IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 9, 2022, accepted April 16, 2022, date of publication May 18, 2022, date of current version June 21, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3175981

A Robust Comparative Analysis of Graph Neural Networks on Dynamic Link Prediction

JOAKIM SKARDING, MATTHEW HELLMICH, BOGDAN GABRYS, (Senior Member, IEEE), AND KATARZYNA MUSIAL

Complex Adaptive Systems Laboratory, Data Science Institute, University of Technology Sydney, Sydney, NSW 2007, Australia

Corresponding author: Joakim Skarding (joakim.skarding@student.uts.edu.au)

This work was supported by the Australian Research Council through the "Dynamics and Control of Complex Social Networks" under Grant DP190101087.

ABSTRACT Graph neural networks (GNNs) are rapidly becoming the dominant way to learn on graph-structured data. Link prediction is a near-universal benchmark for new GNN models. Many advanced models such as Dynamic graph neural networks (DGNNs) specifically target dynamic graphs. However, these models, particularly DGNNs, are rarely compared to each other or existing heuristics. Different works evaluate their models in different ways, thus one cannot compare evaluation metrics and their results directly. Motivated by this, we perform a comprehensive comparison study. We compare link prediction heuristics, GNNs, discrete DGNNs, and continuous DGNNs on the dynamic link prediction task. In total we summarize the results of over 3200 experimental runs ($\approx$ 1.5 years of computation time). We find that simple link prediction heuristics perform better than GNNs and DGNNs, different sliding window sizes greatly affect performance, and of all examined graph neural networks, that DGNNs consistently outperform static GNNs. This work is a continuation of our previous work, a foundation of dynamic networks and theoretical review of DGNNs. In combination with our survey, we provide both a theoretical and empirical comparison of DGNNs.

INDEX TERMS Dynamic network models, graph neural networks, link prediction, temporal networks.

I. INTRODUCTION

Recently, there has been a drive to enable fair comparisons and benchmarks of GNNs [1]–[3]. The reason for this was in part due to a lack of common practice in the validation and testing of GNNs as well as concerns around replicability and reproducibility. Reproducibility is a challenge for a wider field of machine learning [4], [5] and even science in general [6] making it difficult to identify actual scientific advances.

In the space of dynamic graph neural networks (DGNNs), these problems are further exacerbated by a set of challenges that include (i) the dynamic and heterogeneous nature of the data, (ii) the lack of common terminology [7], (iii) the lack of established strong baselines (most works do not compare performance of a proposed DGNN to other DGNNs), (iv) the divide between discrete and continuous DGNNs and (v) the wide array of experimental design choices. Among the choices are: (i) how to represent the dynamic network (e.g. snapshot, time-windows, continuous, time-to-live of edges, etc.), (ii) which node features to include, (iii) how to split the data into train-validation-test sets, (iv) which metrics to use to evaluate the results, (v) how to use negative sampling rate in reported metrics and (vi) how to choose/optimize neural network parameters (e.g. learning rate, early stopping criterion, embedding space dimensions, etc.). All of this means that comparing the performance of methods through meta-analysis, i.e. by reading research papers, is not possible unless they clearly state all their design choices and those design choices are identical between papers. In our previous work [8], we showed that DGNNs are a promising avenue in modeling network dynamics. This is mainly due to their ability to encode both spatial patterns through GNNs and temporal patterns through time-series components (e.g. recurrent neural networks (RNN) or self-attention). However, the so far proposed DGNNs have been tested on few datasets and are rarely compared to other DGNNs. Different studies compare the methods on different datasets as there is no consensus when it comes to which datasets to use in DGNN benchmarking. In light of these problems, a purely

The associate editor coordinating the review of this manuscript and approving it for publication was Qiang Lai.

theoretical comparison which we provided in our survey is not sufficient [8]. We, therefore, extend our previous work by performing a comprehensive comparison of GNN methods on the dynamic link prediction task. We benchmark each GNN and DGNN using the same experiment design, including the same strategy for optimizing hyperparameters (i.e. grid search).

Previous works in the DGNN space focus on presenting a new architecture. These models are either discrete or continuous DGNNs. However, these types of models are not compared to each other. Discrete models operate on discrete network representations while continuous models operate on continuous representations. A comparison between these kinds of models is not possible unless they are evaluated on the same network representation. To enable this comparison, we introduce the DIScrete and COntinuous dynamic link prediction framework (DISCO). The framework represents the datasets as both discrete and continuous while evaluating the predictions identically. This is achieved by transferring the continuous DGNNs to the discrete domain. In short, this is done by training them separately on the continuous representation, then training a decoder using the continuous node embeddings on the discrete representation.

Using our new framework we compare link prediction heuristics, static GNNs, discrete DGNNs, and continuous DGNNs. This is therefore not just a comparison of different models, but a comparison of different kinds of models. Our aim is to give an indication of which GNNs are best capable of encoding dynamic network topology and indeed if they are better than well-established heuristics commonly used in the network science community for link prediction. To the best of our knowledge, this is the first comparison of discrete and continuous DGNNs.

Another important question is whether DGNNs are better than traditional GNNs at encoding dynamic graph structure. There are multiple ways that dynamic networks can be represented as static networks, thus allowing GNNs to encode dynamic networks. Previous works compare DGNNs to GNNs and their results indicate that DGNNs do indeed perform better than GNNs [9]–[12]. However, these works use only one of the many ways of converting dynamic networks to static ones. Without exploring these options it is still uncertain whether DGNNs tend to outperform GNNs on dynamic network encoding. To gain more insight, we use three different ways of aggregating dynamic networks to static networks,

Our primary contribution is a fair comparison of graph neural networks and link prediction heuristics on dynamic link prediction task. To this end, we introduce the DISCO framework that can train different types of GNNs; static GNNs, discrete DGNNs, and continuous DGNNs. Empirically we find that 1) simple heuristics outperform DGNNs in terms of mAP on almost all datasets. 2) Heuristics and discrete models perform better in terms of mAP while continuous models score better in terms of AUC. Indicating that heuristics and discrete models are better at ranking the most highly ranked links, while continuous models rank subsequent links better. 3) Static GNNs are outperformed by DGNNs. 4) Sliding training windows greatly affects the model's performance, the models tend to perform better with a sliding training window size of 5 or 10. 5) The compared continuous DGNNs' performance is greatly affected by informative edge features.

To enable reproduction of our results and facilitate future work we publicly release our code and configuration files.[1]

The relevant background is covered in Section II-A. Section III covers the framework and how the experiments are performed. The results are presented and discussed in Section IV

II. BACKGROUND

This section covers the background and previous work of dynamic network data, methods that we compare, namely, graph neural networks and link prediction heuristics, the dynamic link prediction task, and previous related comparisons and benchmarks.

A. DYNAMIC NETWORKS

Dynamic network terminology has yet to converge. Networks, where edges and nodes may appear or disappear over time, go by many names in the literature [7]. Here we will refer to these networks as dynamic networks. There are several kinds of dynamic networks and these may also go by different names. In this work, we adopt the dynamic network cube terminology for dynamic networks, a conceptual framework that groups dynamic networks along three dimensions and enables more precise terminology [8].

Definition 1 (Dynamic Network): a dynamic network is a graph $G = (V, E)$ where: $V = \{(v, t_s, t_e)\}$, with v being a vertex of the graph and t_s, t_e are respectively the start and end timestamps for the existence of the vertex (with $t_s \leq t_e$). $E = \{(u, v, t_s, t_e)\}$, with $u, v \in V$ and t_s, t_e are respectively the start and end timestamps for the existence of the edge (with $t_s \leq t_e$).

Temporal granularity refers to how coarse or fine-grained a network representation is [8]. In order of increasingly fine-grained representation, we have static, discrete, and continuous networks (Figure 1. Static networks are networks with no information about time.

Discrete representations are ordered sets of static graphs, examples of such representations include snapshots. A discrete network representation is an ordered set of graphs,

$$DG = \{G^1, G^2, \ldots, G^T\}, \tag{1}$$

where T is the number of snapshots/time-windows.

Continuous network representations model exact temporal information. They represent the dynamic network as one graph with time stamps on the nodes and/or edges. Examples of such representations include interval graphs [7], [13] and graph streams [14]. Since this work uses interaction networks where links have no duration, we use a contact sequence [13]

