negar Maleki  
Muma College of Business  
University of South Florida  
negarmaleki@usf.edu

Balaji Padmanabhan  
Muma College of Business  
University of South Florida  
bp@usf.edu

Kaushik Dutta  
Muma College of Business  
University of South Florida  
duttak@usf.edu

Abstract—Recently, temporal graphs have substituted dynamic graphs as many real-world problems evolve in continuous time rather than in discrete time, and besides time almost all problems are designed in a heterogeneous format rather than a homogeneous one. However, most existing graph representations do not consider time in their components. To this end, in this paper, we present a new heterogeneous graph representation including time in every single component of the graph, i.e., nodes and edges. We also introduce four time-dependent queries to address machine learning or deep learning problems. Our findings reveal that considering the size of the enormous graphs, our time-dependent queries execute efficiently. In order to show the expressive power of time in graph representation, we construct a graph for a new social media platform (Steemit), and address a DL prediction task using graph neural networks (GNNs). Predicting the payout for a newly published post is one of the most fascinating classification problems in the Steemit setting, and we address this problem with two approaches followed by GNN models.

Index Terms—Heterogeneous Graph, Temporal Graph, GNNs, Steemit

I. INTRODUCTION

Social media play an essential role in our life that refers to collections of websites and applications designed to allow people to share content in real-time, efficiently, and quickly. Each social media platform specializes in one area of communication, engagement, or collaboration. Twitter, for example, is designed for brief written messages and sharing links, while Instagram is designed for sharing videos and photographs, and Reddit is a place where users may discuss posted content. Social media is one of the famous forms of the network, where in a very simple one, users are considered as nodes that are linked to the others via edges. However, social media platforms like those we already mention are not that simple and each has specific features that make the difference among the others. The variety of distinct aspects in networks makes them enormous, and most of the time hard to analyze in real-time and efficiently. Furthermore, considering the entire network may not be as advantageous during specific time intervals, such as presidential election seasons, when most activities are dedicated to election matters compared to other critical concerns that were the greatest concern a year before such events. Consequently, time would be a detached element from social media networks, and including time in their representation is a big step towards a real-time and efficient analysis, and extracting sub-graphs in a specific time intervals.

As another example, in LinkedIn, where users’ feeds get updated based on their connections’ activity, they might not be interested in connecting to users who are not actively engaged with them. A time period can be used to assess engagement on this platform; for example, assume a connection loses interest in another user’s material and hasn’t read, liked, or supported their stuff in a long time. As a result, the unsupported user may wish to disconnect from such users, which is not possible until we establish a time in which case any users who were not active during that period would be designated as non-active users for the unsupported user. From a marketing perspective, time also matters; for example, consider a retailer plans to launch a discount campaign and wants to prioritize its customers based on their purchases in a specific period, e.g., Black-Friday, to offer them an appropriate discount. In this context, the retailer needs to establish a time and assesses the amount of purchases each customer made in that period and then decide how much discount each customer should receive. The importance of time in deep learning algorithms also discussed in the literature, for example Long-term Short Memory (LSTM) algorithm and temporal graph models are time sensitive algorithm where the order of the states matters. To this end, this paper shows the expressive power of including time in the graph representation. To the best of our knowledge, this is the first time that time element is considered in all components of the graph representation.

The prevalence of various information in social media makes their network an interesting source of information. Graphs have so many different representations such as trade networks, communication networks, economic networks, transportation networks, biological networks, social networks, etc. Graphs, typically, show as $G = (V,E)$, consisting of nodes $V$ and edges $E$. Each node and edge may have its own set of node and edge properties. Graph features can be extracted depending on which level of the graph is the main focus. Three-level is mainly defined in a graph; node-level, edge-level, and graph-level. At the node-level, the main goal is to characterize the structure and position of the node in the network, which can be done through node degree, node centrality, clustering coefficient, and graphlets approaches. At the edge-level or link-level, the goal is to featureize or create a descriptor of the relationship between two nodes in the network, which can be done through distance-based feature,
local neighborhood overlap, and global neighborhood overlap approaches. At the graph-level, the goal is to capture features that characterize the structure of an entire graph, and to do so kernel methods are used. Extracted features, then, depending on the level of focus, will be used in various machine learning (ML) and deep learning (DL) tasks. After showing the graph representation including time, we focus on different part of sub-graphs (extracted from entire graph between \( t_1 \) and \( t_2 \)) and do different operations using temporal graph algorithms as time element is part of these algorithms.

To show the expressive power of time in graph representation, we use a social media platform, Steemit [1], in form of a case study. Steemit is a social media platform that runs on top of a blockchain called Steem, which rewards its users through a cryptocurrency, called Steem as well. Steemit allows its members to publish content in the form of posts. Posts are basically blogs that user creates and posts to the Steemit website. Users can also up-vote/ down-vote other users’ posts, comment on posts, and resteem other users’ posts, which is similar to what retweeting a post on Twitter is or sharing someone else’s post on Facebook. Additionally, users can get paid for up-voting/ down-voting and commenting on other posts besides posting a content.

