Link Prediction on the Patent Citation Network

Samuel Chen
David Dang
Robert Macy
Chris Rockwell

ABSTRACT

By utilizing increasingly accurate graph embedding methods, link prediction has been demonstratively successful on a growing number of datasets, many of which can serve clear utility. However, link prediction is less well studied on large, sparse, and temporal settings. In this paper, we apply link prediction for the first time to the Patent Citation Network: a large, temporal, sparse graph. Experiments demonstrate weaknesses of a neural network based method, Structural Deep Network Embedding, in contrast with surprisingly good performance of node2vec. By comparing with link prediction results on a more typical dataset, BlogCatalog, we highlight the scalability of the PyTorch BigGraph method, demonstrate difficulties resulting from sparsity, and provide evidence that link prediction can generalize to a temporal setting.

1 INTRODUCTION

The primary focus of this project is on applying large scale graph mining techniques on the Patent Citation dataset [6]. Due to limitations of this dataset and for comparison, we also utilize the BlogCatalog dataset [19]. We focus on Link Prediction (LP), which can be broadly defined as the prediction of edges between nodes in a graph. In the case of the Patent Citation dataset, this task can be thought of as making recommendations of past patents to cite or explore for an author of a patent in development. In other words, once the author has found some patents from which to build, our system will be able to provide other patents of interest for the current project.

Formally, LP takes as input a partially observed graph $G \in \{0, 1, ?\}^{n \times n}$, where 0 denotes a known absent link, 1 denotes a known present link, and ? denotes an unknown status link [13]. The goal is to make predictions on the ? entries.

The task of link prediction is closely related to the task of *representation learning* [5]. Link prediction can be made between nodes that have "similar" embeddings by first embedding the nodes in a lower-dimensional feature space. This is illustrated in Fig. 1.

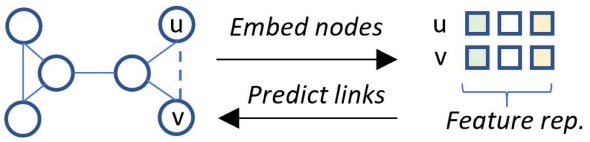

Figure 1: Link prediction via representation learning

Although our graph is directed, we perform this task with the assumption that it is undirected, similar to past work in [13, 18]. In the patent citation network, the direction of links is trivial given that patents will always cite older patents. At test time, the direction of edges can be inferred considering edges it is trained upon and predict can only link with nodes in one direction: backward in time.

2 DATASET

2.1 Blog-Catalog

The BlogCatalog dataset [19] was introduced to first validate performance of the proposed SDNE method. Results of SDNE on the BlogCatalog dataset were reported by Wang *et al.* [18], so the dataset is a useful benchmark of the SDNE implementation. The exact details of the dataset are not important, but the dataset itself contains 10,312 nodes and 333,983 edges in an undirected graph. Ground truth classifications are also available to aid in visualization of the embedding results.

The degree distribution for the Blog Catalog graph is shown in Fig. 2. Note the y-axis is in log base 2. There is a wide range of node degrees, from 1 to $2^{12} \approx 4000$. The overall node-to-edge ratio indicates that this is a rather *dense* graph with a moderate number of nodes, and is therefore an excellent test case for link prediction methods.

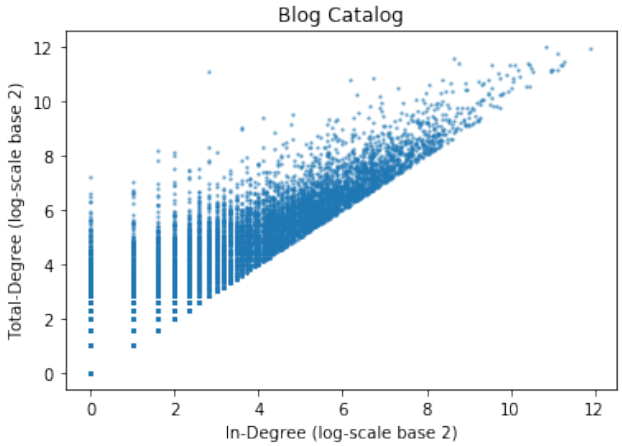

Figure 2: Blog Catalog dataset degree distribution

2.2 Patent Citation Network

The Patent Citation dataset consists of all utility patents granted from 1963 to 1999. It represents these patents in a directed, temporal graph. In addition, the dataset contains feature information for each patent filed after 1974 [6].

The citation network can be abstractly represented as a directed acyclic graph (DAG), $G = (V, E)$, where each vertex, $u, v \in V$, represents an individual patent, and each directed edge, $(u, v) \in E$

parameterizes the connectivity between two patents. It is described in greater detail by Hall *et al.* [6]. The network is large, containing 2,923,922 patents, which correspond to nodes, and 16,522,438 citations, which correspond to edges.