[1] Code available at https://github.com/xkcd1838/bench-DGNN

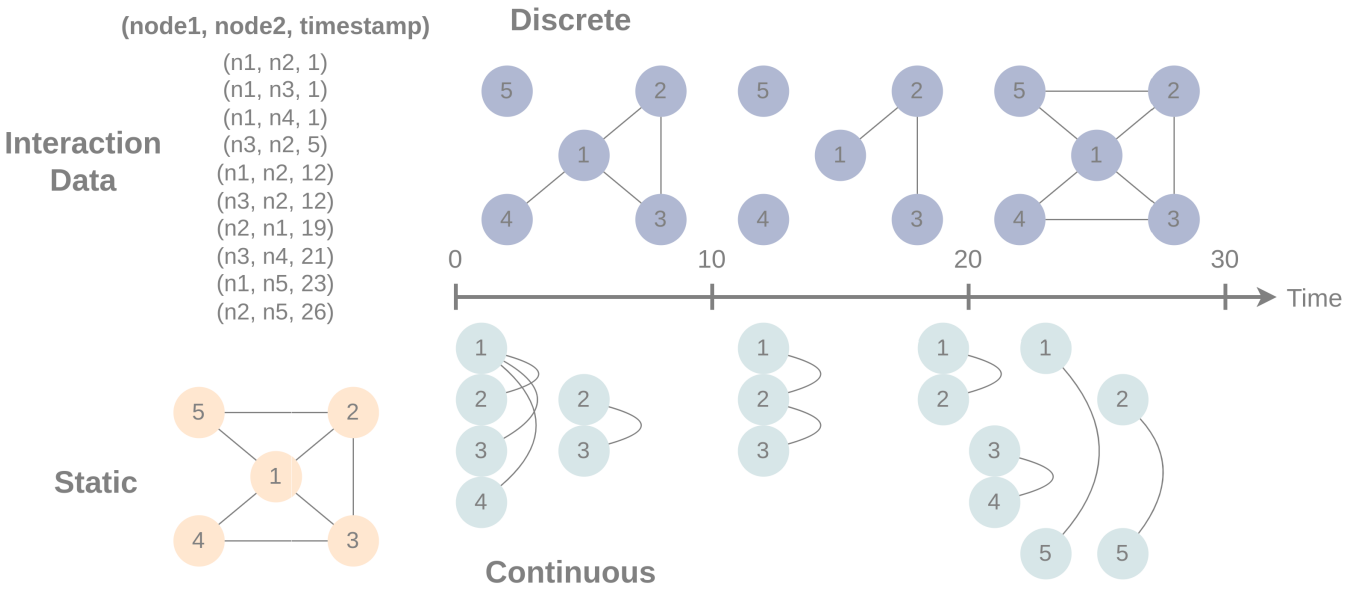

FIGURE 1. An example of a static, discrete, and continuous representations of an interaction network. The interaction data consists of the pairwise interaction between nodes and the time they interact at. The static representation only shows the nodes interacting but with no time information. The discrete representation collects the links into snapshots (here the snapshots are of size 10). The continuous representation includes exact temporal information.

to represent our continuous networks. A contact sequence is a time-ordered list of triplets, where one triplet represents an interaction between two nodes at a given time.

$$CS = \{(u_i, v_i, t_i); i = 1, 2, \ldots\}, \tag{2}$$

where u_i and v_i is the node pair and t_i is the time the nodes interacted.

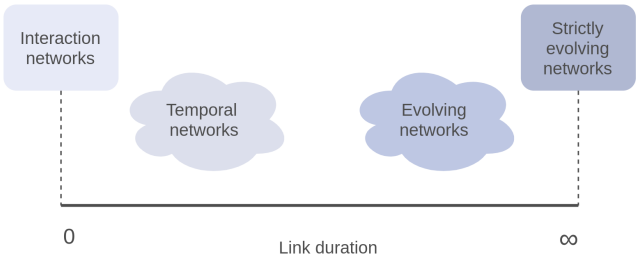

FIGURE 2. The link duration spectrum visualizes the different network types on a spectrum from networks with links of instantaneous duration (interaction networks) to networks with links of infinite duration (strictly evolving networks).

The link duration spectrum (Fig. 2) distinguishes between dynamic networks based on a link's time-to-live (TTL).

Dynamic networks can also be distinguished by the link duration spectrum [8]. Evolving networks are characterized by links persisting for longer, while links in temporal networks are ephemeral. An interaction network is a type of temporal network where links have no duration.

An instantaneous snapshot of an evolving network yields a network structure, an instantaneous snapshot of a temporal network may yield no edges at all. Links in temporal networks do not persist for long, therefore a time-window is required to observe a network structure. Interaction networks are special cases of temporal networks and strictly evolving networks are special cases of evolving networks. Links (interactions) are instantaneous in interaction networks and no link disappears in a strictly evolving network.

The distinction between discrete and continuous networks is important because a model, e.g. a DGNN, is made to encode one of the representation forms. A discrete DGNN can only encode networks represented as discrete networks and a continuous DGNN can only encode networks represented as continuous networks. In this work, we feed discrete graphs (Equation 1) to the discrete models and contact sequences (Equation 2) to the continuous models.

B. GRAPH NEURAL NETWORKS & LINK PREDICTION HEURISTICS

Graph neural networks (GNN) have seen a surge in popularity in recent years. GNNs are representation learning models which aim to store a latent representation of a graph structure. GNN models use message passing to aggregate features of neighboring nodes together [15]. A common output of a GNN layer is node embeddings. Most GNNs can only encode static networks, in this work we refer to these models as static GNNs.

Dynamic graph neural networks (DGNN) is a subclass of GNNs capable of encoding dynamic networks [8].[2] The two main types of DGNNs are discrete DGNNs and continuous DGNNs which are capable of encoding discrete and continuous networks respectively. Due to the difference

[2]There also exist GNN architectures for static networks with dynamic node and/or edge labels. These so-called spatio-temporal graph neural networks are discussed in some GNN surveys but are out of scope of this study [15], [16].

in network representation, their architectures are radically different.

Discrete DGNN architectures usually consist of GNN layers and time series layers, with the time series layers being RNN layers or attention layers. Continuous DGNNs on the other hand have more varied architectures. The key challenge for the continuous models is to encode the inter-event time between node interactions. For this, some models [17], [18] use RNNs, such as a time-aware LSTM [19], other models use temporal point processes [20]–[22] and the most recently emerged approaches use time embeddings [23]. There are currently only two time embedding based models, TGAT [11] and TGN [12]. Discrete DGNNs iterate over the data snapshot by snapshot, while continuous DGNNs iterate over the data edge by edge.

Link prediction heuristics are compared to GNNs fairly simple methods. The heuristics calculate a similarity score which is then used to rank the links. The heuristics we compare are all based on the idea that two nodes are more likely to form links if they have common neighbors. The simplest implementation of this idea is the common neighbor heuristic [24] where the similarity score is given by

$$|\Gamma(u) \cap \Gamma(v)| \tag{3}$$

where $\Gamma(u)$ is the neighbors of node u. There is a rich literature on these methods in the network science community [24], [25].

The two modern methods that we compare are slightly more complex. Newton [26] combines the node degree (degree centrality) and the shortest path between two nodes to compute the similarity score. CCPA [27], combines the common neighbor heuristic with the shortest path approach.

C. DYNAMIC LINK PREDICTION

Traditionally, link prediction is the task of predicting links in static networks. Where there is no distinction between predicting missing links and future links. When predicting links in dynamic networks, this distinction is important. Missing link prediction can be referred to as *interpolation* and future link prediction can be referred to as *extrapolation* [28]. In this work, we compare methods on the future link prediction (extrapolation) task on discrete networks.

We predict which links will appear in the next time-window. From the perspective of discrete methods, this is a very natural prediction task. The methods read in time-windows (snapshots) which are in chronological order and predict which links appear in the next snapshot. Adapting static and continuous models for dynamic link prediction is more involved. We cover the details on how static, discrete, and continuous models are adapted to dynamic link prediction in Section III-E.

Link prediction can be seen as a special case of dynamic link prediction, where there are only three snapshots; the train, validation, and test snapshots. In that sense, dynamic link prediction is simply an extension of link prediction with multiple snapshots in the train, validation, and test sets.

D. FAIR COMPARISONS AND BENCHMARKS

A benchmark with representative datasets, thoughtfully applied metrics and clear reporting can help, but is not by itself sufficient for a fair comparison between different methods [29]. A fair comparison requires, among other things, that each model is tuned in a consistent manner to fit each dataset. The words benchmarking and comparative analysis can often be used interchangeably when describing works that compare different models. However, we have observed a trend where there are two related but distinct types of comparative papers in the machine learning community. The first kind of paper, which we refer to as benchmarking papers, are papers that aim to establish standardized benchmarks for specific tasks [2], [3]. Their contributions often come in the form of presenting novel datasets, or novel evaluation frameworks for established tasks. These works tend to not tune their compared methods thoroughly, a fair comparison can then be achieved between new models assuming that each author thoroughly tunes their respective models. In this type of paper, the expensive task of tuning each model is thus distributed to new authors. The second type of paper, which we refer to as a comparative analysis, are papers that aim to provide a fair comparison of existing models [1], [30]. Their contributions often come in the form of an analysis providing insight into how the models compare and the effects of hyperparameters. In these terms, this work is intended to be a comparative analysis rather than a benchmark.

III. EXPERIMENTAL SETUP

In this section we cover in detail how the comparison is done and the justification for why these choices were made. This includes an overview of the selected datasets and their statistics, the specific methods that we compare, how we define the dynamic link prediction task, how we measure performance, how the framework trains the GNN models including all the different training pipelines it supports, how we optimize hyperparameters, and the hardware and computation time used to run the experiments.

A. DATASETS

Table 1 shows statistics of the datasets used in the experiments. We select six continuous interaction networks and one discrete evolving network (Autonomous) as datasets. The datasets are shown on the link duration spectrum in Figure 3. We chose interaction networks as they allow us to easily convert to more coarse-grained temporal granularities such as discrete networks. Sparser snapshots indicate a greater imbalance between links and non-links, thus making the classification problem harder. Autonomous was chosen as a representative for evolving networks it is also one of very few publicly available network datasets where links have duration and may disappear again.

Enron,[3] is an email communication network, where a link is an email sent between two people. Enron is a small network

[3]http://networkrepository.com/ia-enron-employees.php

TABLE 1. Dataset statistics. Snap density is the average density of the snapshots, Snap size, is the size of the snapshots (time-windows) in days. Num snap is the number of snapshots in the network. Total time is the time period (in days) that the dataset covers. Persistence is the probability of an existing edge to still exist in the next snapshot. Reoccurrence is the probability of an edge that has existed at one point, that is currently not existing, to reappear. For persistence and reoccurrence, we report mean and standard deviation (in parenthesis) across the snapshots.