II. RELATED WORK

A. Graph Neural Networks

Graph neural networks (GNNs) have many designs dependent on how they define the concept of a message and the concept of aggregation. It refers to how a single layer in a GNN is formed, built by transforming and aggregating messages from children – the bottom layer of a neural network. Graph Convolutional Network (GCN) [2] is a well-known GNNs method that, in terms of weight sharing, is comparable to Convolutional Neural Network (CNN). Fast approximation spectral-based graph convolutional network [3] is one of the key algorithms classified as GCNs and utilized for node clustering. Kipf and Welling [3] built their approach by including the graph’s adjacency matrix in the forward propagation equation. They did, however, utilize a normalized version of the adjacency matrix, which they called the “renormalization trick.”

A limitation in the GCN algorithm is that it assigns the same level of importance to each edge. However, this problem is addressed in the Veličković et al. [4] paper by expanding the aggregation function of the GCN layer by assigning attention coefficients to each edge. It means the aggregation function in the graph attention network (GAT) is similar to the one in GCN, except neighbors’ embeddings scaled by attention coefficients aggregate together. Algorithms that use low-dimensional node embeddings are used in different prediction tasks. However, most of these algorithms like DeepWalk [5] are inherently transductive; adding a new node requires retraining the model, and cannot be generalized to unseen or new nodes. GraphSAGE is an algorithm that addresses this issue and can be used in dynamic graphs, where the structure is not fixed. GraphSAGE was first introduced in Hamilton et al. [6] paper to aggregate a new node nearby to generate node embedding for it efficiently. Therefore, we use GraphSAGE in our GNN architecture as the time component makes the graph dynamic.

B. Temporal Graphs

A temporal graph was firstly introduced by Kostakos [7] where he represented a graph structure including time. The dynamic features of a dataset as well as the entities and connections that it represents can be understood using temporal graphs. Learning on temporal graphs, continues-time, is a relatively new field, while most approaches focus on discrete-time dynamic graphs represented as a series of snapshots of the graph [8]–[11]. Discrete-time approaches are unsuitable for real-world applications because they tend to preserve only a small amount of structural data and capture temporal data at a coarse level, resulting in data loss between snapshots and a lack of ability to capture fine-grained temporal dynamics. To this end, several techniques that support the continuous-time situation have just lately been suggested, and we introduce some in the following.

The Dynamic Graph Neural Network (DGNN) [12] model, used for link prediction and node classification, continuously updates node information by sequentially recording new edges (connections) data, the time window between edges, and information propagation. The continuous-time dynamic network embeddings (CTDNE) [13] algorithm learns embeddings based on the temporal random walks concept, which is used for link prediction. A temporal walk is a temporally valid sequence of edges walked in ascending order of edge timings that respects time. The system is broadly applicable, with many interchangeable components, and it can be used effectively to include temporal dependencies into current node embedding and deep network models that employ random walks. DyRep [14] is a novel modeling framework that learns richer node representation over time, which is used for link and time prediction. Information in the form of association (topological evolution) and communication (activities between nodes) events are given to the framework that updates the node representations over time. The Temporal Graph Attention (TGAT) [15] layer aggregates temporal and topological features besides learning time-feature interactions. By adding TGAT layers, the network learns node embeddings as a function of time and then can be used for node classification and link prediction tasks. Temporal Graph Network (TGN) [16] framework was developed at Twitter, while it can be applied to different problems represented as sequences of timed events. TGN consists of different elements: the “memory” that stores the state of all nodes, the “message function” that updates the memory, the “memory updater” that is used to update memory when new messages appear, and the “embedding” that computes a node’s temporal embedding by executing a graph aggregation across that node’s spatio-temporal neighbors. They used their framework on Wikipedia, Reddit and Twitter datasets and showed that their model outperforms the state-of-the-art models – i.e. GAT, GraphSAGE, CTDNE, TGAT, and DyRep – that we
have already introduced. All the above models applied on the homogeneous graph representation, where there are one type of node and one type of edge. However, most real-world graphs evolve dynamically in the context of heterogeneous graph structures, where more than one type of both node and edge are included. For example, in a recent paper [17], spatial and temporal dependencies of a heterogeneous graph were integrated to learn node representations over Heterogeneous temporal graph (HTG). They described the HTG as an ordered list of heterogeneous graph slices connected by a set of temporal connections. They addressed the same problem as we do in this paper, but with the difference that we have time in every component of our heterogeneous graph instead of having them only for slicing the graph. To this end, like [17] paper, we use time to slicing graph for train-validation-test sets. Moreover, unlike previous graph representations, we include time in every component of the graph, i.e., nodes and edges.

C. Multiplex Network

Multiplexity refers to multifaceted connections between two persons [18]: therefore, social networks are a good example of multiplex networks. A group of networks containing different types of relations is considered a multiplex network where each kind of relationship creates a layer of the network. For example, in a social network such as Reddit, users often have different interactions like visiting the main subreddit page, voting from a subreddit page, posting a comment/link to a subreddit, etc. Each of these interactions creates a layer of the network among all users, and when all interactions are considered a united one, we have a massive multiplex network.