For context, link prediction tasks are typically performed on datasets with significantly fewer nodes. SDNE [18], one of the methods studied, reports LP on Blog-Catalog [19] and ARXIV GR-QC [7] datasets, both of which have under 11k nodes. On the other hand, using more than 500K nodes is typically reserved for papers focused on a very scalable method, such as Pytorch BigGraph [11]. To make node embedding and link prediction tractable for all methods studied, we focus on a small subgraph of the dataset.

Our first subgraph takes all nodes and edges from within 1988-1989 (inclusive). This subgraph consists of approximately 40K nodes and 30K edges. Sparsity is inevitable within any temporally contiguous extraction of the full network. This subgraph is, however, the densest for a contiguous 2 year period of time.

The sparsity can be seen in Fig. 3 (note the non-log scale), where we observe much smaller in-degree. The most common in-degrees observed are 0 and 1.

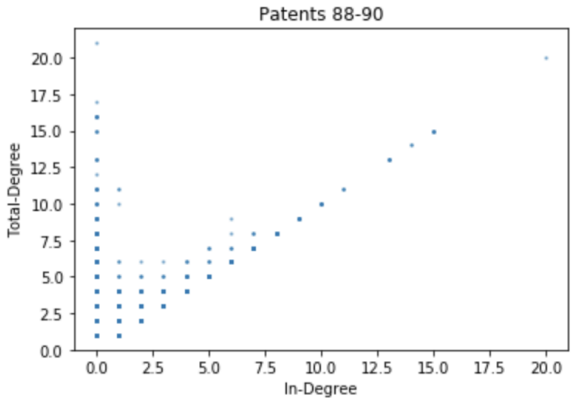

Figure 3: 1988-1989 Patent citation subgraph in-degree distribution

Our second subgraph consists of all nodes and edges from within 1990-1996 (inclusive). This subgraph consists of approximately 580K nodes and 1.2M edges. Although this is slightly more dense, it still is very sparse compared to the entire dataset and its large size proves challenging for most methods.

2.3 Data Partitioning

Two tools were developed to preprocess the patent citation network. The first tool extracts a subgraph based on lower and upper bounded years. This subgraph is effectively the "test" set and used to assess the predictive performance of the methods that are tested. To generate the training dataset, 15% of the links are randomly removed (i.e., hidden).

The second tool preprocesses the node ids (patent numbers) and map them to $\{0, 1, 2, \ldots, N-1\}$ where N is the number of patents. This was necessary because we noticed that SDNE allocates memory corresponding to the max node id rather than the number of unique node ids. This was problematic given that the original data consisted of node ids starting at approximately three million, resulting in unnecessary memory allocation.

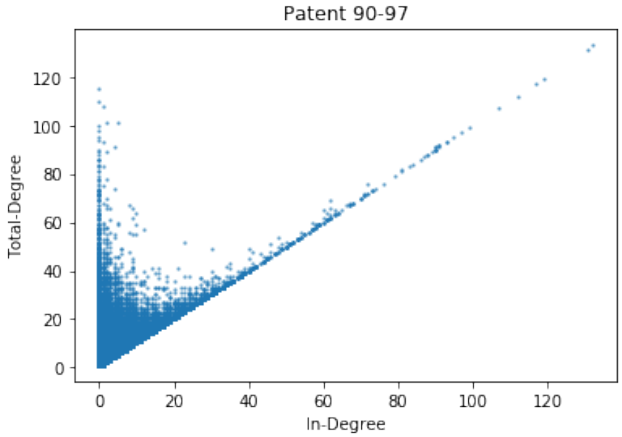

Figure 4: 1990-1997 Patent citation subgraph in-degree distribution

3 PROPOSED METHODS

Several methods for link prediction are evaluated as part of this work, which are described in the following sections.

3.1 Graph Convolutional Network Method: Structural Deep Network Embedding

3.1.1 Representation Learning. SDNE is considered as it has been shown to be effective in dealing with sparse networks and in capturing significant non-linearity in the graph structure, as well as both first and second order proximity.

SDNE consists of a two step process consisting of supervised learning steps (first order proximity), and unsupervised learning (second order proximity). First order proximity is defined as whether or not there is an edge between two nodes in the graph. This step uses laplacian eigenmaps that penalizes more distant mappings of similar vertices. It is thus able to capture pairwise similarities, i.e., common neighbors. Second order proximity captures global structure or role in network communities. This part is performed using a deep autoencoder that embeds the graph nodes into a low dimensional latent space that simultaneously captures the nonlinearity in the graph structure. It can be thought of as mimicking a graph convolutional network [8]. Together, the two paradigms capture the explicit and implicit similarities between nodes.

SDNE uses a a neural network framework that is implemented with TensorFlow using user-defined hyperparameters. The hyperparameters here consist of the number of hidden layers, number of nodes per hidden layer, the output layer dimension, and weights of first and second order proximity losses. The input layer dimension is always equal to the number of nodes in the respective graph. The output layer dimension is equal to the desired embedding size, which is referred to as the low dimensional latent space.