Dataset	Nodes	Edges	Unique edges	Density	Snap density	Cont. edges	Snap size	Num snap	Total time	Splits	Persistence	Reoccurrence
Enron	151	5,780	1,569	0.13854	0.01379	50,572	30	37	1,137	80-10-10	0.578(0.160)	0.067(0.044)
UC	1,899	22,497	13,838	0.00769	0.00014	59,835	2.2	88	193	71-10-19	0.031(0.024)	0.002(0.002)
Bitcoin-OTC	5,881	23,686	21,492	0.00124	0.00001	35,592	14	135	1,903	70-10-20	0.039(0.059)	0.002(0.004)
Autonomous	7,716	583,946	7,796	0.00026	0.00020	2,335,784	1	99	99	70-10-20	0.233(0.108)	0.006(0.009)
Wikipedia	9,227	39,804	18,257	0.00043	0.00003	157,474	1	30	30	70-15-15	0.300(0.018)	0.040(0.014)
Reddit	10,984	307,593	78,516	0.00130	0.00017	672,447	1	30	30	70-15-15	0.451(0.020)	0.068(0.019)

spatially (number of nodes), but medium-sized temporally with a reasonable number of continuous links and covering a time span of over 3 years. Due to the small number of nodes (151) and a comparatively large number of edges (5,780), it is much denser than the other networks.

UC Irvine messages,[4] shortened to UC, is an online forum network from the University of California, Irvine. Two students are connected if they interact on the same forum post. Thus, this was originally a bipartite network but it has been projected to have nodes of only one type. The odd choice of snapshot size is adapted from EvolveGCN [9] which observes that a smaller snapshot size yields some snapshots without any edges.

Bitcoin-OTC,[5] is a who-trust-whom network of people trading on the Bitcoin OTC platform. A link is an evaluation by one user of another. The bitcoin network is medium-sized in terms of nodes, however, most of its edges are unique edges which indicate that very few edges are reoccurring. A low recurrence rate is expected from the way a link is defined since it is fairly rare to frequently update trust reviews. Lack of reoccurring edges causes each snapshot to be much sparser than most of the other datasets.

Autonomous-systems,[6] shortened to Autonomous, is an internet router communication network. A link is a router exchanging traffic flow with a peer. This network is already aggregated as a discrete network. We follow Pareja *et al.* [9] in selecting the first 99 days and using that as our dataset. It is the only evolving network among the selected datasets, this is reflected by the persistence measure in Table 1. It is also, by far the dataset with the most edges.

Wikipedia,[7] a bipartite Wikipedia page editing network. Nodes are either a Wikipedia user or a Wikipedia page. A link is a user editing a Wikipedia page, it is thus a bipartite graph. Wikipedia also has few reoccurring edges and similarly to Bitcoin, has then comparably sparse snapshots.

Reddit,[8] a bipartite Reddit posting network. Nodes are either a Reddit user or a subreddit. A link is a user posting on a subreddit, it is thus, like Wikipedia, a bipartite graph.

[4]http://konect.cc/networks/opsahl-ucforum/
[5]https://snap.stanford.edu/data/soc-sign-bitcoin-otc.html
[6]http://snap.stanford.edu/data/as-733.html
[7]http://snap.stanford.edu/jodie/wikipedia.csv
[8]http://snap.stanford.edu/jodie/reddit.csv

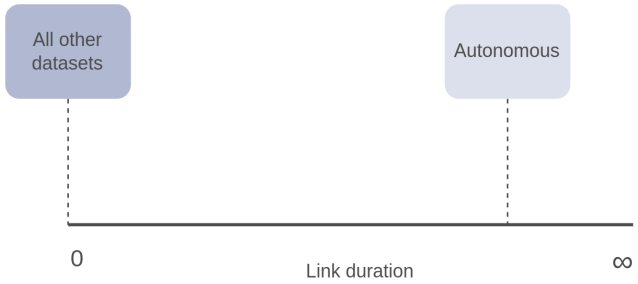

FIGURE 3. The compared datasets on the link duration spectrum from Figure 2.

Reddit is the largest network spatially as it has the largest number of nodes and unique edges.

The UC, Bitcoin, and Autonomous data-splits follow Pareja *et al.* [9], and the Wikipedia and Reddit splits follow Xu *et al.* [11]. We use the same splits so we can compare our results to the previous works. We use a slightly larger train split on Enron as it is a tiny dataset and we had difficulties with the models learning anything if the training set was too small. For details see Table 1.

We prepare two versions of each dataset, a directed continuous interaction network and an undirected discrete network. Continuous models encode the continuous network. The static and discrete models encode the discrete network. In the conversion from continuous to discrete, reciprocal edges are added to make the discrete networks undirected. This was done since not all compared GNN implementations support directed edges. The number of times an edge occurs in a snapshot is added as a weight to the snapshot's edge.

All results are reported predictions on the discrete networks. For continuous models, this is achieved by splitting the continuous parts of the continuous networks into snapshots corresponding to the snapshots in the discrete network. We then let the continuous models encode the continuous network before the target snapshot and then try to predict the link occurrence in the discrete network.

Figure 4 is the dynamic network cube from our previous survey [8]. Figure 4 and Table 2 show different types of network families. The figure and table are color-coded to indicate which type of dynamic network we run the comparative experiments on. As seen from Figure 2 and Figure 3 the Autonomous dataset is of type 1, while all the other

TABLE 2. Terminology of the dynamic network cube.

Node	Temporal granularity	Node dynamics	Link duration	Precise dynamic network term
1	Discrete	Node-static	Evolving	Discrete node-static evolving network
2			Temporal	Discrete node-static temporal network
3		Node-dynamic	Evolving	Discrete node-dynamic evolving network
4			Temporal	Discrete node-dynamic temporal network
5	Continuous	Node-static	Evolving	Continuous node-static evolving network
6			Temporal	Continuous node-static temporal network
7		Node-dynamic	Evolving	Continuous node-dynamic evolving network
8			Temporal	Continuous node-dynamic temporal network

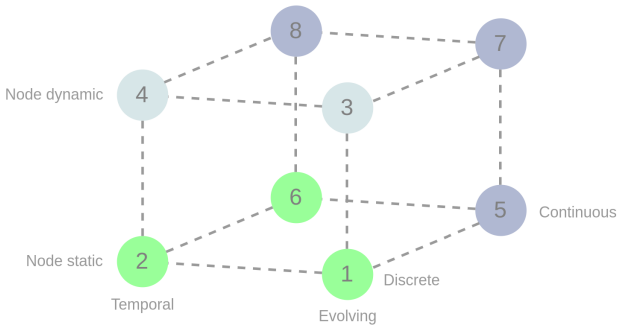

FIGURE 4. The dynamic network cube. Each node represents a dynamic network family. The nodes covered in this work are colored in green. The terminology is covered in Table 2.

TABLE 3. Overview of compared methods.

Models	Temporal granularity	Method/architecture type
CN [24]	Static	Heuristic
AA [35]	Static	Heuristic
Jaccard [24]	Static	Heuristic
Newton [26]	Static	Heuristic
CCPA [27]	Static	Heuristic
GCN [31]	Static	Spectral GNN [15]
GAT [32]	Static	Spatial GNN [15]
EGCN-H [9]	Discrete	Integrated DGNN [8]
EGCN-O [9]	Discrete	Integrated DGNN [8]
GC-LSTM [10]	Discrete	Integrated DGNN [8]
TGAT [11]	Continuous	Time embedding based [8]
TGN [12]	Continuous	Time embedding based [8]

datasets are of type 2 or 6 depending on which method (see Table 3) they are prepared for. While many of the methods compared support node dynamics, we opted to only compare on datasets where new nodes did not appear in the test set and we focus on the link prediction problem. This was done due to the experiment being computationally expensive (see Section III-G). We did not run any continuous models on evolving networks as the continuous models we compared are not suited for evolving data. In fact, we are unaware of any continuous DGNN that is suited for such networks [8].

B. METHODS

For each of the three network categories, static, discrete, and continuous, we select at least two GNNs to benchmark. We select GCN [31] and GAT [32] as static models as they are known to be fairly universal and representative. To represent discrete models we select EGCN-H, EGCN-O and GC-LSTM. The EGCN [9] models differ from other DGNN models as they use RNNs to evolve the GCN weights rather than evolve node embeddings. GC-LSTM is selected as it is an integrated DGNN and the architecture was used by [10] specifically for dynamic link prediction. The GC-LSTM encoder integrates an LSTM and a spectral GCN [33], it is very similar to the first DGNNs introduced by [34]. To represent continuous models we select two time embedding based continuous DGNNs, TGAT [11] and TGN [12].

To represent link prediction heuristics we select three well-established methods [24], [25], Common Neighbors (CN), Adamic-Adar (AA) [35] and Jaccard. As well as two modern heuristics; Newton's gravitational law (Newton) [26] and Common Neighbor and Centrality based Parameterized Algorithm (CCPA) [27]. The modern methods use the shortest path between nodes to get scores for missing links beyond common neighbors, whereas the other methods give a similarity score of 0 if there are no common neighbors. Unlike the other heuristics, Newton does not rely on common neighbors, but rather on the degree centrality of the compared nodes and distance between them.

Table 3 is an overview of the methods, their temporal granularity, and method type (architecture type in the case of GNNs). The architecture types are categories of GNNs identified by surveys [8], [15].

