Tab2Graph: Transforming Heterogeneous Tables as Graphs

Rajat Singh
CSE, IIT Delhi
India
rajat.singh@cse.iitd.ac.in

Raajita Bhamidipaty*
CSE, IIT Delhi
India
bhamidipatyraajita81@gmail.com

Anjali Sharma*
CSE, IIT Delhi
India
anjalisha396@gmail.com

Srikanta Bedathur
CSE, IIT Delhi
India
srikanta@cse.iitd.ac.in

ABSTRACT

Tables continue to be one of the most widely used data formats for representing information due to their flexibility and intuitive appeal. Despite this, extracting meaningful insights and gaining a comprehensive understanding from these tables remains challenging. Several factors contribute to this – including the sheer volume of tabular content, the difficulty of their use in modern deep-learning techniques, and more. Significant effort has been put into developing advanced deep-learning models that can learn dense representations for tabular data and utilize these representations to perform various downstream tasks.

In this paper, we investigate various methods for transforming a large entity-centric heterogeneous tabular corpus into a graph. We demonstrate how the accuracy of different deep learning models varies with changes in graph structure for the Missing Value Imputation (MVI) task. We show that by selecting an appropriate table-to-graph transformation, namely a *dual-node heterogeneous graph*, and leveraging powerful graph models that are also scalable, we can achieve better performance in low resource settings as compared to the state-of-the-art attention model while requiring 30 times less training time.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Information systems** → **Graph-based database models**; • **Theory of computation** → **Incomplete, inconsistent, and uncertain databases**.

KEYWORDS

Structured Data, Tabular Data, Graph Neural Network, Missing Value Imputation

*Both authors contributed equally to this research.

AI-ML Systems '24, October 08–11, 2024, Baton Rouge, USA

ACM ISBN 979-8-4007-1161-9…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Rajat Singh, Raajita Bhamidipaty, Anjali Sharma, and Srikanta Bedathur. 2024. Tab2Graph: Transforming Heterogeneous Tables as Graphs. In *Baton Rouge '24: AI-ML Systems, October 08–11, 2024, Baton Rouge, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

With the rapid advancement of internet technology, the volume of data being generated has surged dramatically [1]. This data comes in various modalities: text, image, audio, video, and more. Tables are one of the most commonly used data structures to store this tremendous amount of generated data. They are highly efficient for storing, managing, and retrieving information, making them indispensable in many applications. As a result, the use of tables has become increasingly prevalent in the digital age.

Today, nearly all institutes, companies, and hospitals depend on databases, which consist of various interconnected tables, to store and process their data. The data stored in these databases is crucial for making a wide range of important decisions. Additionally, a massive amount of heterogeneous tabular data resides in data lakes, which can be of great use in various domains. Consequently, a new area of research has emerged dedicated to employing diverse learning techniques on tabular data and performing various downstream tasks.

Despite tables being one of the easiest ways to store data in the form of rows and columns, extracting meaningful insights and gaining a comprehensive understanding from these tables remains challenging. The primary factor contributing to this difficulty is the inherent structure of tables, which consist of permutable rows and columns. Each row in a table represents a specific fact, while all the values within a column pertain to a similar entity. This rigid structure makes it challenging to use tabular data out of the box directly when applying machine learning (ML) techniques. Consequently, it becomes necessary to transform tabular data into other modalities to facilitate the application of ML techniques and retain the intrinsic properties of the tables.

Researchers from various domains have looked at tabular data from different perspectives: viewing tables as trees [5–7, 9, 14, 20, 24, 26, 29], tables as sentences [11, 12, 15, 18, 31, 35, 36], and tables as graphs [3, 8, 10]. Among these, tree representation is one of the most used methods but is primarily limited to classification and regression tasks. On the other hand, transforming tables as linear sentences opens up the possibility of applying Natural Language Processing (NLP) models [2] to these tables. These NLP

models learn latent representations, also known as embeddings, of the table. These embeddings can then be used to perform various downstream tasks [2] on the tables. However, converting tables into linear sentences has its drawbacks. While it expands the range of tasks that can be applied to tables, it also results in the loss of the tables' inherent structural information. Moreover, training these NLP models is a highly time-consuming and resource-intensive process.