A sparse adjacency matrix is first constructed from a list of input graph edges, after pre-processing has been performed with the previously mentioned partitioning tools on the raw dataset(s). Training is then performed using the neural network, and weights of each layer are optimized using mini-batch gradient descent. The batch size and the learning rate are also user-defined.

3.1.2 Link Prediction. Following [18], 15% of the edges were randomly hidden in the pre-processing step to form the *training* graph. The original graph containing 100% of the edges forms the *test* graph, which is the ground truth used to assess the accuracy of the embeddings.

After training, both reconstruction and link prediction are assessed within the SDNE framework. In the SDNE implementation, the link prediction utilizes the previously learned embeddings through a simple notion of distance, i.e. nodes that have similar embeddings are likely to have a mutually shared link between each other. To predict, all nodes are dotted with all other nodes, and pairs of nodes are ranked by similarity. Edges are then predicted between the most similar nodes pairs.

Note link prediction only considers the prediction accuracy on edges hidden from the training set. This score can then be thought of as how well the embedding represents the entire graph, as simply memorizing the training graph is not enough to produce results.

3.1.3 Future Link Prediction on Patent Citation Network. In reality, we want to predict patents of interest for citation to new patents in the case of the Patent Citation Network. These patents, of course, do not have future patents citing them to give us information. The link prediction task set aside above, and used in non-temporal graphs previously [18], is therefore not directly relevant.

To test prediction of future links, we create a special 500 node set from directly after the remainder of the training set. However, these nodes do not have citations coming from the future. To avoid embedding nodes based on zero edges, we consider only nodes with at least two backward looking citations. The remainder of the testing is exactly the same as link prediction above, with the caveat that embeddings of the entire graph are only compared with these 500 nodes to predict links. Note we do not test future link prediction on BlogCatalog as it is not a temporal set.

3.1.4 Reconstruction. Within the SDNE framework, graph reconstruction is also evaluated. Reconstruction evaluates how well the embedding is learned on the *training* graph, with 15% of edges omitted.

In constrast to link prediction, reconstruction measures how well the learned embeddings can reconstruct the training graph edges (which is not the ground truth). This task is easier given memorization is a viable strategy. Although reconstruction is not directly applicable in our theorized task of recommending patents to cite, it provides a way to evaluate performance of different methods that gives information not represented in link prediction. On the hardest of tasks, when link prediction scores are very low, it can also give an easier benchmark.

3.1.5 Implementation. The specific implementation for SDNE in this work is based off https://github.com/suanrong/SDNE. The code was modified for scalability.

3.2 Random Walk Based Method: node2vec

3.2.1 Representation Learning. An alternative method to generate the node embeddings is node2vec [5]. In this algorithm, node embeddings are generated via 2nd-order random walks through the graph. These random walks may be biased using the return parameter p and the in-out parameter q. By tuning these parameters, node2vec has been shown to capture either homophily or structural equivalence. This is akin to a breadth-first search (BFS) or depth-first search (DFS) approach. For this work, unbiased walks are utilized ($p = q$). Stochastic gradient descent (SGD) is used to optimize the loss function:

$$\max_{f} \sum_{u \in V} \Big[-\log \Big(\sum_{v \in V} \exp(f(n_i) \cdot f(u)) \Big) + \sum_{n_i \in N_S(u)} f(n_i) \cdot f(u) \Big] \tag{1}$$

Since this is computationally expensive to evaluate across all nodes, negative sampling is utilized. N_S is the node set across the random walk strategy S.

3.2.2 Link Prediction. In this method, link prediction is performed similarly as in SDNE. For node2vec, the generated node embeddings are evaluated pair-wise using the binary Hadamard operator to determine if a link exists between two nodes. This is only performed over the subset of *missing* edges.

3.2.3 Reconstruction. The reconstruction task is similar to the link prediction task, except that it is evaluated over the *training* edge set. No a priori edges are assumed, but the learned embeddings are used to reconstruct the original graph by determining if an edge exists between two nodes *solely* from their embeddings. This reconstructed graph is compared to the original training graph for accuracy.

3.2.4 Implementation. The specific implementation for node2vec in this work is based off https://github.com/lucashu1/link-prediction. Routines to compute precision@K for link prediction and reconstruction tasks were implemented on top of the existing code base.

3.3 Non Neural Network Based Method: Jaccard Coefficients

One of the methods we use as a baseline approach for link prediction is the Jaccard coefficient. This value is a measure of similarity between two nodes based on their direct neighbors, and is calculated as the intersection of their neighbor sets over their union, as follows:

$$JC(X, Y) = \frac{N_X \cap N_Y}{N_X \cup N_Y}$$

Where X and Y are nodes, and N_X, N_Y are their neighbor sets. The larger this value, the higher our confidence that the two nodes share an edge. Note that with this approach, no node embeddings are computed or learned. While this method is very simple and intuitively reasonable, it is also rigid, and our assumptions about its value seem less reasonable under certain conditions. For example, in a temporal graph like the patent dataset, if the time span over which we sample from the whole dataset is short, then we may have many situations where two patents filed in the same year cite a similar set of prior work, but do not share a citation between

them, so having a high Jaccard coefficient may not be as effective a predictor as in other graph settings. This method is also related to SDNE in the sense that it is precisely a measure of second-order proximity, how similar the neighborhoods of two nodes are.