We use either standardized implementations or the original authors' code. For GCN and GAT, we use the PyTorch Geometric implementation [36],[9] for GC-LSTM we use the PyTorch Geometric Temporal implementation [37][10] and we use the original authors' code for the EGCN models,[11] TGAT[12] and TGN.[13]

Unless otherwise stated, we do not modify any of the tested models. Some minor modifications were made to GC-LSTM, TGAT, and TGN to enable the comparison. The PyTorch Geometric Temporal implementation of GC-LSTM [37] was modified to enable sliding windows in the same manner as originally used by EvolveGCN [9]. For TGAT and TGN we leave the training unchanged, but add the functionality to

[9]https://github.com/rusty1s/pytorch_geometric
[10]https://github.com/benedekrozemberczki/pytorch_geometric_temporal
[11]https://github.com/IBM/EvolveGCN
[12]https://github.com/StatsDLMathsRecomSys/Inductive-representation-learning-on-temporal-graphs
[13]https://github.com/twitter-research/tgn

extract node embeddings to enable comparison to the discrete models.

We attempted to add the static GNN, SEAL [38] to our benchmark as it is a promising GNN [39] specifically targeted at link prediction. However, the model requires preprocessing 2-hop subgraphs for every node pair. We found this to not scale well. This is particularly due to us not using negative sampling (sampling of non-links) during validation and testing. Not using negative sampling allows us to get a complete and accurate picture of the performance of the methods. Simply storing these preprocessed subgraphs for one of our larger datasets would require several TB of disk space.

C. TASK & METRICS

The dynamic link prediction task is to give a probability score for each node pair in the network that a link between the two nodes will be created in the next snapshot. The prediction problem is extremely unbalanced; in each selected network (except for Enron) we see more than 10,000 non-links for every link. This leads [40] to recommend using precision-recall curves when evaluating link prediction. We use mean average precision (mAP), which is equivalent to the area under the precision-recall curve. We also report the Area under the receiver operating characteristic curve (AUC) as this is commonly done for link prediction methods.

The link prediction task becomes harder the more imbalanced the classes are. The class imbalance can be measured through network density. Since we test on snapshots, the mean snapshot density in Table 1 indicates how imbalanced the prediction task on each dataset is. The mAP score was selected in part because it is sensitive to this increased difficulty. We, therefore, expect mAP scores for networks with lower snapshot density to be lower.

This extreme class imbalance makes it tempting to use sampling of non-links (negative sampling) to balance the datasets when reporting the metrics. However, according to Yang *et al.* [40] the use of negative sampling has been shown to cause a high variance in the reported AUC. This variance may lead to incorrect ordering of models and would affect hyperparameter optimization decisions; e.g. which epoch to select when applying early stopping, and which hyperparameter settings perform better. While this can be counteracted by running multiple runs, we opted to minimize variance where we could. Contrary to the recommendations by Yang *et al.*, GNN link prediction works tend to use a negative sampling ratio of 1 to 1 [11], [12], [38]. As far as we know there is no explicit comment by these works on the validity of Yang *et al.*'s recommendations for GNNs evaluation.

D. EVALUATION SCHEME

This subsection covers details of the experiment relevant to how the final evaluation is done and aspects of the framework that is common for all GNN models. We perform a chronological train-validation-test split. The snapshot size and split sizes are shown in Table 1. All models use the same snapshot sizes and the same train-validation-test splits. We report the results of the test results of the best performing (highest mAP) validation run.

We ensure that all models are trained in the same way by using a common framework that supports models of the three different temporal granularities (static, discrete, and continuous). Our framework is an extension of the framework used by EvolveGCN [9]. The EvolveGCN framework supports training of discrete DGNN models with snapshots and sliding windows, the hyperparameter optimization supported was random search. Major differences with that framework and ours include adding: (i) two additional training pipelines, one for static and one for continuous models (we use the already existing pipeline for the discrete models), (ii) grid search functionality, (iii) enabling the use of continuous embeddings on discrete network representations, and (iv) link prediction heuristics. The evaluation on the validation and test set is however identical, thus our results are comparable to Pareja *et al.* [9]. Details on the three training pipelines are found in, Section III-E.

Figure 5 shows the architecture used by the framework. All GNN models use the same decoder (a two-layered MLP) and binary-cross entropy loss. Models are trained by predicting the next snapshot in the training set. For each epoch, the framework iterates through the dynamic network, snapshot by snapshot. For each iteration, the framework feeds the GNN with the network before the current time step and evaluates the predictions of the GNN against the next snapshot. Previous snapshots may be aggregated, but the next snapshot is always of the same size to ensure that the results are comparable. The heuristics don't require any training and are thus run directly on the test set.

The loss function is weighted, giving non-existing links a weight of 0.1 and existing ones a weight of 0.9. Neural network weights are initialized using uniform random initialization. Static and discrete models use a negative sampling ratio of 1 to 100 during training, and continuous models a ratio of 1 to 1. Importantly, we do not use negative sampling when reporting validation and test scores. We use the one-hot encoded node degree as initial node features for static and discrete models. This was shown by Errica *et al.* [1] to improve the performance of GNNs on graph classification. This is a key design decision, as it dictates the size of the node embedding. The continuous models are limited to using the same dimensions on the node and edge features. Following the original works, we use a node and edge feature size of 172. The node features are randomly initialized and for datasets without edge features (every dataset except for Wikipedia and Reddit) we also randomly initialize the edge features.

The use of node degree as node features means that static and discrete models rely solely on the graph structure to perform link prediction. In many cases, a dataset would have informative node or edge features that can aid with prediction tasks. The only cases where informative features are used in this analysis are the edge features of the Wikipedia and Reddit datasets. And these are only leveraged by the continuous DGNNs. If GNNs and DGNNs are able to leverage

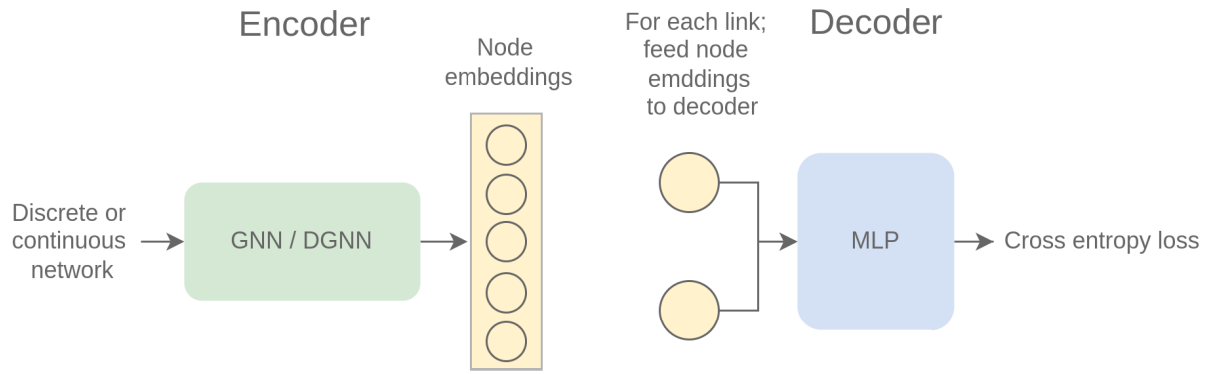

FIGURE 5. The architecture surrounding the GNNs and DGNNs. For each link we try to predict, the corresponding node embeddings are passed to a two-layered multilayered perceptron (MLP). For a view of the framework surrounding the deep learning architecture see Figure 6.

informative node or edge features we would expect to see a significant performance increase. This is explored in Section IV-B.

E. TRAINING PIPELINES

While the evaluation is identical for all methods, the training procedure is different between the different kinds of methods. This is a necessity due to the different network representations that the methods operate on. The static and discrete pipelines are shown in Figure 6

1) STATIC

Static GNNs encode a graph. Since our training set includes multiple snapshots, we convert these snapshots into one graph to enable the training of the GNN. We can aggregate an arbitrary number of snapshots into one snapshot by including a link in the output snapshot if it occurs in any of the input snapshots, thus turning a discrete network into a static one. We do this in three different ways and consider these approaches a hyperparameter.

The most straightforward way to train a GNN on a dynamic network is to combine all the snapshots in the training set into one big snapshot. We call this approach 'static'. This is the approach taken by traditional link prediction and continuous DGNN works [11], [12]. This is the only approach not to train in a ''roll forward'' manner.

It is presumably beneficial to exploit the temporal information in the training set and roll forward during training. One way to do this is to only encode the previous snapshot when attempting to predict the next snapshot. This does not require any snapshot aggregation. This can be seen as a sliding window of size 1 and is the approach used by Pareja *et al.* [9] for static GNNs.

Complex networks tend to be rather sparse. It might therefore be beneficial to use a sliding window. We explore sliding windows of sizes 5 and 10. Size 10 is the default for EGCN, we chose to additionally use size 5 to investigate whether the size of the sliding window is influential. For the static models, these ''sliding snapshot windows'' are aggregated into one snapshot. For even sparser networks it may be beneficial to represent the dynamic network as an evolving network. For this, we use an expanding window. We refer to this option as 'expanding'.

2) DISCRETE

Most discrete models inherently support multiple snapshots, but the number of snapshots cannot vary during training. It is therefore necessary to use a sliding window that feeds a consistent number of snapshots to the model. We use sliding window sizes of 1, 5, and 10. While this is comparable to the sliding window of the static models, it is also different since the snapshots in these sliding windows are not aggregated together.

3) CONTINUOUS

Continuous models have no notion of snapshots, and we are unaware of anyone training continuous models on discrete networks. As this is a comparative study we aim to train the models the same way they were originally trained, yet also in a way that allows us to compare the results fairly. Our solution is to train in two steps. Firstly we train the encoder, secondly the decoder.