Most existing research is primarily concerned with homogeneous or single-layered networks. Real-world networks, on the other hand, are significantly more sophisticated, as cross-domain interactions across various networks are well-observed, resulting in a sort of multi-layered network [19]. A Recent paper studied how to perform network embedding for nodes on multiplex networks by providing a unified optimization framework, called MANE [19]. Their framework is considered as a heterogeneous information network, since different node types are placed in different layers. However, having different layers with different node types adds to the complexity of multiplex networks. Thus, researchers focus on different types of layers with a single node type and suggest a scalable multiplex network embedding model, called MNE [20]. However, this framework aggregates information over all layers even the irrelevant ones, so it is not computationally efficient. To address this drawback, the DEEPLEX [21] framework was proposed which considers sampling k-nearest layers with the most similar embeddings. However, although DEEPLEX sacrifices all other embeddings’ information for the sake of time complexity, it is still suffered from the computational complexity of learning k-nearest layers. In a most recent paper [22], a framework is suggested to be more efficient in terms of computational complexity, so they addressed the MNE problem by selecting information from relevant layers, and also addressed DEEPLEX drawback by adaptively learning a sampling distribution over the relevant layers. The idea of having information from relevant layers inspired us to extract sub-graphs out of the larger heterogeneous graph using time component queries. Although extracting sub-graph decreased embedding and topological information, our findings show that we still get at least the same or better results than including all information in the payout classification problem, besides sub-graph time efficiency.

III. Graph Representation

A graph is a generic mathematical language for describing complex networks. Graphs were represented as $G = (V, E)$, with nodes $V$ and edges $E$. In every context, nodes and edges translate to the relevant domain of its application, and so does our graph representation. In this paper, the graph is constituted of actors and actionable items $<\text{actor}_1, \text{action} \rightarrow \text{actor}_2>$, where each may have its own set of attributes. Attributes can be divided into two main categories, static and dynamic attributes. Static attributes mainly define the state of an actor or actionable item. However, dynamic attributes as their name states can be changed over time, so they are time-dependent attributes. Networks, in reality, are not static, which means actors take actions and modify the networks in various ways. Three main operations take place in networks: items addition, items deletion, and items modification. In items addition operation, new actors or actionable items appear and create new connections to the network, which extend the network. Items modification operation gives actors or actionable items an opportunity to update or modify the existing actors or actionable items. Lastly, the items deletion operation removes existing actors or actionable items where all the dependent children will be disappeared as their parents are removed.

Graphs are searchable and able us to query and extract intuitive information, which later can be used in different ML tasks. Relational algebra operations like selection, projection, and join are common among different databases management systems (DBMS), including network database systems. Selection ($\sigma$) operation as one of the unary relational operations is used for selecting a subset of tuples that satisfy the selection condition. Projection ($\pi$) operation as another operation in the unary relational operations is used for keeping specific attributes from a relation and ignoring the rest attributes. Lastly, join ($\bowtie$) operation as one of the binary relational operations is used for joining variously related tuples from different relations. As an example, we followed the Facebook representation presented in [23] to illustrate Steemit representation in form of a graph, Fig.1 but there are some nuances in Steemit compared to Facebook. In Facebook representation, an actionable item is called tagged/ tagged-at, while in Steemit, such actionable item is not the case. Instead, time and some other features relevant to the incentive mechanism of Steemit exist that explain the differences between Steemit and other social platforms.
TABLE I

| Item        | Item Type | Description                                      |
|-------------|-----------|--------------------------------------------------|
| authored    | Actionable| A person is the author of a post.                |
| reply       | Actionable| A post is replied by a comment.                  |
| vote        | Actionable| A post is voted by a person.                     |
| follow      | Actionable| A person is followed by a person.                |
| user        | Actor     | Take actionable items on another actor.          |
| post        | Actor     | Take actionable items on another actor.          |
| comment     | Actor     | Take actionable items on another actor.          |

A. Application Programming Interface

SteemOps is a dataset presented in [24], which contains ten key types of Steemit operations organized into three sub-datasets: (1) the social-network operation dataset (SOD), (2) the witness-election operation dataset (WOD), and (3) the value-transfer operation dataset (VOD). The data was collected from 2016/03/24 16:05:00 to 2019/12/01 00:00:00. The main sub-dataset we use in this paper is SOD consisting of three operational keys - comment, vote, and custom-json. The comment operation consists of five fields, see Table I. According to [24], a new post is indicated when both parent-author and parent-permlink fields are empty. When these two fields are not empty, it represents a comment to a post/comment. Each post in Steemit remains active for seven days, so each time the author made any changes to his post, the post will be recorded as a new post to the dataset. Moreover, each post’s permlink is unique, so considering all of these, the dataset consists of 17805355 new posts.