3.3.1 Implementation. The specific implementation for Jaccard in this work is based off tools in the Python networkx library, implementation of all metrics were done by us for evaluation.

3.4 PyTorch BigGraph

In addition to the node embedding methods discussed already, we also considered a method from a recent work by Lerer et al [11] called PyTorch-Biggraph, which is a tool developed in PyTorch for learning node embeddings for web-scale graphs. The method used in this work differs from that of node2vec in that is does not use a random walk sampling approach, but rather learns an embedding that attempts to maximize a similarity function for nodes that share an edge while minimizing the score between nodes not sharing an edge. This can be done by sampling edges from the graph and creating negative samples (edges that don't exist in the graph). Then we can train a model that maximizes the following loss function

$$\mathcal{L} = \sum_{e \in G} \sum_{e' \in S'_e} \max(f(e) - f(e') + \lambda, 0)$$

where λ is a margin hyperparameter and

$$S'_e = \{(s', r, d) | s' \in V\} \cup \{(s, r, d') | d' \in V\}$$

S'_e is the set of all possible mutated edges where only the source or destination are changed, and r is a relation between the two nodes. In our setting, there is only 1 relation, which is that the source cites the destination node. Each training pair is a positive edge e and a mutated edge e' with either the source or the destination (but not both) mutated to create a negative sample. This has the effect of making nodes that share an edge close together in embedding space and nodes which do not share an edge are farther apart in embedding space. PyTorch-Biggraph also uses other techniques to increase scalability such as partitioning the graph into different components and performing training in each component. This introduces the potential of corrupting the training data, however, in the author's experiments and ours, this does not lead to poor results. The corruption occurs because when determining negative samples by mutating existing edges, we can not check the whole graph to guarantee that a mutated edge does not exist in the original graph.

3.4.1 Implementation. The specific implementation for PyTorch BigGraph in this work is based off https://github.com/facebookresearch/PyTorch-BigGraph.

4 EXPERIMENTS

LP can be formulated as a binary classification problem to answer the underlying question: does there exist an edge, $e \in E$, between two nodes $\in V$. For the patent dataset, the edges will be divided into three partitions: (1) training set (2) validation set and (3) test set.

We use a similar experimental setup as previous work such as SDNE [18] and DeepWalk [14] where a random selection of edges in the patent dataset will be hidden and partitioned into validation and test sets. The remaining edges will be used to train a network embedding model [18]. Note that the nodes themselves are not partitioned in this process.

4.1 Experimental Setup

The hyperparameters used for the Blog-catalog case are shown in Table 1. This follows the configuration reported in [18]. Running on 4 CPUs, total training runtime was approximately 4 hours.

For the Patent dataset, the hyperparameters were chosen largely out of memory and computational constraints.

Hyperparameter	Blog Catalog	Patent
Number of Hidden Layers	1	1
Nodes in hidden layer(s)	1000	400
Network embedding size	100	40
Learning rate	0.01	0.01
Batch size	32	16

Table 1: SDNE Hyperparameters for Blog Catalog and Patent Citation networks

Likewise, the parameters for the node2vec method are shown in Table 2 for both the Blog Catalog and Patent Citation graphs.

Parameter	Blog Catalog	Patent
Embedding Dimension	128	128
Return p	1	1
In-out q	1	1
Walk Length	80	80
No. Walks per source	10	10

Table 2: node2vec parameters for Blog Catalog and Patent Citation networks

4.2 Evaluation Metrics

Following previous works using embedding methods for LP [18], we evaluate our performance using the *precision@k* metric. Precision gives us the percentage of true positive predictions from our model, which is relevant for LP since we are only directly predicting which links should exist in LP and only implicitly which should not exist. Precision at k is a useful metric often used in information retrieval for identifying the number of relevant results in top k ranked predictions. In graph mining it can be used for precision in top k ranked by confidence predicted links. The calculation for *precision@k* is as follows:

$$precision@k(i) = \frac{|\{j | i, j \in V, index(j) \leq k, \Delta_i(j) = 1\}|}{k}$$

where V is the vertex set, $index(j)$ is the ranked index of the j-th vertex and $\Delta_i(j) = 1$ indicates that v_i and v_j have a link.[18]

Recall *precision@k* for link prediction is only computed on edges predicted not in the training set. This means an edge predicted that exists in the training set is simply skipped over, without count nor the correct edges increasing.