Furthermore, transforming tables as graphs opens up the possibility of applying graph-learning models [3, 8, 10] to these tables. Like NLP models, graph-learning models also learn latent representations [16]. Converting tables into graphs offers several advantages over converting them into linear sentences. Graphs are permutation invariant, meaning they can inherently incorporate tables' column and row-invariant properties. This makes them particularly well-suited for preserving the structural information of tabular data. Additionally, graphs are a better choice of representation when dealing with interconnected tables, as they can more efficiently represent the relationships between multiple tables compared to linear sentences.

However, there are challenges associated with the graph itself. If the graph is too large, graph-learning models may struggle with performance; if it is too sparse, they lack sufficient information, both leading to poor embeddings. Overall, while graph representations offer significant benefits for handling tabular data, careful consideration must be given to the size and density of the graph to ensure effective learning and application of graph-based models.

This research aims to address these challenges by proposing better table-to-graph transformation to enhance the performance of graph learning models on entity-centric heterogeneous tabular data. We introduce two efficient techniques for transforming entity-centric heterogeneous tabular data into graphs, intending to improve the effectiveness of various graph learning methods on downstream tabular tasks. We evaluated the effectiveness of our proposed table-to-graph transformation algorithms on the Missing Value Imputation (MVI) task. Our results demonstrate that these optimized graph transformations significantly improve the performance of graph-based learning models, offering a promising solution for the complex challenges of handling tabular data.

The key contributions of this paper include:

- We propose two novel table-to-graph transformation techniques to effectively encode entity-centric heterogeneous tables as a compact graph.
- Further, we propose methods to incorporate edge weight into the transformed graphs to encode entity co-occurrence information.

1.1 Organization

The rest of the paper is organized as follows: in 2 we discuss different ways in which tabular data has been represented for learning its representations. Next, in Section 3, we present the preliminary concepts required in the paper, before outlining our proposed algorithms for representing tabular data as graphs in Section 4. The large-scale WikiTable corpus used in our experiments and the details of our empirical evaluation are discussed in detail in Section 5. Finally, concluding remarks, along with possible future work, are given in Section 6.

2 RELATED WORK

A significant amount of research has been dedicated to identifying and understanding hidden patterns in tabular data, as well as applying these learned patterns to various downstream tasks. This section provides an overview of the research dedicated to the development of this field.

Tabular data has garnered attention from researchers across various domains, leading to diverse approaches to visualizing and interpreting this type of data. Depending on their specific use cases, researchers have visualized tabular data in multiple ways: some have treated it as a tree [5–7, 9, 14, 20, 24, 26, 29], some as a linear sequence [11, 12, 15, 18, 31, 35, 36] similar to a sentence, and others have treated it as a graph [3, 8, 10]. This variety of perspectives highlights the versatility and complexity of tabular data, as well as the innovative methodologies employed to extract meaningful insights from it.

Tabular Data as Tree: One of the initial lines of the work on tabular data was representing them as trees (Decision Tree [6, 24, 29], XGBoost [9], CatBoost [26], LightGBM [20], AdaBoost [14], Random Forest [5, 19], and Oblique Decision Tree [7]) and then performing classification and regression tasks. These models have dominated the field for decades due to their simplicity, interpretability, and strong performance.

All tree-based approaches fundamentally revolve around constructing effective decision trees. These approaches include both single-tree models, like Decision Trees [6, 24, 29], and ensemble methods, such as XGBoost [9], CatBoost [26], LightGBM [20], AdaBoost [14], Random Forests [5, 19], and Oblique Decision Trees [7]. The construction process of these models involves iteratively selecting optimal splits that divide the tabular data into two distinct parts. The quality of these splits is evaluated using various statistical metrics, including weighted Gini impurity [32], information gain [25], and chi-square [27]. By continually refining these splits, tree-based models aim to improve their accuracy and overall performance.

Tree-based methods, while highly trusted and widely used, have several significant drawbacks. Firstly, their applications are primarily limited to regression and classification tasks. As the size of the dataset increases, the depth of the trees can become unmanageable, and the number of trees required for ensemble models can grow exponentially, leading to increased computational complexity. Additionally, the decision boundaries created by these models are hard and inflexible, making them unsuitable for many cases where more nuanced decisions are needed. Lastly, tree-based methods often require extensive feature engineering to achieve optimal performance, which can be time-consuming and labor-intensive.