TABLE II

| Field Name       | Type      | Description                                      |
|------------------|-----------|--------------------------------------------------|
| block_no         | Integer   | the block recording this operation               |
| parent_author    | String    | the author that comment is being submitted to     |
| parent_permlink  | String    | specific post that comment is being submitted to  |
| author           | String    | author of the post/comment being submitted (account name) |
| permlink         | String    | unique string identifier for the post, linked to the author of the post |

B. Heterogeneous Graph Construction

We denote Steemit as a directed graph \( <V,E> \), where \( V \) represents actors (nodes), and \( E \) represents actionable items (edges). In Steemit, an actor can be of different types, such as user and content (post or comment). The interactions between actors are captured by actionable items which also can be of different types. Table I \( <actor_1 - actionivable_item → actor_2> \). For example, a user can follow another person, which we denote as \( <user_1 - follow → user_2> \). Similarly, a user can make a post, which we denote as \( <user_1 - Author → post_1> \). or a user can vote a post or a comment, which we denote as \( <user_1 - vote → post_1> \), \( <user_1 - vote → comment_1> \). As mentioned before, actors and actionable items have static and/or dynamic attributes. For example, user object type (otype) or actor has “name”, “bot”, and “joined” date as static attributes and number of followers and following as dynamic attribute that can be calculated using graph queries. Post object type (otype) or actor has many static attributes illustrated in Fig I except “title”, “body”, and “last_update” which are considered as dynamic attributes where their changes will be stored in form of a list or JSON depending on the attribute characteristics, and also a total number of post’s votes can be calculated using graph queries (sum of incoming votes (actionable item)). Please note that dynamic attributes calculated through queries do not need to be shown in the graph representation. In Steemit, actionable items have time as their attributes that make the graph representation in form of a Dynamic Graph. We constructed Steemit graph representation in a form of directed graph (DiGraph) using NetworkX Python library. The constructed graph is then converted into the PyTorch Geometric (PyG) graph in order to be used for GNN models. By doing so, we create a dynamic graph from which needed sub-graphs can be extracted using queries, and the derived sub-graphs may subsequently be utilized to solve various DL tasks.

IV. Graph Queries

As mentioned, three main operations in real networks take place, and Steemit is not an exception. Items addition in the context of Steemit is the simplest one where actors take new actions and extend the network, like a user creates a post, writes a comment, follows another user, etc. Item modification operation happens when, for example, a user decides to update existing content. These modifications are not replaced with the old version, instead, they will be stored in form of a list or JSON. Lastly, items deletion is the trickiest one where, for example, a user decides to delete content, so all the dependent children – like comments and comments of the comments as actors; and votes, replies, authored, and follow as actionable items – will be subsequently disappeared as their parents are removed.

The entire graph is not always the point of interest in different tasks. As we mentioned before, an interesting part of the Steemit representation that makes it unique is time in its actions and actionable items. To this end, we may like to...
take a sub-graph out of the entire graph, which represents all activities between a specific time interval. The application of such representation could be different; for example, in presidential election seasons, people’s activities on social media become attractive. Therefore, analysis of such activities gives interesting information required to be gained from specific time intervals. The key point in this representation is that we only include actions and actionable items in the specified time intervals. It means that any actions or actionable items connected to nodes but did not occur during the given time interval will be removed from the sub-graph. Algorithm 1 shows the pseudocode of sub-graph query between $t_1$ and $t_2$, which is the first query part for all the following queries. In the following queries, the output of the Algorithm 1 will be given as an input. The time complexity of the Algorithm 1 is $O(n + m)$ and if $n \geq m$, the time complexity will equal to $O(n)$, or vice versa.

Consider Fig 2 which shows a heterogeneous network including users, posts, and comments as different types of the nodes and different types of edges connecting nodes (nodes and edges types are similar to what we explained in the previous section (Table I)). At the top of each edge, there is a time ($t$) that shows when the actionable items take place, and the numbers at their subscriptions show the order of those actionable items, so $t_2 > t_1$. Imagine we are interested in time interval between $t_1$ and $t_2$, so when we apply the Algorithm 1 to this network, all the nodes and edges that are not placed in the given time interval will be deleted and the final sub-graph will include exactly actions and actionable items between $t_1$ and $t_2$.

Since having a sub-graph between a specific time interval is the main focus of this paper to show the expressive power of time in networks, the following queries’ input will be a sub-graph extracted for a given time interval. Three different queries that we are interested in pursuing in this paper are as follows:

### A. Post’s Category Query

Post’s category is one of the post’s attributes in Steemit. However, if the first tag was among the Steemit popular tags, it remains as it is; otherwise, the Steemit