4.3 Blog-catalog

Table 3 shows the Precision@K metric evaluated at various k values for reconstruction (relative to the sparsified training graph) and link prediction (relative to the ground truth). The reconstruction results match the reported results of [18] quite well, and show good reproducibility. The exact batch size and learning used in [18] were not reported, so minor differences are expected. Note that link prediction was only done up to $k = 1000$, and demonstrates 50-60% accuracy in link prediction over the randomly omitted nodes. Scores are comparable at $K < 10000$ to what was reported in the SDNE paper for reconstruction.

Precision@K	Reconstruction	Link Prediction
10	1.00	0.60
50	0.85	0.59
100	0.80	0.56
500	0.77	0.51
1000	0.76	0.49

Table 3: SDNE precision@K results for reconstruction and link prediction on Blog Catalog

Table 4 shows the reconstruction and link prediction results using node2vec. Note that link prediction was not reported in that paper. The reconstruction scores are *lower* than SDNE, but link prediction scores are *higher*. This could be due to the dense Blog Catalog graph being insufficiently explored with the random walk approach.

Precision@K	Reconstruction	Link Prediction
10	0.700	0.800
50	0.660	0.920
100	0.790	0.810
500	0.794	0.806
1000	0.796	0.796

Table 4: node2vec precision@K results for reconstruction and link prediction on Blog Catalog

Due to the computational complexity of link prediction with Jaccard coefficients, we do not report those experiments for Blog-Catalog as it is much larger than our patent subgraphs.

PyTorch BigGraph does not perform the reconstruction task, so scores are not reported. Figure 5 below shows the comparison of link prediction for all three methods evaluated on the Blog Catalog graph.

4.4 Patent Citation Network

Due to the current limited computing resources, we have resorted to using a very small portion of the patent citation network to assess the performance of SDNE. The preliminary results in this section uses the sub-graph that consist of patent edges between 1988 and 1989. This subgraph consists of 40,770 nodes and 30,206 edges, containing all patents from 1988 through 1989 *that cite only each other*. The training was performed using SDNE.

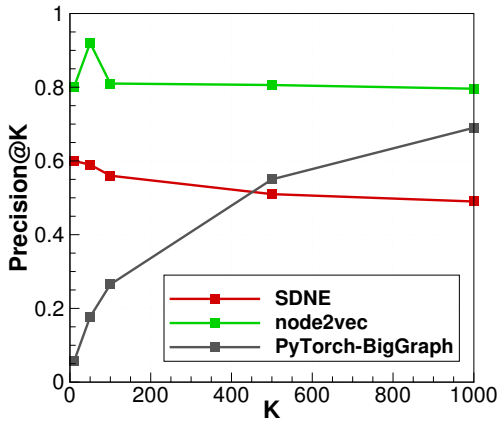

Figure 5: Blog Catalog link prediction results

4.4.1 Baseline: No Future Nodes. In this study, the size of the hidden layer and network embedding is limited due to memory and computational bottlenecks. Running on single GPU unit, the training runtime took upwards of 20 hours even on the reduced subgraph.

The results of the reconstruction and link prediction for the Patent citation subgraph are shown in Table 5. It is apparent that the SDNE method performs quite poorly on this subgraph in both the reconstruction and the link prediction tasks. It is likely pairwise similarities is not very helpful in this task, but interesting to note even when changing weights to consider global structure more the result remains low. Here unsupervised loss is weighted 100:1 ratio to supervised.

Precision@K	Reconstruction	Link Prediction
10	0.200	0.000
50	0.080	0.000
100	0.060	0.000
500	0.074	0.004
1000	0.061	0.006

Table 5: SDNE precision@K results for reconstruction and link prediction on *baseline* Patent Citation subgraph.

Performance in general is weak likely because of the sparse nature of the subgraph, having a edge-to-node ratio of < 1. This implies that many nodes have have only one edge connecting to it, and the graph may contain nodes with mutual edges that are completely disconnected from other nodes. Physically, this is an inherent feature of the patent citation dataset (patent “lag" time [6]), where patents are not likely to cite other patents that were filed within the same year or two, instead citing much older patents.

The reconstruction scores in Table 6 indicate that node2vec is surprisingly effective at the reconstruction task, even for extremely sparse graphs. Likewise, link prediction scores are $> 10\%$, significantly higher that SDNE. The results for reconstruction and link prediction using Jaccard coefficients are presented in Table 7, but show much poorer performance in both the reconstruction and link prediction tasks.