Tabular Data as Sentence: With the advancement of deep learning models in natural language processing (NLP) research, researchers have begun to view tables as linear sequences of sentences. Considerable effort has been dedicated to determining how to represent tables as sentences while preserving their inherent properties effectively.

One of the pioneering works in this area is Table2Vec [11], which leverages both table data (entities and cell values) and its metadata (captions and column headings) to construct sentences. Table2Vec linearizes the rows of the table and concatenates the metadata to form these sentences, which are then fed into a deep learning model to learn embeddings. Subsequent works, such as SAINT [31], RelBERT [15], and TaPaS [18], also construct sentences by linearizing the rows of the table. The key difference between these methods is their incorporation of the tabular structure to learn embeddings from these linear sentences. TaPaS [18] employs a variety of specialized embeddings to capture the table structure during the embedding learning process. Meanwhile, SAINT [31] applies attention mechanisms to both the rows and columns of the table by using inter-sample attention [33].

Other approaches, such as GraPPa [36], focus on the task of table semantic parsing. GraPPa uses natural language (NL) questions and synthetic text-to-SQL examples, concatenating them with row-wise linearized tables, including column headers, to facilitate understanding of the tabular structure and data. TaBERT [35] takes a different approach by carefully selecting relevant rows from the table to linearize. It incorporates the column names and data types, using a special separator character (vertical bar) to incorporate the tabular structure during linearization.

TURL [12] is the state-of-the-art model for learning embeddings on entity-centric tabular data. It uses the table caption, table header, and entities to linearize the table. TURL uses a variety of specialized embeddings and a visibility matrix to capture the table structure and learn effective table embeddings. Consequently, we used TURL as one of the baselines in this paper.

These innovative methods showcase the potential of transforming tabular data into sentences, leveraging the power of deep learning models to enhance the understanding of tabular data. However, linearizing tables presents several challenges. One major disadvantage is the difficulty preserving the inherent table structure, where entities in a column share similar semantic meanings and individual rows pertain to the same object. Additionally, the permutability of rows and columns – a property crucial to the table's structure – is not captured when tables are linearized. These complexities can hinder the effective representation and integration of the data, ultimately affecting the performance of models that rely on these linearized table transformations.

Tabular Data as Graph: Visualizing tabular data as graphs is another compelling approach. One of the pioneering works in this direction is EmbDI [8], which constructs a tripartite graph from one or more tables. This graph consists of three types of nodes: Entity nodes, Row nodes, and Column nodes, which are connected based on the structural relationships within the tables. Random walks are then performed on this graph to generate linear sentences, which are subsequently fed into a deep neural network to learn embeddings of entities in the tables.

Another approach, ATJ-Net [3], visualizes tables as complex graphs, specifically hypergraphs, rather than simple tripartite graphs. In this hypergraph, vertices represent joinable attributes among the tables, and hyperedges represent tuples of the tables. ATJ-Net uses a Message-Passing Neural Network (MPNN) to process this hypergraph and learn embeddings, which can be used to perform various downstream tasks on tables. On the other hand, Milan Cvitkovic et al. [10] propose visualizing tables as directed multi-graphs. In this representation, rows are treated as nodes, and foreign key references are depicted as directed edges. These directed multi-graphs are passed through a Graph Neural Network (GNN) to learn embeddings.

This highlights the versatility and effectiveness of graph-based visualizations for tabular data, enabling more nuanced and structurally aware learning. However, graph learning models, which rely on the concept of message passing to learn embeddings, face challenges when dealing with huge graphs — those with more than millions of nodes — since this scale becomes a bottleneck. Similarly, overly dense graphs can lead to nodes learning redundant information, resulting in poor-quality embeddings. Therefore, it is crucial to balance the size and density of the graph to effectively apply graph learning models to tabular data.

Note: Throughout the rest of this paper, we will focus exclusively on entity-centric heterogeneous tables. Therefore, any mention of "tables" will refer specifically to entity-centric heterogeneous tables.

3 PRELIMINARIES

This section provides a concise overview of the necessary background knowledge required to comprehend the paper.

Entity: An entity is a distinct and identifiable object or concept. For the scope of this work, we classify a token as an entity if it is associated with a Wikipedia page.