---

Algorithm 1: Query Sub-Graph between $t_1$ and $t_2$

1. **Input:** $G(V,E)$: graph with $n$ nodes and $m$ edges
2. **Input:** $t_1, t_2$: timestamp
3. **Output:** $G_{post} = t_1, t_2 (A, B)$: Subgraph between $t_1$ and $t_2$
4. **define function** get_key(val):
5.   set key_index_node to [ ]
6.   set key_index_edge to [ ]
7.   for key, value in networkx.get_attributes(G, 'otype').items() do
8.     if val equals value then
9.       add key to key_index_node
10.      add key to key_index_edge
11.   end if
12.   end for
13. return key_index_node, key_index_edge
14. call set post, comment, user to get_key(('[post', 'comment', 'user'])
15. call set vote, authored, reply to get_key(('[vote', 'authored', 'reply'])
16. set subG_post, subG_comment to [ ]
17. for i in post (or comment) do
18.   if $t_1 \leq G.nodes()[i][\text{created}] \leq t_2$ then
19.     add i to subG_post (or subG_comment)
20.   end if
21. end for
22. set subG_vote, subG_authored, subG_reply to [ ]
23. for i in vote (or authored or reply) do
24.   if $t_1 \leq G.edges()[i][\text{time}] \leq t_2$ then
25.     add i to subG_vote (or subG_authored or subG_reply)
26.   end if
27. end for
28. set G_node to G.subgraph(subG_post + subG_comment + user)
29. set $G_{post} = t_1, t_2$ to $G_{node}.edge_subgraph(subG\_vote + subG\_authored + subG\_reply)
platform puts different words as category [25]. This attribute can be used for selecting specific field in Steemit posts, for example, if we are interested in health-related posts, we may look for categories like healthcare, medical, medicine, etc. Following the selection of an specific field, we may interesting in applying topic modeling, which is a trade topic in machine learning applications. **Algorithm 2** is the query following the sub-graph between \( t_1 \) and \( t_2 \) to selecting posts with a specific category. Its time complexity equals to \( O(n^2) \) where \( n \) is the number of nodes in sub-graph between \( t_1 \) and \( t_2 \).

Consider Fig 3, which shows a heterogeneous network with nodes representing users, posts, and comments. In this query, the goal is to have specific categories between a given time interval. Posts' categories are \( A, B, C, D \) that are shown below each post in Fig 3 and we are interested in \( CandD \) categories between \( t_1 \) and \( t_6 \). After applying **Algorithm 2** to this graph, any nodes and edges that are not placed in the given time interval and any posts that do not meet mentioned categories will be eliminated. Therefore, the final sub-graph will include exactly actions and actionable items between \( t_1 \) and \( t_6 \), as well as posts with \( C \) and \( D \) categories.

**B. Vote and Comment Velocity Query**

Different factors affect the payout amount in Steemit [24]; two of these factors that we already have access to are votes and comments velocity, which both of them are obtained by counting the number of vote and reply edges as they are dynamic attributes. Based on these factors, we may want to predict the payout for upcoming posts, as posts’ payout will be determined after the 7-day time window after the post is publicly available on Steemit, using temporal graph algorithm. **Algorithm 3** is the query following the sub-graph between \( t_1 \) and \( t_2 \) query to sort posts based on their vote and comment velocity. Its time complexity equals to \( O(n \log(n) + nm) \), where \( n \) is the number of nodes and \( m \) is the number of edges in sub-graph between \( t_1 \) and \( t_2 \). If \( (nm) \) is greater than \( O(n \log(n)) \), the time complexity equals to \( O(nm) \), or vice versa.

Consider we are interested in sorting posts based on their votes and comments velocity between "Thursday, August 1, 2019 4:00:00 AM GMT" and "Saturday, August 10, 2019 4:00:00 AM" (10-day). In the entire heterogeneous graph, we have 165180 nodes and 1405939 edges, which between the given time interval, the number of nodes and edges reduced into 85062 and 907437, respectively. Table III includes 10

**Algorithm 2** Query SubGraph between \( t_1 \) and \( t_2 \) with a specified category

1. **Input:** \( G_{post\_t_1\_t_2}(A, B) \) : Subgraph between \( t_1 \) and \( t_2 \)
2. **Input:** category: string
3. **Output:** \( G_{post\_t_1\_t_2\_category}(C, D) \) : Subgraph between \( t_1 \) and \( t_2 \) with an specified category
4. **call set post** to get_key('post')
5. **set subG_{t_1\_t_2\_post} to []**
6. for \( i \) in post:
7. if \( G_{post\_t_1\_t_2\_nodes}(\ket{category}) \) equals category then
8. add \( i \) to subG_{t_1\_t_2\_post}
9. end if
10. end for
11. **set related\_node to []**
12. **set related\_cm to []**
13. for \( i \) in subG_{t_1\_t_2\_post}
14. for \( j \) in range(len(subG\_vote))
15. if \( i \) equals subG\_vote[j][1] then
16. add subG\_vote[j][0] related\_node
17. for \( j \) in range(len(subG\_authored))
18. if \( i \) equals subG\_authored[j][1] then
19. add subG\_authored[j][0] related\_node
20. for \( j \) in range(len(subG\_reply))
21. if \( i \) equals subG\_reply[j][1] then
22. add subG\_reply[j][1] related\_node
23. add subG\_reply[j][1] related\_cm
24. end if
25. end if
26. end if
27. end for
28. end if
29. end if
30. end for
31. for \( i \) in related\_cm
32. for \( j \) in range(len(subG\_vote))
33. if \( i \) equals subG\_vote[j][1] then
34. add subG\_vote[j][0] related\_node
35. for \( j \) in range(len(subG\_authored))
36. if \( i \) equals subG\_authored[j][1] then
37. add subG\_authored[j][0] related\_node
38. end if
39. end if
40. end if
41. end for
42. end for
43. **set category\_node to drop.duplicates(related\_node)**
44. **set G_{post\_t_1\_t_2\_category} to G_{post\_t_1\_t_2\_subgraph} (category\_node + subG_{t_1\_t_2\_post})**
Algorithm 3: Query to Sort posts based on their vote and comment velocity