Precision@K	Reconstruction	Link Prediction
10	0.800	0.100
50	0.860	0.120
100	0.840	0.120
500	0.818	0.118
1000	0.815	0.127

Table 6: node2vec precision@K results for reconstruction and link prediction on *baseline* Patent Citation subgraph

Precision@K	Reconstruction	Link Prediction
10	0.000	0.000
50	0.000	0.000
100	0.069	0.069
500	0.032	0.032
1000	0.023	0.019

Table 7: Jaccard precision@K results for reconstruction and link prediction on *baseline* Patent Citation subgraph

We can see from Figure 6 that PyTorch-Biggraph performs much better than Jaccard and SDNE on both experiments in the patent dataset, however, not as good at node2vec. One thing to note in all our experiments with Pytorch-Biggraph is that performance increases as we increase k. This result is in line with the results presented on other datasets in the original paper for PyTorch-Biggraph [10]. One hypothesis for why we see the upward trend in performance has to do with the fact that the implemented method maximizes similarity based only on whether two nodes share an edge. In graphs like the patent subgraph we consider, it is more likely a node will be closer to another node that cites the same patents, rather than one where an edge has been removed. However, this would not explain the upward trend in BlogCatalog and the FreeBase dataset [10].

Figure 6 shows the comparison of link prediction among the four methods. Overall, the link prediction scores are much lower for the Patent citation graph than the Blog Catalog. We observe that node2vec performs the best, with PyTorch a close seecond. As discussed, both SDNE and Jaccard seem to perform very poorly on this sparse graph.

4.4.2 Future Nodes. This set focuses on prediction of links to the 500 *future* nodes from their embeddings, rather than the missing 15% links that were intentionally omitted during training. These nodes were selected to have at least two backward-looking edges to make link prediction meaningful. Forward-looking edges were removed for realism, as addressed earlier.

Reconstruction was only evaluated with the node2vec and Jaccard methods. Both reconstruction and link prediction scores for node2vec are tabulated in Table 8. Link prediction scores for node2vec here are marginally higher than for the baseline case, likely due to the increased edge density associated with the 500 future nodes, and reconstruction performs similarly. Likewise, the results for Jaccard are presented in Table 9, and show similarly poor performance as in the previous case. Scores are non-zero at higher K values, but never exceed 7%.

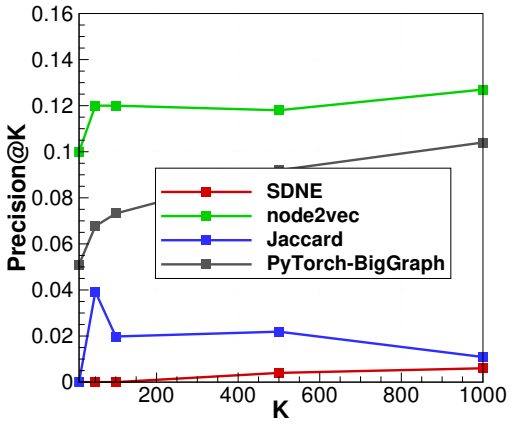

Figure 6: Baseline 1988-1989 patent link prediction results

Precision@K	Reconstruction	Link Prediction
10	0.600	0.400
50	0.840	0.160
100	0.840	0.140
500	0.836	0.122
1000	0.826	0.135

Table 8: node2vec precision@K results for reconstruction and link prediction on *future* Patent Citation subgraph

Precision@K	Reconstruction	Link Prediction
10	0.000	0.000
50	0.000	0.000
100	0.069	0.069
500	0.034	0.0034
1000	0.023	0.019

Table 9: Jaccard precision@K results for reconstruction and link prediction on *future* Patent Citation subgraph

As in the patent baseline subgraph, Pytorch-Biggraph shows an upward trend in performance for the future subgraph, and performs below node2vec.

The link prediction data is summarized for all methods on the 1988-1989 Patent subgraph in Fig. 7. It is difficult to discern a clear trend here, although PyTorch's score increases with K.

4.5 Discussion

Surprisingly, node2vec demonstrates superior performance in all three cases examined in this work, especially for the extremely sparse 1988-1989 Patent citation subgraph. This is likely because the random walk approach respects the structure of the graph itself. If no restarts are allowed, and disconnected components of the graph are treated as such, starting from the seed node. There is also a lot more noise in the results from node2vec, largely depending on the starting seed locations.

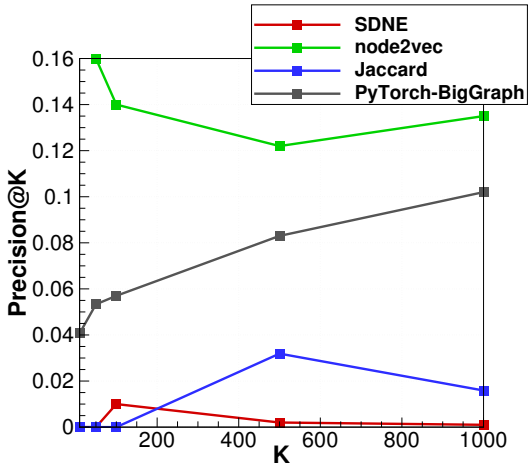

Figure 7: Future 1988-1989 patent link prediction results

Because the largest difficulty faced in our 1988-1989 dataset was sparsity, we hypothesized that partitions with higher densities could improve results. For instance, the subgraph from 1990 through 1996 contains 577,669 nodes and 1,226,658 edges, which is more than twice as dense as 1988-1989. However, for many methods this is too large to get good results, or even results at all. We do not report detailed results on this subgraph as methods were either unable to run or had such poor results.