Heterogeneous Tables: Heterogeneity in tables can be of different types, such as a table can be heterogeneous based on the modality of data stored in the tables or based on the structure of tables. In this paper, we look into the heterogeneous tables in terms of tabular structure, where different tables have different schemas.

Embeddings: An embedding is a technique used to represent high-dimensional data in a lower-dimensional space as continuous vectors. This transformation captures the essential information and relationships between data points. When embeddings are close to each other in this lower-dimensional space, it indicates that the data points they represent are similar. Embeddings are widely used in machine learning to convert complex data into a format that can be more effectively processed and analyzed by algorithms.

Graph Neural Network (GNN): A GNN [30] is a type of neural network designed to work directly with graph-structured data. It leverages a graph's relationships and interactions between nodes (vertices) and edges to learn node embeddings.

Graph Convolutional Networks (GCN): GCNs [22] were initially proposed to address the problem of node classification in graphs, such as citation networks, where only a subset of nodes is labeled. Despite their original purpose, GCNs can be adapted for various other tasks as well. In our study, we utilize a multi-layer GCN with the following layer-wise propagation rule:

$$H^{l+1} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{l}W^{l}) \tag{1}$$

Here, $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph G with added self-connections. I_N is the identity matrix, $\tilde{D}_{ii} = \sum_j \tilde{A_{ij}}$ and $W^{(l)}$ is a layer-specific trainable weight matrix. $\sigma(\cdot)$

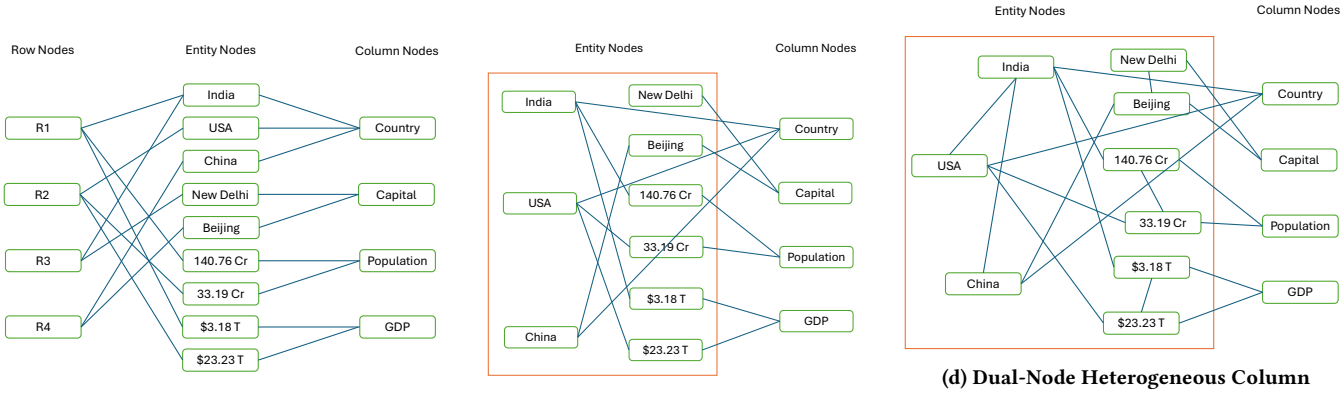
Table A

	Country (subject entity)	Population	GDP
R1	India	140.76 Cr	$3.18 T
R2	USA	33.19 Cr	$23.23 T

Table B

	Country (subject entity)	Capital
R3	India	New Delhi
R4	China	Beijing

(a) Sample tables for constructing graphs



(b) Tripartite Graph **(c) Dual-Node Heterogeneous Graph** **(d) Dual-Node Heterogeneous Column Graph**

Figure 1: Illustration of the three models of graph construction from tabular data using a pair of sample tables.

denotes an activation function, such as the $ReLU(\cdot) = max(0, \cdot)$. $H^{(l)} \in R^{N \times D}$ is the matrix activations in the l^{th} layer; $H^{(0)} = X$.

GraphSAGE: GraphSAGE [17] offers a versatile framework for generating inductive node embeddings. It leverages node features to learn an embedding function that can generalize to previously unseen nodes. Additionally, this algorithm can be applied to heterogeneous graphs with various types of nodes by using different weight matrices for each node type.