1: **Input:** $G_{post,t_1,t_2}(A, B):$ Subgraph between $t_1$ and $t_2$
2: **Output:** comments_sorted, votes_sorted: Lists of posts in descending order based on their comment and vote velocity
3: set vote_count to []
4: set comment_count to []
5: for $i$ in $G_{post,t_1,t_2}.nodes()$ do
6: set vote to 0
7: set comment to 0
8: if $G_{post,t_1,t_2}.nodes()[i]['otype']$ equals ‘post’ then
9: for $j$ in $G_{post,t_1,t_2}.out_edges(i)$ do
10: if $G_{post,t_1,t_2}.edges()[i][j][1]['otype']$ equals ‘reply’ then
11: comment += 1
12: end if
13: end for
14: add [i, comment] to comment_count
15: for $j$ in $G_{post,t_1,t_2}.in_edges(i)$ do
16: if $G_{post,t_1,t_2}.edges()[j][0][1]['otype']$ equals ‘vote’ then
17: vote += 1
18: end if
19: end for
20: add [i, vote+1] to vote_count
21: end if
22: end for
23: set comments_sorted to sorted(comment_count, key=lambda x: x[1], reverse=TRUE)
24: set votes_sorted to sorted(vote_count, key=lambda x: x[1], reverse=TRUE)

Algorithm 4: Query to Sort users who are actively engaged

1: **Input:** $G_{post,t_1,t_2}(A, B):$ Subgraph between $t_1$ and $t_2$
2: **Output:** user_activity_sorted: List of users in descending order who are actively engaged
3: set user_activity to []
4: for $i$ in range(len(user)) do
5: if user[i] in $G_{post,t_1,t_2}.nodes()$ then
6: add [user[i], len($G_{post,t_1,t_2}.out_edges(user[i])$)] to user_activity
7: end if
8: end for
9: set user_activity_sorted to sorted(user_activity, key=lambda x: x[1], reverse=TRUE)

**C. Actively Engaged Query**

Social media platforms quickly became the most used online services, through which billions of users interact intensively every day. This makes social media a significant resource for advertising, marketing, or politics, collecting information, and launching campaigns. A challenging problem is identifying actively engaged users on Steemit, which may be described as users with a high number of votes, comments, and the number of posts they have published. **Algorithm 4** is the query following the sub-graph between $t_1$ and $t_2$ query to sort users who are actively engaged. Its time complexity equals to $O(n)$, where $n$ is the number of nodes in sub-graph between $t_1$ and $t_2$.

**Table III**

| Post Title                                      | Payout    | Num. of Comments | Num. of Votes |
|------------------------------------------------|-----------|------------------|---------------|
| Announcing the eSteem Token: ESTM              | 142.047   | 73               | 555           |
| IT IS A FLOWER AND BENEFICIAL INSECT WEEKEND!  | 3.477     | 66               | 225           |
| “Guess Who Is In This Photo” Contest (Sport’s Figure) | 0.103     | 62               | 116           |
| BLOODY IMPORTANT QUESTION should we consider STEEM-ENGINE tokens a security or utility! | 52.223    | 60               | 325           |
| “Guess Who Is In This Photo” Contest (Sport’s Figure) | 0.465     | 57               | 131           |
| Giving Away 500 Steem To One Random Person     | 3.926     | 55               | 136           |
| “Guess Who Is In This Photo” Contest (Sport’s Figure) | 0.903     | 54               | 77            |
| Actifi Major Announcement: Move AFIT To Steem-Engine. AFITX Our New Exclusive Token Announced - Detailed Dive. Airdrop Announce. Daily Updates | 107.177   | 53               | 673           |
| Pony Auction on Market Friday                   | 5.105     | 52               | 159           |
| “Guess Who Is In This Photo” Contest (Sport’s Figure) | 0.035     | 51               | 76            |

Consider we are interested in getting know those users who are actively engaged in terms of number of votes they cast and number of posts/comments they wrote between “Thursday, August 1, 2019 4:00:00 AM GMT” and “Saturday, August 10, 2019 4:00:00 AM” (10-day). **Table IV** includes 10 top users.
with their number of activities in casting votes and writing posts or comments. Total execution time of this query took 14,1105 seconds. (Please note that because users’ username are anonymous and publicly available, we include them in the Table IV)