Continuous models are trained edge-by-edge. Like other time-series models the edges are batched. This hinders us from training the continuous models end-to-end with our decoder (recall that we want to use the same decoder for all models) since the decoder backpropagates on each snapshot.

It is theoretically possible to train the continuous models end-to-end with our decoder by changing the edge batch size from snapshot to snapshot. This will however lead the number of edges in the batches to change from batch to batch. Whether the radical change in batch size throughout training is a viable way to train is unknown. However, we deem it as too different from the original way these networks were trained to include this approach in our study. We plan as the future work to explore end-to-end training on snapshots for continuous models.

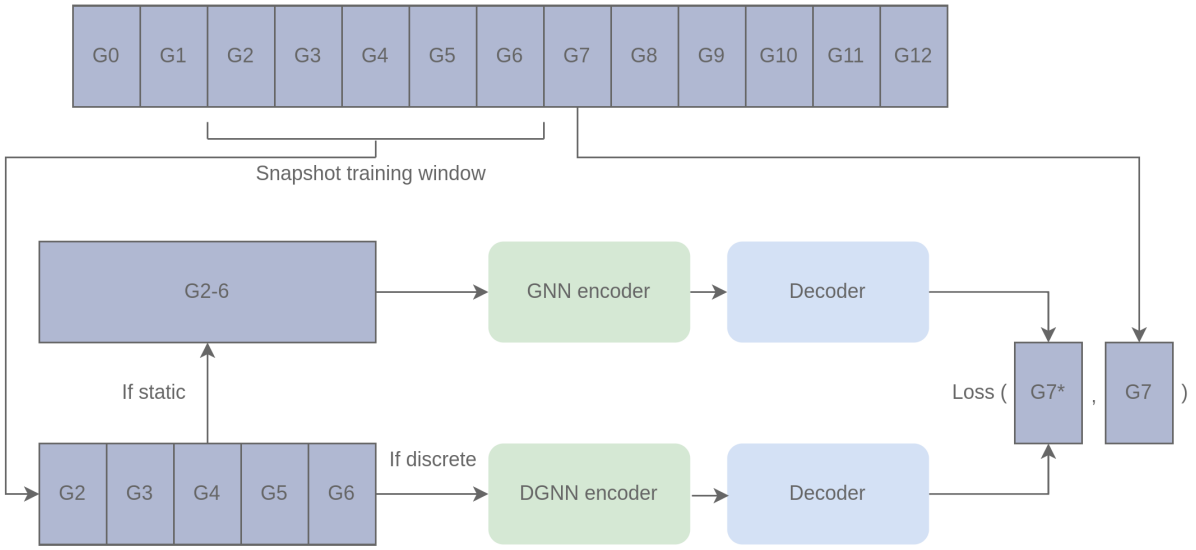

FIGURE 6. The framework's static and discrete pipelines. Along the top are 12 snapshots (each of them a graph). The snapshot training window is in this example of size 5 and feeds the snapshots to the DGNN encoder, or if trained in a static fashion, aggregates the snapshots in the sliding window, and feeds the resulting graph to a GNN. The predictions of which links will appear in the next snapshot (G7*) are compared to the actual next snapshot (G7). Then the snapshot training window will slide forward one step and feed another set of snapshots to the models. One epoch is one pass through all snapshots. For a detailed view of the encoder and decoder see Figure 5.

We opt to train the continuous models in two steps. First, we train identically to how the models were originally trained, with a constant batch size (essentially pretending snapshots do not exist), using contrastive learning, and a negative sampling rate of 1 to 1. We then extract the node embeddings and train our decoder separately with the encoder (the continuous DGNN) frozen, this time with a batch size matching the number of edges in the snapshot. Training with the same decoder allows us to use the same class weights and negative sampling rate (1 to 100) as the static and discrete models.

To speed up decoder training, we cache the node embeddings produced by the frozen encoder in the first epoch of the decoder training. This speeds up training by a factor of at least $10x$.

Despite the effort to optimize training speed, we opted to not test the continuous models on the Autonomous dataset. A single (encoder) epoch took on average 16 hours to run (wall-time) on our hardware (see Section III-G for the hardware). The discrete evolving network could be preprocessed into a more suitable format for continuous models, but doing so is non-trivial and we consider that outside the scope of this work.

F. HYPERPARAMETERS

We search for good hyperparameters by performing a grid search. While being very time-consuming, this allows us to analyze the impact that different time-windows have model performance. We then run each model four times with the best found hyperparameters with different random seeds.

If the hyperparameter is not included in the grid search, we use the values used by the original studies. We search the learning rate on all models as it is recommended by Goodfellow *et al.* [41] as an important hyperparameter to optimize. For static GNNs and discrete DGNNs we chose to search the hidden layer size to regulate the number of parameters of the model as too few might lead to underfitting and too many to overfitting.

For the static GNNs we search the parameters: learning rate, snapshot training window, and hidden layer size. The search for parameters on discrete DGNNs is identical to the static except for the snapshot training window which only searches sliding windows. The grid search is slightly modified on the Enron dataset where we search smaller layer sizes due to the dataset being very small. The parameters searched in the grid search are shown in Table 4, the selected hyperparameters are shown in Table 8.

Continuous models are trained with the original hyperparameters and we perform a grid search on the second training stage where we train the decoder. The parameters optimized are learning rate, decoder learning rate, and decoder weight decay.

By default, link prediction heuristics don't predict scores for already existing edges. We choose to explore two options, we refer to these as the different ways for existing edge treatment in Table 4: (i) calculating a score as if the edge didn't exist; and (ii) the default option, simply assuming existing links will persist. We also search the snapshot training window as shown in Table 4.

Some hyperparameters are common for all GNNs. The maximum number of epochs is 500. We use early stopping with an early stop patience of 100. The TGAT and TGN encoders are trained as in their original works, with a maximum of 50 epochs (epochs on continuous models are significantly longer than on static or discrete models. Training using a high number of epochs is therefore impractical). We evaluate the models on the validation set every 5 epochs.

TABLE 4. Hyperparameters searched by the grid search.

Hyperparameter	Prediction Methods	Values
Snapshot training window	Heuristic	1, 5, 10, static
	Static	1, 5, 10, expanding, static
	Discrete	1, 5, 10
Existing edge treatment	Heuristic	Default, score
Learning rate	Static, discrete, continuous	0.0001, 0.005, 0.001, 0.05, 0.01
Hidden layer size	Static, discrete	50, 100, 200 (10, 20, 30 on Enron)
Decoder learning rate	Continuous	0.0001, 0.005, 0.001, 0.05, 0.01
Decoder weight decay	Continuous	0, 0.0001, 0.01

Other hyperparameters are specific for models; we use the original authors' parameters and we keep them the same across all datasets. For GAT we use 8 attention heads and a dropout value of 0.5. GC-LSTM uses a spectral GCN [33] which approximates the graph convolution and takes a hyperparameter K. It indicates the number of hops included in the neighborhood convolution. We use $K = 3$. For continuous models, we use a batch size of 200, 2 attention heads, and a dropout of 0.1. For TGN we activate memory as that is its major feature distinguishing it from TGAT.

G. RUNTIME & HARDWARE

The grid search consisted of searching 2910 different parameter settings, each setting took on average approximately 3.9 hours to complete. The grid search took around 11, 270 hours. Getting the final results, including running the best found hyperparameters with four different seeds was 168 different runs. Each of these runs took on average approximately 6.3 hours to complete, so in total those runs took around 1, 050 hours. The parameter budget runs were also time consuming (174 runs), both the grid search and stability took 1260 hours. In total, running the experiments took 13, 590 hours (566 days).

The runs were computed in parallel on HPC clusters. The cluster nodes varied slightly in their architecture, but a typical node had a processor equivalent to an Intel Xeon Gold 6126 and a GPU equivalent to an NVIDIA Quadro P6000. A singularity container was used to ensure a consistent runtime environment [42].

IV. RESULTS AND DISCUSSION

In this section we show and discuss the results of the experiments. This includes the comparison of the methods, an exploration of the importance of edge features for continuous DGNNs, the importance of a snapshot training window, and an experiment that ensures that the compared models have the same number of learnable parameters.

A. MODEL COMPARISON

We report the mAP and the AUC scores shown in Table 5 and Table 6 respectively. All results are the average scores taken from four runs of the best parameter setting found by the grid search. The selected parameter settings are reported in Table 8.

1) HOW DO LINK PREDICTION HEURISTICS AND GNNs COMPARE?

On all datasets the heuristic baselines outperformed the GNNs. Among the GNNs, the discrete DGNNs performed consistently better, with continuous DGNNs performing poorly, except for on the Wikipedia and Reddit datasets where the continuous DGNNs performed relatively well.

These findings are similar to the findings of Huang *et al.* [43] which determined that a simple method, using label propagation followed by logistic regression, can outperform GNNs. The link prediction heuristics are even simpler, so it is still surprising that they perform this well. Reasons for this may be that the heuristics, despite being relatively simple, are based on observed phenomena in complex networks. The GNNs seem to not be able to learn these phenomena.

2) WHY DO SOME MODELS SCORE LOW IN ONE METRIC AND HIGH IN ANOTHER?

An explanation for this disparity lies in the extreme class imbalance inherent to link prediction [40]. The AUC score relies on the false positive rate. Due to the large number of non-existing links, the false positive rate may stay relatively low despite the precision is also being low. In fact, the mAP and AUC are shown to be approximately the same, except for the precision of the highest ranked links [44]. Our results thus indicate that the link prediction heuristics are better at ranking links initially, but later links are ranked better by the GNNs.