4.5.1 Computational Cost Comparison. In the SDNE paper, the algorithm seemed to need larger and deeper network layers to perform well with increasing graph size. However, on this large of a graph it was necessary to have smaller network layers to fit within memory, hurting performance significantly. On the larger subgraph, SDNE had link prediction precision@K of 0.000 for all k tested.

Node2vec is able to run economically for the Blog Catalog and 1988-1989 Patent subgraphs. This is largely a function of the use of random walks, SGD, and negative sampling. However, node2vec runs into similar issues as SDNE for the larger Patent citation subgraph (1990-1996), exceeding normal memory allocations (>32 GB).

The Jaccard Coefficient has the worst asymptotic complexity of any method we considered. In order to predict the existance of a link between two nodes, we need to compute both of their neighborhoods and use them for the Jaccard coefficient calculation. Since we do not have an efficient method of determining which nodes to compare as we do with similarity for graph embeddings, we need to calculate the coefficient for all pairs, which is $O(n^2)$.

As may be expected based on its name, PyTorch BigGraph was found to have the most promise in scaling up to the larger graph size. Because of the graph partitioning methods and approximate negative sampling discussed in the approaches section, it is able to handle larger dataset, and further work would try applying it to the entire patent dataset.

5 RELATED WORK

5.1 Related Work on the Dataset

There have been numerous studies utilizing the patent citation dataset, although most of the work do not fall into the realm of data mining, and are not strictly relevant in the context of this project. One such study that is relevant is in [6]. In this, the authors describe an interesting “lag" effect between *citing* and *cited* patents. This issue likely contributes to the sparsity of subgraphs and makes link prediction difficult.

Graph mining techniques have also been applied on the dataset. In [15], community detection was performed on the patent citation dataset via spectral clustering techniques. The authors showed the singular vectors of the graph exhibit a a distinct “EigenSpokes" structure, characteristic of large, sparse social graphs. Graph mining techniques were applied in [16] primarily for the purposes of link analysis and outlier detection. The author effectively showed which patents were “most influential" within the network.

Community detection has also been roughly extended to the idea of LP. In [2], the authors first utilized hierarchical clustering techniques to form a dendrogram representation of the network by explicitly computing the “distance" between citation vectors. Then, structural changes can be identified by comparing the dendrogram at discrete instances in time. Semi-supervised classification has been demonstrated on the patent citation dataset, and is much more scalable approach to classification, using a random-walk with restarts (RWR) approach [12] to approximate this “betweenness."

5.2 Related Work on LP

A topical search of LP in the recent literature reveals several promising approaches, none of which have been performed on the patent citation dataset to our knowledge. Earlier works on LP relied on matrix factorizations techniques [13] and stochastic gradient descent optimization for scalability. This approach also took into account latent and side information about each node, which improved the prediction performance. More generally, LP in relational data has also been performed using *probabilistic* models [17]. Here, the authors model the graph as a Markov network, capturing both the link structure and attributes of each node.

Within the neural network framework, several distinct approaches have been demonstrated with applications to LP. Unsupervised learning of graph representations, such as with Deepwalk [14], is an important component in first learning the structure and classification of such networks. Variational graph autoencoders (VGAE) have been used for LP, with especially promising results [9]. A convolutional neural network is used to “encode" the node features, and nodes attributes/metadata can be incorporated into the encoding. Decoding is performed with a inner product of the representation vectors. Graph convolutional networks [8] use an approach similar to our unsupervised method. We note these neural network methods should be expected to struggle similarly to SDNE on the larger, sparser graph.

There are additional approaches used for LP in the literature, such as evolutionary algorithms [1], but are not discussed in detail here. LP in directed graphs has also been investigated in a few cases [4]. In [3], the researchers discuss how similarity features commonly used to represent undirected graphs (e.g. Katz, Jaccard)

can be extended to *directed* graphs. However, this does not apply to *learned* features, and requires additional topological features to be computed for directed graphs.

6 CONCLUSIONS AND FUTURE WORK

Performance of all methods dropped significantly from BlogCatalog to the more sparse Patent Citation Dataset. Node2vec and PyTorch-BigGraph were more robust to the change, while SDNE struggled on the 1988-1989 partition.

Methods also had difficulty scaling to the larger partition of the dataset. Node2vec was not able to run on the 1990-1996 partition, while SDNE registered link prediction scores of essentially 0.

Methods showed a strong ability to generalize temporally. When comparing link prediction results within 1988-1989 to those on future nodes, most methods showed similar performance while node2vec actually outperformed on future nodes. This is an encouraging result for link prediction to be used on temporal graphs.