TABLE IV
TOP 10 ACTIVE ENGAGED USERS BETWEEN "THURSDAY, AUGUST 1, 2019 4:00:00 AM GMT" AND "SATURDAY, AUGUST 10, 2019 4:00:00 AM"

| Username          | Num. of Cast Votes | Num. of Written Posts/Comments |
|-------------------|--------------------|-------------------------------|
| 'laisssez-faire'  | 5352               | 0                             |
| 'steemitboard'    | 1235               | 422                           |
| 'accelerator'     | 2547               | 5                             |
| anomaly           | 2459               | 0                             |
| 'imisstheoldkanye'| 2444               | 0                             |
| 'actift'          | 994                | 192                           |
| 'map10k'          | 1730               | 2                             |
| 'fyrstikken'      | 1739               | 0                             |
| 'hdu'             | 1513               | 0                             |
| 'steem-plus'      | 686                | 134                           |

D. Queries Time Complexity

As we discussed in the previous sub-sections, these queries have a low execution time compared to the size of the graph. Fig 4 illustrates the time complexity of each query for different number of nodes (i.e., number of posts). As we have a heterogeneous graph, we focused on the number of posts in each run and implemented four queries on the extracted sub-graph for that number of posts. We considered t1 and t2 the same during implementing all the queries. For the second query, we picked ”Steemit” to retrieve posts with the ”Steemit” category. As shown in the Fig 4 we can see some queries with similar time complexity have different execution times; this is because of extracted sub-graph with a number of posts within. It means that although we specified the number of posts and extracted a sub-graph out of the entire graph, it has a different number of nodes and edges at each time because of the heterogeneous graph structure. Moreover, we extracted another sub-graph out of the first sub-graph to have a sub-graph between t1 and t2 in order to use it for subsequent queries. Therefore, we have different nodes and edges in each step, which cause different execution times, while the time complexity between queries might be close to each other.

V. ILLUSTRATIVE PREDICTION TASKS

In the previous section, we illustrated different queries that each of which pursuing one or more ML tasks, like regression, classification, and clustering. Predicting the payout for a newly published post is one of the most fascinating classification problems in the Steemit setting. According to the Steemit documentation, the payout is determined by a variety of factors, including the number of up-votes and down-votes, the number of comments, which user with what reputation score voted or replied to the post, how much new STEEM is created each week and distributed to users as rewards over a rolling 7-day period, and so on. As a result, because there are so many dynamic elements that impact the ultimate payout, no exact formula defines what the payout of a new post will be. Moreover, as shown in Table IIII posts with high comments and votes velocity did not necessarily get paid highly that acknowledged the earlier statement of lack of exact formula for calculating post’s payout. Although predicting payout seems to be a regression task, the amount of unpublished data - such as the amount of payout each post has gained during the 7-day time window at each second (rather than the final payout that we have in the dataset) - limited us to defining this problem as a classification task. Therefore, in this problem, we have two approaches to classify the payout amount into low, medium, and high categories. Fig 5 shows only the distribution of posts’ payout with nonzero payout.

A. Model 1: Steemit Features

In this approach, we include all the published features in the dataset, Table V for each type of node and edge, along with topological information of the graph, and then predict the payout of a new post. As we have time in each component of the graph, we ensure that all the connections to each post node are in a 7day time window using queries since all payouts...
occur exactly 7 days after a post is made. Moreover, as it is a temporal graph, we sort all the post nodes based on their creation time and divide the entire graph into three sub-graphs for train, validation, and test with 80%, 10%, and 10% of heterogeneous graph data, respectively, which would be used as input into the GNN model, similar to [17] paper.

| Features         | Node Type     | Description                        |
|------------------|---------------|------------------------------------|
| node_type        | post, user, comment | Type of node                       |
| net_rshares      | post, comment  | Sum of positive and negative rewards|
| abs_rshares      | post, comment  | Total absolute weight of votes      |
| vote_rshares     | post, comment  | Total positive rshares from all votes|
| author_rewards   | post, comment  | Tracks the author payout            |
| author_reputation| post, comment  | Author’s reputation                 |
| payout           | comment        | Total payout                        |

As we mentioned in section 3.2, we build the heterogeneous graph using NetworkX Python library, and for implementing GNN models we need to convert it into PyTorch Geometric (PyG) library. However, so far, there is not a built-in function in the PyTorch library to convert the heterogeneous graph into PyG format. Therefore, this paper gives interested readers an insight to convert their heterogeneous NetworkX graph into a heterogeneous PyG graph.

In the heterogeneous graph data, as different types of node and edge features exist, standard Message Passing (MP) cannot easily applied and processed by the same function. As shown in the Fig 6, the method starts with an GAT GNN model and repeats the message functions to work on each edge type independently.

B. Model 2: Steemit Posts’ Content

In the second approach, we take the embedding vectors out of the body of the posts using a pre-trained Google Universal Sentence Encoder model. The Universal Sentence Encoder encodes text into high dimensional vectors (512 embeddings) that can be used for text classification, clustering, semantic similarity, etc. The pre-trained model is publicly available in Tensorflow-hub[^26]. It is available in two variants, one trained using a Transformer encoder and the other with a Deep Averaging Network (DAN). The accuracy and computational resource requirements of the two are trade-offs. While the one with the Transformer encoder is more accurate, it is also more computationally expensive. The one with DNA encoding is less costly computationally and has somewhat lower accuracy. To this end, we used the one with transformer to encodes the posts’ content into 512 dimensional vectors, and gave them to the GNN model to predict new posts payout.