For applications focused on the highest ranked links, which is common in information retrieval and recommender systems, our results indicate that link prediction heuristics will typically outperform GNNs in the absence of informative node or edge features.

3) HOW DO DIFFERENT GNNs COMPARE?

Among the GNN models, DGNNs performed better than static GNNs. In terms of mAP, the discrete models performed well, and in terms of AUC, the continuous models did well. With the exception of Enron, a DGNN always outperformed the static GNNs across all reported metrics. Particularly GC-LSTM performs comparatively well. Static GNNs are on some datasets, e.g. Wikipedia, closer to the performance of random predictions.

TABLE 5. Dynamic link prediction mAP scores. The scores reported are the mean and standard deviation (in parenthesis) of mAP scores averaged across four runs with different random seeds. The heuristics are deterministic and thus they all have a standard deviation of 0. The best performances are highlighted in **bold**, and the highest scoring GNN performances are underlined. Cells marked by † were not run (see Section III-E3). Random is completely random predictions. R-embed is predictions when the decoder is given random embeddings.

Models	Enron	UC	Bitcoin-OTC	Autonomous	Wikipedia	Reddit
Random	0.003(0.000)	$2 \cdot 10^{-5}$(0.000)	$2 \cdot 10^{-6}$(0.000)	$2 \cdot 10^{-4}$(0.000)	$3 \cdot 10^{-5}$(0.000)	$2 \cdot 10^{-4}$(0.000)
R-embed	0.003(0.000)	$3 \cdot 10^{-5}$(0.000)	$3 \cdot 10^{-6}$(0.000)	0.002(0.001)	$3 \cdot 10^{-5}$(0.000)	$2 \cdot 10^{-4}$(0.000)
CN	0.207	0.338	0.365	0.060	0.190	0.014
AA	0.439	**0.722**	**0.693**	0.719	**0.755**	**0.794**
Jaccard	**0.571**	0.440	0.464	0.131	0.356	0.023
Newton	0.219	0.466	0.459	0.279	0.169	0.062
CCPA	0.281	0.066	0.273	**0.931**	0.087	0.196
GCN	0.337(0.027)	0.021(0.005)	$1 \cdot 10^{-5}$(0.000)	0.012(0.014)	$3 \cdot 10^{-4}$(0.000)	0.025(0.010)
GAT	0.055(0.021)	0.018(0.007)	$4 \cdot 10^{-5}$(0.000)	0.025(0.017)	$4 \cdot 10^{-4}$(0.001)	0.015(0.004)
EGCN-H	0.307(0.043)	0.013(0.004)	0.002(0.001)	0.190(0.034)	0.004(0.001)	0.039(0.009)
EGCN-O	0.360(0.011)	0.014(0.002)	0.002(0.001)	0.165(0.033)	0.005(0.001)	0.032(0.016)
GC-LSTM	0.214(0.123)	0.044(0.003)	0.002(0.001)	0.386(0.029)	0.007(0.001)	0.097(0.007)
TGAT	0.025(0.001)	0.006(0.001)	$6 \cdot 10^{-5}$(0.000)	†	0.006(0.000)	0.108(0.007)
TGN	0.004(0.001)	0.007(0.001)	0.001(0.000)	†	0.004(0.000)	0.042(0.009)

TABLE 6. Dynamic link prediction AUC scores. Formatted identically to Table 5.

Models	Enron	UC	Bitcoin-OTC	Autonomous	Wikipedia	Reddit
Random	0.521(0.048)	0.497(0.013)	0.487(0.008)	0.500(0.000)	0.501(0.001)	0.499(0.001)
R-embed	0.443(0.084)	0.514(0.023)	0.497(0.007)	0.516(0.009)	0.505(0.003)	0.504(0.008)
CN	0.852	**0.999**	**0.999**	0.768	**0.999**	0.994
AA	0.952	**0.999**	**0.999**	0.979	**0.999**	**0.999**
Jaccard	**0.959**	**0.999**	**0.999**	0.978	**0.999**	0.997
Newton	0.868	**0.999**	**0.999**	**0.996**	**0.999**	0.810
CCPA	0.834	0.628	**0.999**	0.973	0.686	0.798
GCN	0.898(0.084)	0.646(0.023)	0.220(0.054)	0.562(0.094)	0.897(0.007)	0.759(0.093)
GAT	0.829(0.017)	0.618(0.053)	0.618(0.029)	0.566(0.103)	0.275(0.009)	0.947(0.025)
EGCN-H	0.913(0.049)	0.690(0.046)	0.788(0.005)	0.878(0.043)	0.675(0.032)	0.951(0.009)
EGCN-O	0.926(0.027)	0.662(0.054)	0.771(0.016)	0.904(0.036)	0.708(0.006)	0.937(0.021)
GC-LSTM	0.819(0.184)	0.579(0.026)	0.792(0.000)	0.944(0.007)	0.706(0.006)	0.968(0.001)
TGAT	0.545(0.055)	0.875(0.005)	0.827(0.046)	†	0.948(0.001)	0.985(0.000)
TGN	0.681(0.153)	0.947(0.001)	0.869(0.014)	†	0.965(0.002)	0.969(0.001)

Even in terms of mAP, the continuous models do comparatively well on Wikipedia and Reddit where they have informative edge features. However, their performance is lacking on other datasets. This implies that they may rely on edge features for good performance; we explore this further in Section IV-B. Their performance in terms of AUC is compared to other GNNs, good. This performance gap, and as explained earlier, difficulty with ranking initial links, may be caused by the choice of negative sampling when training the encoders. During training only one randomly chosen negative sample is used per link. Increasing the quality or quantity of negative samples will possibly improve their performance.

4) WHY DID GC-LSTM OUTPERFORM THE OTHER DGNNs?

In terms of mAP the GC-LSTM model outperformed the other DGNNs on all datasets except Enron.

GC-LSTM is a fairly simple DGNN architecture where each snapshot in the snapshot window is first encoded by a GNN and the resulting node embeddings are then encoded by an LSTM. The EGCN models focus on evolving the GNN weights rather than the node embeddings. The EGCN-O model does not take node embeddings as an input, while EGCN-H only takes the top-k node embeddings as input. We hypothesize that the performance difference is caused by the EGCN models deemphasizing node embeddings.

Another difference in our comparison of GC-LSTM and the EGCN models is the GNNs and their implementation. GC-LSTM uses a GNN from PyTorch geometric [36], while the EGCN models use the original author's GCN [9] implementation.[14]

[14]We attempted to use the PyTorch Geometric Temporal implementation of the EGCN models [37], but the original author's code outperformed the PyTorch Geometric Temporal implementation. We believe this was due to a bug in PyTorch Geometric Temporal that has since been fixed.

TABLE 7. Continuous DGNNs with randomized edge features. '-RE' indicate random edge features.

Models	Wikipedia	Wikipedia-RE	% diff	Reddit	Reddit-RE	% diff
TGAT	0.006(0.000)	0.002(0.002)	−61%	0.108(0.007)	0.088(0.005)	−18%
TGN	0.004(0.000)	0.001(0.000)	−83%	0.042(0.009)	0.023(0.008)	−29%

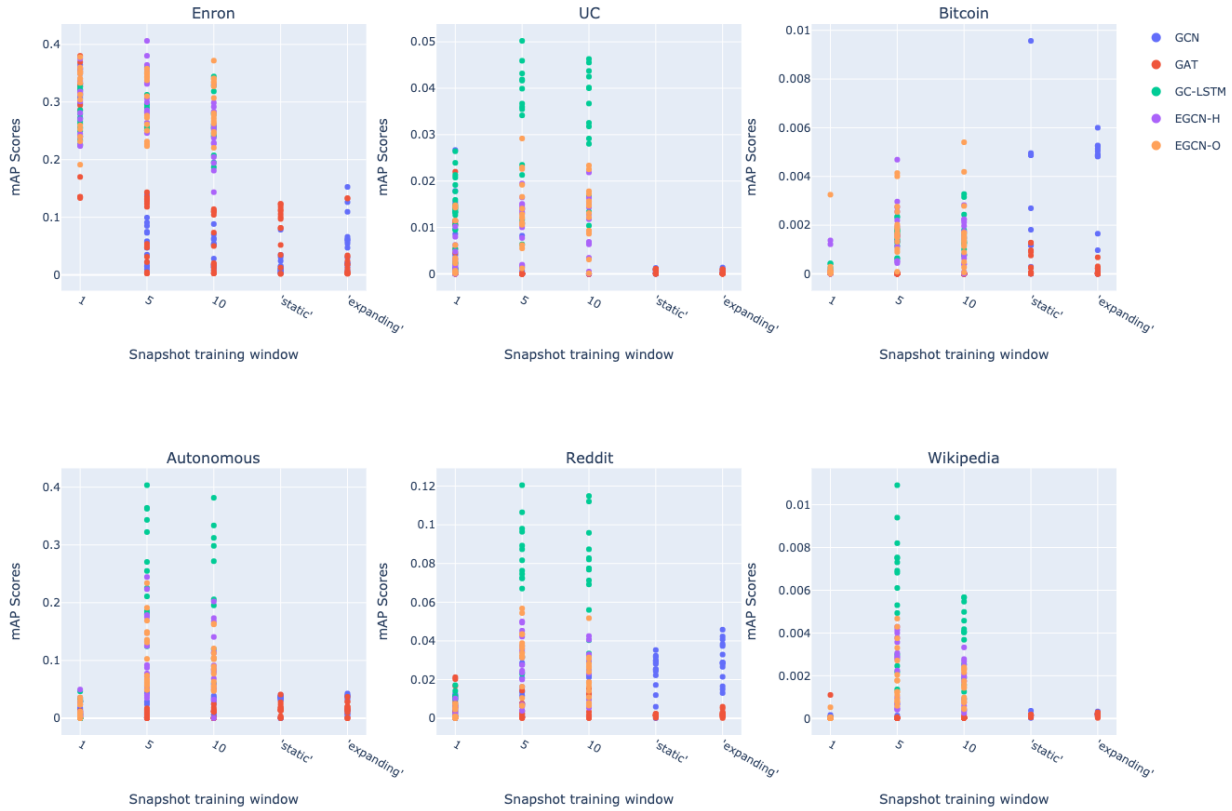

FIGURE 7. Grid search mAP scores against the size of the snapshot training window. The snapshot training window is explained in Section III-E1. Each color represents the five GNNs. All figures show different y-axes for the separate datasets due to large differences in scores between datasets.