NodeFormer: The Graph Transformer [13] struggles to scale to large graphs due to the $O(N^2)$ complexity of its all-pair attention calculation. To address this, NodeFormer [34] introduces an efficient approach to the all-pair message passing computation in the Graph Transformer model, reducing its complexity from quadratic to linear. This enhancement allows it to scale to very large graphs. NodeFormer primarily focuses on node classification tasks, learning a latent graph structure from a large, potentially fully connected graph through differentiable computation. Although originally designed for node classification, we adapt NodeFormer by modifying the loss function to an unsupervised link prediction loss for our specific task.

Samplers: When the graph is too large, it becomes challenging to fit it into memory. In such cases, graph sampling techniques are employed. In this paper, we use the GraphSAINT Sampler [31] and Neighbor Sampler [17] whenever the graph size becomes too large for the model.

Missing Value Imputation (MVI): Given a table with subject entity columns and object entity columns where certain object entities are missing. MVI aims to accurately determine the missing object entities for their corresponding subject entities. Here, a subject entity refers to the main entity listed in the primary column of the table, while an object entity pertains to any other entity present within the same row.

4 MODELS OF GRAPH CONSTRUCTION FROM TABLES

In this paper, we introduce two novel methods for constructing graphs from a large web table corpus, which consists of multiple entity-centric heterogeneous tables. These methods are termed as *Dual-Node Heterogeneous Graph* and *Dual-Node Heterogeneous Column Graph*. Additionally, we enhance these graphs by incorporating weights into the edges, thus making both graphs weighted and directed. This added information enriches the graphs and provides entity co-occurrence information between two entities.

To evaluate the effectiveness of the proposed graphs, we test them on the MVI task. This involves initially learning embeddings of entities in the graph by feeding the graph into graph learning models. For the scope of this study, we use the following graph learning models: Graph Convolutional Networks (GCN) [22], GraphSAGE [17], and NodeFormer [34]. Once the embeddings are learned, they are used to impute the missing values in the tables.

4.1 Tripartite Graph (TG)

This graph construction is inspired by EmbDI [8]. It includes three types of nodes: Entity nodes, Row nodes, and Column nodes. Entity nodes correspond to the unique entities in the table corpus. Row nodes represent unique labels for each row in every table. Column nodes correspond to the unique column names in the table corpus. These nodes are connected by edges based on the structural relationships within the tables. Specifically, an edge exists between an entity node and a row node if that entity appears in the given row. Similarly, an edge exists between an entity node and a column node if that entity appears in the given column. This results in an unweighted and undirected TG (a sample example is depicted in Figure 1b).

Table 1: Statistics of the Train, Dev, and Test split of the preprocessed WikiTable corpus (670,171 tables).

	Train Set					Dev Set					Test Set				
	min	median	max	std dev	mean	min	median	max	std dev	mean	min	median	max	std dev	mean
rows/table	1	8	4670	27.31	13.17	5	12	667	29.95	20.18	5	12	3143	53.07	20.97
columns/table	1	5	99	3.39	5.58	3	5	43	2.74	5.90	3	5	43	2.59	5.86
entities/table	3	9	3911	43.36	18.72	8	34	2132	86.76	57.12	8	34	9215	160.05	59.92

4.2 Dual-Node Heterogeneous Graph (DNHG)

One of the major disadvantages of using TG is the presence of a vast number of row nodes. These row nodes inflate the size of the graph and do not significantly contribute to the downstream task, MVI, in our case. Therefore, the goal is to get rid of row nodes without impacting the overall effectiveness of the graph, which will substantially reduce the graph size. To achieve this, we propose a Dual-node heterogeneous graph that has two variants: an unweighted and undirected DNHG (Figure 1c) and a weighted and directed DNHG. The construction of both graphs is the same, except for the edge weights that are being added to make the graph weighted and directed.

Graph Construction: This graph contains only two types of nodes: Entity node and Column node. Entity nodes correspond to the unique entities in the table corpus. Column nodes correspond to the unique column names in the table corpus. An edge exists between an Entity node and a Column node if that entity occurs in that column in any table. Similarly, an edge exists between two Entity nodes if they occur in the same row in any table, and one of them is a subject entity.