Again, as we have the heterogeneous graph data, different types of node and edge features exist, so the standard MP cannot easily applied and processed by the same function. As shown in the Fig 7 the method starts with an SAGE GNN model and repeats the message functions to work on each edge type independently.

Fig. 6. Model 1 for classification task

Fig. 7. Model 2 for classification task

REFERENCES

[1] U. W. Chohan, “The Concept and Criticisms of Steemit,” (February 20, 2021), CBRI Working Papers: Notes on the 21st Century, Available at SSRN: 10.2139/ssrn.3129410.

[2] M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on graphs with fast localized spectral filtering.” In Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS), 2017.

[^26]: https://tfhub.dev/google/universal-sentence-encoder/4
[3] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," arXiv preprint [arXiv:1609.02907] 2017.

[4] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," arXiv preprint [arXiv:1710.10903] 2017 Oct 30.

[5] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: online learning of social representations," In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '14), Association for Computing Machinery, New York, NY, USA, 2014, 701–710. 10.1145/2623330.2623732.

[6] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," Advances in neural information processing systems, 2018, 30.

[7] V. Kostakos, "Temporal graphs," Physica A: Statistical Mechanics and its Applications, 2009, 15, 388(6), 1007-23.

[8] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. Schardl, and C. Leiserson, "EvolveGCN: Evolving graph convolutional networks for dynamic graphs," InProceedings of the AAAI Conference on Artificial Intelligence 2020, Vol. 34, No. 04, pp. 5363-5370.

[9] L. Zhu, D. Guo, J. Yin, G. Ver Steeg, and A. Galstyan, "Scalable temporal latent space inference for link prediction in dynamic social networks," IEEE Transactions on Knowledge and Data Engineering, 2016, 28(10), pp. 2765-77.

[10] Y. Zuo, G. Liu, H. Lin, J. Guo, X. Hu, and J. Wu, "Embedding temporal network via neighborhood formation," InProceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, 2018, pp. 2857-2866.

[11] P. Goyal, N. Kanra, X. He, and Y. Liu, "Dyngem: Deep embedding method for dynamic graphs," arXiv preprint [arXiv:1805.11273] 2018.

[12] Y. Ma, Z. Guo, Z. Ren, J. Tang, and D. Yin, "Streaming graph neural networks," InProceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2020, pp. 719-728.

[13] GH. Nguyen, JB. Lee, RA. Rossi, NK. Ahmed, E. Koh, and S. Kim, "Dynamic network embeddings: From random walks to temporal random walks," In2018 IEEE International Conference on Big Data (Big Data), 2018, pp. 1085-1092.

[14] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "Representation learning over dynamic graphs," arXiv preprint [arXiv:1803.04051] 2018.

[15] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achanta, "Inductive representation learning on temporal graphs," arXiv preprint [arXiv:2002.07962] 2020.

[16] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," arXiv preprint [arXiv:2006.10637] 2020.

[17] Y. Fan, M. Ju, C. Zhang, and Y. Ye, "Heterogeneous Temporal Graph Neural Network," InProceedings of the 2022 SIAM International Conference on Data Mining (SDM), 2022, pp. 657-665; Society for Industrial and Applied Mathematics.

[18] L. M. Verbrugge, "Multiplexity in adult friendships. Social Forces," 1979, 57(4), pp. 1286-309.

[19] J. Li, C. Chen, H. Tong, and H. Liu, "Multi-layered network embedding," InProceedings of the 2018 SIAM International Conference on Data Mining, 2018, pp. 684-692; Society for Industrial and Applied Mathematics.

[20] H. Zhang, L. Qiu, L. Yi, and Yi Song. "Scalable multiplex network embedding," In IJCAI, vol. 18, 2018, pp. 3082-3088.

[21] V. K. Potluru, R. E. Tillman, P. Reddy, and M. Veloso, "Deeplex: A gun for link prediction in multiplex networks," InSIAM Workshop on Network Science, 2020.

[22] C. Baykal, V. K. Potluru, S. Shah, and M. M. Veloso, "Bandit Sampling for Multiplex Networks," arXiv preprint [arXiv:2202.03621] 2022.

[23] N. Bronson, Z. Arnison, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. Li, and M. Marchukov, "TAO:Facebook's Distributed Data Store for the Social Graph," In2013 USENIX Annual Technical Conference, 2013, pp. 49-60.

[24] C. Li, B. Palanisamy, R. Xu, J. Xu, and J. Wang, "SteemOps: Extracting and Analyzing Key Operations in Steemit Blockchain-based Social Media Platform," InProceedings of the Eleventh ACM Conference on Data and Application Security and Privacy, 2021, pp. 113-118.