Despite GPU access and background research, scaling algorithms was difficult both in setup and computation. A main lesson from the project has been to very carefully select algorithms and computing systems before trying larger-scale graph mining techniques. Sampling and other techniques can be very helpful in a task such as link prediction.

When considering the Patent dataset specifically, link prediction on subgraphs saw lower results than we hoped. Partitioning entire subgraphs from time periods results in a very small node to edge ratio. Rather than extracting subgraphs based on years, future work could explore extracting subgraphs based on communities. For this, community detection could be performed, and subgraphs could be made up of one or multiple communities. This is posited to greatly increase the node to edge ratio, improving link prediction performance.

A focus of this work was to provide insight into the predictive comparative performance of the algorithms tested. However, much of this work was limited by computational constraints, making optimizing hyperparameters a difficult task. Future work could perform cross-validation on the algorithms to determine optimal hyperparameters, and comparisons could then be made for the algorithms while using their optimized hyperparameters, providing a more accurate measurement of their comparative performance.

7 AUTHORS AND CONTRIBUTIONS

Samuel Chen SDNE on Blog Catalog dataset, tool development, node2vec on all cases and analyzing results

David Dang Responsible for building preprocessing tools, parsing patent network, and building subgraphs

Robert Macy Data analysis and visualization, Jaccard coefficient, PyTorch-Biggraph, evaluation metrics

Chris Rockwell SDNE on Patent dataset, building sampling for large link prediction, "future" formulation and tests

REFERENCES

[1] Catherine A. Bliss, Morgan R. Frank, Christopher M. Danforth, and Peter Sheridan Dodds. 2014. An evolutionary algorithm approach to link prediction in dynamicsocial networks. *J. Comput. Sci.* 5 (2014), 750–764. https://doi.org/10.1016/j.jocs.2014.01.003

[2] Péter Érdi, Kinga Makovi, Zoltán Somogyvári, Katherine Strandburg, Jan Tobochnik, Péter Volf, and László Zalányi. 2013. Prediction of emerging technologies based on analysis of the US patent citation network. *Scientometrics* 95, 1 (2013), 225–242.

[3] Michael Fire, Lena Tenenboim, Ofrit Lesser, Rami Puzis, Lior Rokach, and Yuval Elovici. 2011. Link prediction in social networks using computationally efficient topological features. In *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*. IEEE, 73–80.

[4] Dario Garcia-Gasulla. 2015. *Link Prediction in Large Directed Graphs.* Ph.D. Dissertation. Polytechnic University of Catalonia, Barcelona, Spain.

[5] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.

[6] Bronwyn H. Hall, Adam B. Jaffe, and Manuel Trajtenberg. 2001. *The NBER Patent Citation Data File: Lessons, Insights and Methodological Tools.* Technical Report W-8498. National Bureau of Economic Research, Cambridge, MA.

[7] J. Kleinberg J. Leskovec and C. Faloutsos. 2007. Graph evolution: Densification and shrinking diameters. In *ACM Transactions on Knowledge Discovery from Data (TKDD)*.

[8] Thomas N. Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[9] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).

[10] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. 2019. PyTorch-BigGraph: A Large-scale Graph Embedding System. *arXiv preprint arXiv:1903.12287* (2019).

[11] Wu L. Shen J. Lacroix T. Wehrstedt L. Bose A. Peysakhovich A. Lerer, A. 2019. PyTorch-BigGraph: A Large-scale Graph Embedding System. *arXiv preprint arXiv:1903.12287* (2019).

[12] Amin Mantrach, Nicolas Van Zeebroeck, Pascal Francq, Masashi Shimbo, Hugues Bersini, and Marco Saerens. 2011. Semi-supervised classification and betweenness computation on large, sparse, directed graphs. *Pattern recognition* 44, 6 (2011), 1212–1224.

[13] Aditya Krishna Menon and Charles Elkan. 2011. Link prediction via matrix factorization. In *Joint european conference on machine learning and knowledge discovery in databases*. Springer, 437–452.

[14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[15] B. Aditya Prakash, Ashwin Sridharan, Mukund Seshadri, Sridhar Machiraju, and Christos Faloutsos. 2010. EigenSpokes: Surprising Patterns and Scalable Community Chipping in Large Graphs. In *Advances in Knowledge Discovery and Data Mining*, Mohammed J. Zaki, Jeffrey Xu Yu, B. Ravindran, and Vikram Pudi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 435–448.

[16] Andrew Rodriguez. 2015. *Graph Mining Algorithms for the Analysis of Patent Citation Networks.* Ph.D. Dissertation. Rutgers University, New Brunswick, NJ.

[17] Ben Taskar, Ming-Fai Wong, Pieter Abbeel, and Daphne Koller. 2004. Link prediction in relational data. In *Advances in neural information processing systems*. 659–666.

[18] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1225–1234.

[19] R. Zafarani and H. Liu. 2009. Social Computing Data Repository at ASU. http://socialcomputing.asu.edu