5) WHY ARE THE mAP SCORES SO DIFFERENT BETWEEN DATASETS?

The scores vary notably between datasets. Datasets with low snapshot density have lower mAP scores than those with high snapshot density. This reflects the increased difficulty of the classification problem which comes with increased class imbalance [40].

The Autonomous dataset does however not appear to follow this pattern. The methods can perform well despite the network being rather sparse. We speculate that this might be due to Autonomous being an evolving network rather than an interaction network (Section II-A). In slowly evolving networks there is less change between snapshots, and methods only need to predict gradual changes to the network as fewer links disappear. It is shown that network characteristics, such as the clustering coefficient and average shortest path, influence heuristic performance [45]. It is plausible that link duration influences performance as well, further work is needed to confirm this.

When evaluating all possible links, the number of possible self links (links between the same nodes) are comparable to the number of links in the network. Non existing self-links are challenging for GNNs to predict, as they rely on proximity of node embeddings in embedding space and a self link would be predicted from two identical node embeddings. If negative sampling is used in the evaluation the number of self links evaluated would be negligible and the thus bad evaluation of them would not the noticeable in the mAP or AUC score.

B. EXPLORING THE PERFORMANCE OF CONTINUOUS DGNNs

In Section IV-A we hypothesized that the continuous DGNNs relied on informative edge features to achieve good results on the Wikipedia and Reddit datasets. To check this hypothesis, we run TGAT and TGN on these datasets with randomized edge features. The results, shown in Table 7, show a substantial decrease in performance when edge features are randomized, particularly on the Wikipedia dataset. The performance on Reddit has decreased, but not as drastically. This shows that the edge features are an important, but not always critical, asset to aid encoding.

TABLE 8. Hyperparameters selected by the grid search.

Models	Hyperparameter	Enron	UC	Bitcoin-OTC	Autonomous	Wikipedia	Reddit
CN	Snapshot training window	1	1	1	1	1	1
	Existing edge treatment	score	default	default	score	default	default
AA	Snapshot training window	1	1	1	1	1	1
	Existing edge treatment	default	default	default	default	default	default
Jaccard	Snapshot training window	1	1	1	1	1	1
	Existing edge treatment	default	default	default	default	default	default
Newton	Snapshot training window	1	1	1	1	1	1
	Existing edge treatment	score	default	default	score	default	default
CCPA	Snapshot training window	1	1	1	1	1	1
	Existing edge treatment	score	score	default	score	default	default
GCN	Snapshot training window	1	1	10	expanding	expanding	5
	Learning rate	0.01	0.005	0.005	0.05	0.001	0.001
	Hidden layer size	30	50	200	100	200	50
GAT	Snapshot training window	5	1	1	5	1	10
	Learning rate	0.005	0.005	0.005	0.01	0.005	0.005
	Hidden layer size	20	50	200	50	50	200
EGCN-H	Snapshot training window	1	10	5	5	5	5
	Learning rate	0.005	0.005	0.001	0.01	0.005	0.01
	Hidden layer size	30	50	100	100	200	50
EGCN-O	Snapshot training window	10	5	5	5	5	10
	Learning rate	0.01	0.01	0.0001	0.005	0.005	0.005
	Hidden layer size	30	50	100	200	200	50
GC-LSTM	Snapshot training window	10	10	10	10	5	5
	Learning rate	0.005	0.001	0.0001	0.01	0.001	0.01
	Hidden layer size	20	200	50	200	50	200
TGAT	Learning rate	0.0001	0.0001	0.001	†	0.0001	0.0001
	Decoder learning rate	0.001	0.0001	0.05	†	0.001	0.005
	Decoder weight decay	0	0.0001	0.0001	†	0.0001	0
TGN	Learning rate	0.0001	0.0001	0.0001	†	0.0001	0.0001
	Decoder learning rate	0.005	0.001	0.0001	†	0.001	0.001
	Decoder weight decay	0.001	0.0001	0	†	0.0001	0

Overall, the continuous DGNNs performed relatively poorly compared to discrete DGNNs in terms of mAP. We suspect this is caused by: (i) reliance on edge features; (ii) transferring from a continuous setting to a discrete setting takes the embeddings to some extent out of their element and (iii) the hyperparameters of the continuous DGNNs were optimized by the original authors on Wikipedia and Reddit. The performance of the continuous DGNNs can probably be significantly improved by exploring other ways to apply continuous DGNNs to the discrete network representation and by further optimizing the hyperparameters.

C. SNAPSHOT TRAINING WINDOW ANALYSIS

Figure 7 shows the mAP scores found during the grid search on the static and discrete models. One datapoint is one parameter setting in the search. A sliding time-window of size 5 or 10 consistently produces the best results, particularly for the discrete models. This indicates that it is beneficial to use a sliding window when training DGNNs. Most models are spread across a large spectrum of scores, implying that optimizing the hyperparameters is essential for obtaining a representative and good score for both GNNs and DGNNs.

Table 8 shows the hyperparameters selected by the grid search as the best performing. Link prediction heuristic methods have all the same or very similar optimal hyperparameters. It is especially interesting that the heuristics always performed best with a sliding window of size 1. The hyperparameters for the graph neural network models have no obvious commonalities beyond the sliding time-windows shown in Figure 7.

D. USING A PARAMETER BUDGET

The grid search explored different layer sizes. However, this did not take into account the total number of

TABLE 9. Layer size and total number of learnable parameters for the equal parameter budget runs. The number of learnable parameters of the encoder is in parenthesis.

Models	Enron		UC		Bitcoin-OTC		Autonomous		Wikipedia		Reddit	
GCN	30	(3240)	50	(15400)	200	(199600)	100	(84700)	200	(131400)	50	(130400)
GCN+P	120	(23760)	920	(1083760)	420	(511560)	1200	(2336400)	360	(294120)	1600	(6652800)
GC-LSTM	10	(23340)	200	(1097400)	50	(508350)	200	(2271000)	50	(303750)	200	(6617400)

TABLE 10. mAP scores of GCN and GC-LSTM compared on a parameter budget. GCN+P is the GCN model with the same number of learnable parameters as the GC-LSTM.

Models	Enron	UC	Bitcoin-OTC	Autonomous	Wikipedia	Reddit
GCN	$\mathbf{0.337(0.027)}$	$0.021(0.005)$	$1 \cdot 10^{-5}(0.000)$	$0.012(0.014)$	$3 \cdot 10^{-4}(0.000)$	$0.025(0.010)$
GCN+P	$0.194(0.111)$	$0.014(0.009)$	$\mathbf{0.005(0.000)}$	$0.090(0.087)$	$1 \cdot 10^{-4}(0.000)$	$0.023(0.018)$
GC-LSTM	$0.214(0.123)$	$\mathbf{0.044(0.003)}$	$0.002(0.001)$	$\mathbf{0.386(0.029)}$	$\mathbf{0.007(0.001)}$	$\mathbf{0.097(0.007)}$

learnable parameters. The DGNNs, therefore, ended up with much more learnable parameters than the static GNNs. To explore whether the discrete models can fit the data better simply due to having more parameters, we run GCN with the same number of parameters as the best performing GC-LSTM setting. For the exact change in layer size and the resulting total number of parameters, see Table 9. We perform a full grid search to locate good values for learning rate and snapshot training window given these new layer sizes.

The mAP scores for the GCN with more parameters are compared to the original scores of GCN and GC-LSTM in Table 10. In general, the additional parameters enabled the GCN to achieve a marginally higher score on the larger datasets, but not enough to outperform GC-LSTM, except for on the bitcoin dataset. With the higher number of parameters the GNNs gained more modeling power and they should then be able to fit more patterns in the data which leads to improved performance.

V. CONCLUSION AND FUTURE WORK

In this study, we introduce the DISCO framework that enables the comparison of discrete and continuous DGNNs. We use this framework to perform a comprehensive comparison of link prediction heuristics and three types of GNNs on the dynamic link prediction task. Comparing these different models and these different types of models provides crucial context for understanding DGNNs. This extends our previous theoretical comparison of DGNNs by adding an empirical comparative analysis.

Link prediction heuristics performed better in terms of mAP and AUC. We believe the mAP metric is more indicative of link prediction performance, due to most applications being most interested in the highly ranking links [40].

Despite heuristics being simple, they prove to be strong baselines. Future work on GNNs and DGNNs which are applied to link prediction should be compared to at least one link prediction heuristic and ideally, a variety of different heuristics, note that there are many more heuristics than what we have compared in this study [25]. However, while heuristics currently outperform GNNs, the heuristics cannot leverage informative node or edge features, nor do they leverage temporal patterns. Therefore, GNNs and DGNNs have a lot of potential to improve their performance beyond the heuristics. Not only do the deep models have the potential to outperform heurstics given informative features, but with the rapid progress in this space, they are likely to improve at utilizing graph structure to a point where they would rival or surpass heuristics even without the use of features.