Edge Weights: In addition, we assign weights to the edges of the graph to encode the information on the frequency of the co-occurrence of entities (for entity-entity edges) and the number of times an entity occurs in a particular column (for entity-column edges). After adding the weights, the graph will become weighted and directed. The weight of the edges is calculated as follows:

$$\text{Normalized weight from entity } A \text{ to entity } B = \frac{\text{Number of rows in which } A \text{ and } B \text{ occur together in any table}}{\text{Number of total occurrences of } B} \quad (2)$$

$$\text{Normalized weight from column node } C \text{ to entity } E = \frac{\text{Number of total columns with name } C \text{ in which entity } E \text{ occurs}}{\text{Number of total columns in which } E \text{ occurs}} \quad (3)$$

$$\text{Normalized weight from entity } E \text{ to column node } C = \frac{\text{Number of total columns with name } C \text{ in which entity } E \text{ occurs}}{\text{Number of tables in which column } C \text{ occurs}} \quad (4)$$

4.3 Dual-Node Heterogeneous Column Graph (DNHCG)

The DNHG does not consider the relation between entities within a table's column. Thus, we propose another graph construction method, i.e., a Dual-node heterogeneous column graph, which considers the entity-entity relationship in a given column. Similar to DNHG, we propose two variants of this graph: an unweighted and undirected DNHCG (Figure 1d) and a weighted and directed DNHCG.

Graph Construction: Same as DNHG, this graph also contains two types of nodes: Entity node and Column node. Entity nodes correspond to the unique entities in the table corpus. Column nodes correspond to the unique column names in the table corpus. An edge exists between an entity node and a column node if that entity occurs in that column in any table. Similarly, an edge exists between two entity nodes if they occur in the same row or column in any table. If the edge is because both entities are in the same row in the table, then one of the entities should be a subject entity.

Edge Weights: The edge weight is calculated similarly to DNHG for entity-column and entity-entity edges when both entities are from the same table row. The edge weight for the new type of edge (entity-entity in the same table column) is calculated as follows:

$$\text{Normalized weight from entity } A \text{ to entity } B = \frac{\text{Number of columns in which } A \text{ and } B \text{ occur together in any table}}{\text{Number of total occurrences of } B} \quad (5)$$

5 EXPERIMENT

5.1 Dataset

We use the WikiTables corpus [4] in all our experiments. This corpus is an extensive collection of tables extracted from Wikipedia pages, comprising approximately 1.65 million tables. To ensure the quality of our data, we perform pre-processing on this corpus, which is described further in this section.

5.1.1 ***Data Pre-processing.*** We adopted pre-processing steps similar to TURL [12], treating cells with hyperlinks as entity cells, where the linked page represents the entity. We don't use external knowledge bases for entity linking, relying solely on the hyperlinks. The pre-processing steps are as follows:

Table 2: Statistics of graphs constructed from the collection of training tables in WikiTable corpus.

Graph Name	No. of Nodes			No. of Edges	Degree			No. of Connected Components	No. of Nodes in Largest Connected Component
	Entity Nodes	Column Nodes	Row Nodes		min	max	avg		
TG	897,730	83,715	5,921,825	24,928,778	2	64,601	13.884	169	6,901,348
DNHG	897,730	83,715	-	11,270,548	2	25,917	9.3713	174	980,692
DNHCG	897,730	83,715	-	76,884,383	2	34,768	82.459	174	980,692

- **Removing Noisy Columns:** We first remove noisy columns, which include columns with empty or illegal headers (such as those containing only numbers, notes, or comments).
- **Retaining Columns with Entity Cells:** We retain all the columns that contain at least one entity cell. The remaining columns are entity-centric and are referred to as entity columns.
- **Identifying and Retaining Subject Columns:** We identify and retain tables that contain a subject column. A subject column is defined as one of the first two columns of a table that contains unique entities. Entities in a subject column are referred to as subject entities.
- **Filtering Tables by Size:** We remove tables that contain less than three entities or more than twenty columns.

5.1.2 ***Dataset statistics.*** After pre-processing, we obtain a refined dataset of 670,171 tables. The pre-processed dataset is split into training (~85%, 570,171 tables), validation (~7.5%, 50,360 tables), and testing (~7.5%, 49,640 tables) set. Table 1 details the characteristics of each split. The detailed statistics of graphs constructed on the train set are presented in Table 2.

5.1.3 ***Test dataset preparation.*** We extracted subject-object column pairs that contained at least three valid entity pairs from the test set, which was generated after the train-val-test split. We obtain 9,075 column pairs and a total of 130,711 samples for evaluation.