We find that the snapshot training window greatly affects performance. Future work should explore multiple snapshot training window sizes. Despite searching multiple different static network representations, the discrete DGNNs consistently outperformed static GNNs. Our results also indicate that network characteristics, such as the link duration (temporal vs evolving networks) influence prediction performance. Also, continuous DGNNs performed comparatively well when they had informative edge features, but not without them.

Exciting directions for future work include (i) incorporating link prediction heuristics and recent advances in GNNs into DGNNs; (ii) exploring effective ways of training continuous DGNNs on discrete networks; (iii) exploring the influence of dynamic network characteristics on link prediction performance and (iv) expanding this benchmark to include additional methods and datasets. Interesting additional models to compare include other discrete DGNNs [46], approaches using auto-encoder loss functions [47], generative adversarial network based approaches [48], [49] and temporal point process based continuous DGNNs [21], [22]. Most of these methods were excluded from this experiment due to not having available implementations in PyTorch.

Improving the performance of both discrete and continuous DGNNs remains an exciting research avenue. We consider this work a first step towards improving dynamic link predictions and an important foundation for future work.

REFERENCES

[1] F. Errica, M. Podda, D. Bacciu, and A. Micheli, "A fair comparison of graph neural networks for graph classification," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–16.

[2] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," 2020, *arXiv:2003.00982*.
[3] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," 2020, *arXiv:2005.00687*.
[4] Z. C. Lipton and J. Steinhardt, "Troubling trends in machine learning scholarship: Some ML papers suffer from flaws that could mislead the public and stymie future research," *Queue*, vol. 17, no. 1, pp. 45–77, Feb. 2019.
[5] T. Liao, R. Taori, I. D. Raji, and L. Schmidt, "Are we learning yet? A meta review of evaluation failures across machine learning," in *Proc. 35th Conf. Neural Inf. Process. Syst. Datasets Benchmarks Track, Round 2*, 2021, pp. 1–10.
[6] *Reproducibility and Replicability in Science*, National Academies of Sciences, Engineering, Medicine, and others, Nat. Academies Press, Washington, DC, USA, 2019.
[7] P. Holme, "Modern temporal network theory: A colloquium," *Eur. Phys. J. B*, vol. 88, no. 9, pp. 1–30, Sep. 2015.
[8] J. Skarding, B. Gabrys, and K. Musial, "Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey," *IEEE Access*, vol. 9, pp. 79143–79168, 2021.
[9] A. Pareja, "EvolveGCN: Evolving graph convolutional networks for dynamic graphs," in *Proc. AAAI Conf. Artif. Intell.*, 2020, vol. 34, no. 4, pp. 5363–5370.
[10] J. Chen, X. Wang, and X. Xu, "GC-LSTM: Graph convolution embedded LSTM for dynamic network link prediction," *Appl. Intell.*, vol. 52, pp. 7513–7528, Sep. 2021.
[11] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, "Inductive representation learning on temporal graphs," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–19.
[12] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," 2020, *arXiv:2006.10637*.
[13] P. Holme and J. Saramäki, "Temporal networks," *Phys. Rep.*, vol. 519, no. 3, pp. 97–125, 2012.
[14] J. Zhang, "A survey on streaming algorithms for massive graphs," in *Managing and Mining Graph Data*. Boston, MA, USA: Springer, 2010, pp. 393–420.
[15] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.
[16] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, Jan. 2020.
[17] Y. Ma, Z. Guo, Z. Ren, J. Tang, and D. Yin, "Streaming graph neural networks," in *Proc. 43rd Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.* New York, NY, USA: ACM, Jul. 2020, pp. 719–728.
[18] S. Kumar, X. Zhang, and J. Leskovec, "Predicting dynamic embedding trajectory in temporal interaction networks," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 1269–1278.
[19] I. M. Baytas, C. Xiao, X. Zhang, F. Wang, A. K. Jain, and J. Zhou, "Patient subtyping via time-aware LSTM networks," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Halifax, NS, Canada, Aug. 2017, pp. 65–74.
[20] R. Trivedi, H. Dai, Y. Wang, and L. Song, "Know-evolve: Deep temporal reasoning for dynamic knowledge graphs," 2017, *arXiv:1705.05742*.
[21] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "DyRep: Learning representations over dynamic graphs," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–25.
[22] B. Knyazev, C. Augusta, and G. W. Taylor, "Learning temporal attention in dynamic graphs with bilinear interactions," *PLoS ONE*, vol. 16, no. 3, Mar. 2021, Art. no. e0247936.
[23] S. M. Kazemi, R. Goel, S. Eghbali, J. Ramanan, J. Sahota, S. Thakur, S. Wu, C. Smyth, P. Poupart, and M. Brubaker, "Time2Vec: Learning a vector representation of time," 2019, *arXiv:1907.05321*.
[24] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *J. Amer. Soc. Inf. Sci. Technol.*, vol. 58, no. 7, pp. 1019–1031, 2007.
[25] V. Martínez, F. Berzal, and J.-C. Cubero, "A survey of link prediction in complex networks," *ACM Comput. Surv.*, vol. 49, no. 4, pp. 69:1–69:33, Dec. 2016.
[26] A. Wahid-Ul-Ashraf, M. Budka, and K. Musial-Gabrys, "Newton's gravitational law for link prediction in social networks," in *Proc. Int. Conf. Complex Netw. Appl.* Cham, Switzerland: Springer, 2017, pp. 93–104.
[27] I. Ahmad, M. U. Akhtar, S. Noor, and A. Shahnaz, "Missing link prediction using common neighbor and centrality based parameterized algorithm," *Sci. Rep.*, vol. 10, no. 1, pp. 1–9, Dec. 2020.
[28] S. M. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, and P. Poupart, "Representation learning for dynamic graphs: A survey," *J. Mach. Learn. Res.*, vol. 21, no. 70, pp. 1–73, 2020.
[29] "The difficulty of a fair comparison," *Nature Methods*, vol. 12, p. 273, Apr. 2015.
[30] K. Greff, R. K. Srivastava, J. Koutnìk, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space Odyssey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017.
[31] N. T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, Apr. 2017, pp. 1–14.
[32] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–12.
[33] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2016, pp. 3844–3852.
[34] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," in *Advances in Neural Information Processing Systems* (Lecture Notes in Computer Science), L. Cheng, A. C. S. Leung, S. Ozawa, Eds. Cham, Switzerland: Springer, 2018, pp. 362–373.
[35] L. A. Adamic and E. Adar, "Friends and neighbors on the web," *Soc. Netw.*, vol. 25, no. 3, pp. 211–230, 2003.
[36] M. Fey and E. J. Lenssen, "Fast graph representation learning with PyTorch geometric," in *Proc. ICLR*, 2019, pp. 1–9.
[37] B. Rozemberczki, P. Scherer, Y. He, G. Panagopoulos, A. Riedel, M. Astefanoaei, O. Kiss, F. Beres, G. López, N. Collignon, and R. Sarkar, "PyTorch geometric temporal: Spatiotemporal signal processing with neural machine learning models," 2021, *arXiv:2104.07788*.
[38] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 5165–5175.
[39] M. Zhang, P. Li, Y. Xia, K. Wang, and L. Jin, "Labeling trick: A theory of using graph neural networks for multi-node representation learning," 2020, *arXiv:2010.16103*.
[40] Y. Yang, R. N. Lichtenwalter, and N. V. Chawla, "Evaluating link prediction methods," *Knowl. Inf. Syst.*, vol. 45, no. 3, pp. 751–782, Dec. 2015.
[41] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org
[42] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific containers for mobility of compute," *PLoS ONE*, vol. 12, no. 5, May 2017, Art. no. e0177459.
[43] Q. Huang, H. He, A. Singh, S.-N. Lim, and A. R. Benson, "Combining label propagation and simple models out-performs graph neural networks," 2020, *arXiv:2010.13993*.
[44] W. Su, Y. Yuan, and M. Zhu, "A relationship between the average precision and the area under the ROC curve," in *Proc. Int. Conf. Theory Inf. Retr.*, Sep. 2015, pp. 349–352.
[45] F. Gao, K. Musial, C. Cooper, and S. Tsoka, "Link prediction methods and their accuracy for different social networks and network metrics," *Sci. Program.*, vol. 2015, pp. 1–14, Jun. 2015.
[46] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang, "DySAT: Deep neural representation learning on dynamic graphs via self-attention networks," in *Proc. 13th Int. Conf. Web Search Data Mining*. New York, NY, USA: ACM, Jan. 2020, pp. 519–527.
[47] J. Chen, J. Zhang, X. Xu, C. Fu, D. Zhang, Q. Zhang, and Q. Xuan, "E-LSTM-D: A deep learning framework for dynamic network link prediction," 2019, *arXiv:1902.08329*.
[48] K. Lei, M. Qin, B. Bai, G. Zhang, and M. Yang, "GCN-GAN: A non-linear temporal link prediction model for weighted dynamic networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2019, pp. 388–396.
[49] Y. Xiong, Y. Zhang, H. Fu, W. Wang, Y. Zhu, and S. Y. Philip, "DynGraphGAN: Dynamic graph embedding via generative adversarial networks," in *Proc. Int. Conf. Database Syst. Adv. Appl.* Cham, Switzerland: Springer, 2019, pp. 536–552.