5.2 Baselines

We use TURL [12] as one of our baselines. This model is currently the state-of-the-art for the MVI task on the WikiTables dataset. Specifically, we use a version of TURL trained without table metadata (such as page title, section title, and table caption) for comparison with graph-based approaches, as we cannot incorporate this contextual information in GNN-based models. Another baseline we used is the tripartite graph (see section 3) with various graph learning models such as EmbDI [8], Graph Convolutional Networks (GCN) [22], GraphSAGE [17], and NodeFormer [34].

5.3 Experimental Setup

For different learning models, we used various hyperparameters based on the available resources to train each model. The following section will discuss the specific hyperparameters used in our experiments.

5.3.1 ***TURL.*** We trained TURL [12] for 50 epochs with a batch size of 8, compared to the original 80 epochs with a batch size of 16, due to resource constraints. The model has 304 million trainable parameters, with a transformer encoder of $N = 4$ layers, an input embedding dimension of $d = 312$, and an intermediate dimension of 1200 in the feedforward layers of the transformer [33] blocks. The multi-head attention modules use $k = 12$ attention heads. We used the Adam optimizer [21] with a learning rate starting at 1e-4, decreasing linearly.

5.3.2 ***EmbDI.*** We used two EmbDI variants in our experiments: the default "EmbDI" and a customized "EmbDI*". The default EmbDI, as described in [8], is trained on a tripartite graph with default parameters. For each node, 15 random walks of length 20 were performed, generating sentences, and node embeddings were learned using the skip-gram model from word2vec [23], with a window size of 5 and a minimum count of 2. In contrast, EmbDI* differs by training on alternative graphs instead of the tripartite graph but follows the same procedure for random walks and embedding learning, ensuring comparable embeddings across both variants.

5.3.3 ***GCN and GraphSAGE.*** We trained both GCN [22] and GraphSAGE [17] for 400 epochs. The models were configured with 256 hidden channels, 3 layers, and a learning rate of 0.001.

5.3.4 ***NodeFormer.*** We trained NodeFormer [34] for 300 epochs across all graphs. The model configuration includes 3 layers and 2 attention heads. For optimization, we used the Adam optimizer [21] with a learning rate of 1e-2.

Initialization of Embeddings: For GCN, GraphSAGE, and NodeFormer, we set the node embedding dimensions to 100. We used the TURL initialization technique to initialize the *entity node embeddings*. This technique employs the pre-trained 100-dimensional GloVe [28] embeddings for all entity mentions. The *entity node embeddings* are then initialized by averaging these pre-trained entity mention embeddings. Additionally, the *column* and *row* node embeddings are initialized by averaging the embeddings of all entities present in that column or row, respectively.

Inference: To predict the missing value of an empty cell, we are given the subject entity e from the row and the column header h of the cell. MVI inference involves two steps: first, identifying candidate values for the empty cell e (as done in TURL [12]), and then ranking these candidates based on the cosine similarity between their learned embeddings and the average of the learned embeddings of e and h.

5.4 Loss Function

For TURL and EmbDI, we used the same training loss functions mentioned in their original paper. For all the graph-based models, we used an unsupervised link prediction loss similar to that proposed in [12]. This loss function encourages nearby nodes to have similar representations while enforcing that the representations of

Table 3: MVI results of TURL (a transformer-based model).

Experiment	P@1	P@3	P@5	P@10
TURL without meta information trained on low resource setting	27.91	44.95	46.26	46.01
TURL without meta information with original checkpoint	**40.10**	**62.52**	**59.01**	**50.40**

Table 4: Performance of MVI with various graph neural network models trained on different approaches to graph construction from tabular data.

Graph	Model	P@1	P@3	P@5	P@10
TG	EmbDI	36.22	55.78	53.27	55.09
	GCN with Neighbor Sampler	26.21	56.22	52.31	47.21
	GCN with GraphSAINT Sampler	27.32	55.19	52.24	49.34
	GraphSAGE with Neighbor Sampler	36.31	59.27	51.66	50.17
	GraphSAGE with GraphSAINT sampler	37.41	60.86	52.64	52.12
	Heterogeneous GraphSAGE with Neighbor Sampler	35.14	52.24	44.67	50.22
	Heterogeneous GraphSAGE with GraphSAINT sampler	36.21	54.15	46.58	51.18
	NodeFormer with Neighbor Sampler	36.35	56.54	48.12	51.13
	NodeFormer with GraphSAINT sampler	37.98	57.21	49.32	53.24
DNHG	EmbDI*	37.11	57.28	52.21	**56.12**
	Full batch GCN on unweighted graph	28.72	56.79	52.30	50.24
	Full batch GCN on weighted graph	30.42	57.61	52.01	52.28
	Full batch GraphSAGE on unweighted graph	38.32	61.57	53.68	52.27
	Full batch GraphSAGE on weighted graph	39.11	**62.05**	54.22	53.02
	Full batch Heterogeneous GraphSAGE on unweighted graph	37.41	55.54	47.57	52.38
	Full batch NodeFormer on unweighted graph	38.35	58.12	50.02	54.23
DNHCG	EmbDI*	37.32	58.32	**54.29**	55.14
	GCN with Neighbor Sampler on unweighted graph	27.71	55.79	52.31	50.01
	GCN with GraphSAINT Sampler on unweighted graph	26.13	53.29	50.01	49.14
	GCN with Neighbor Sampler on weighted graph	29.21	55.21	49.13	50.02
	GCN with GraphSAINT Sampler on weighted graph	30.01	56.16	51.22	52.07
	GraphSAGE with Neighbor Sampler on unweighted graph	36.31	59.27	50.28	50.12
	GraphSAGE with GraphSAINT Sampler on unweighted graph	37.22	60.51	54.12	51.24
	Heterogeneous GraphSAGE Neighbor Sampler on unweighted graph	35.21	53.22	47.67	52.38
	Heterogeneous GraphSAGE with GraphSAINT Sampler on unweighted graph	36.43	54.23	46.24	53.18
	Full batch Nodeformer on unweighted graph	**40.45**	59.12	53.02	55.31

disparate nodes are highly distinct.

$$J_G(z_u) = -log(\sigma(z_u^T z_v)) - Q.E_{v_n \sim P_n(v)} log(\sigma(-z_u^T z_{v_n})) \quad (6)$$

where v is a node that is adjacent to u, σ is the sigmoid function, P_n is a negative sampling distribution, and Q defines the number of negative samples. We use a negative sampling ratio of 1:1, i.e., 1 negative sample for every positive sample.

5.5 Evaluation Metric

For value ranking, we only consider test instances where the target object entity is present in the candidate set, and the number of candidates is greater than one. Once the candidates are ranked, we evaluate the results using Precision@K. We only include samples with more than K candidates in the candidate set for calculating Precision@K. This approach differs from the metric used in TURL [12], which uses all samples for Precision@K. We chose not to follow this method because it would count all samples with fewer than K candidates as having correctly predicted the missing entity, which does not accurately reflect the model's efficiency.

5.6 Experimental Results

In this section, we present the results of all the experiments conducted. We begin with the results of the transformer-based baseline, TURL [12]. Following this, we present the results for graph learning models on the Tripartite graph, the Dual-node heterogeneous graph, and the Dual-node heterogeneous column graph, respectively.

5.6.1 ***MVI Results on TURL.*** Table 3 shows TURL's performance on the MVI task using the pre-processed WikiTables corpus (see Section 5.1). The original TURL method leverages meta-information, which is difficult to integrate into graph structures, so we only consider results without meta-information. Inference was conducted using TURL's original pre-trained checkpoint, leading to results that may differ from those in the original paper due to variations in evaluation metrics (Section 5.5).

Training TURL from scratch on our hardware took about 110 hours (50 epochs with batch size 8), compared to the original 80 epochs with double the batch size. Excluding meta-information reduced TURL's accuracy by 5.85% ± 2.05%, but it's important to note that meta-information isn't always available.

5.6.2 ***MVI Results on TG.*** Table 4 shows the result for the TG on EmbDI and various graph learning models. Since the TG constructed on WikiTables corpus is too large, we can't train graph learning models off the shelf. Thus, we used GraphSAINT [31] and Neighbor Sampler (see Section 3) to sample a subset of nodes that can be processed at once by the models.

Generating random walks for EmbDI on the TG took approximately 3 hours, followed by an additional 11 hours for training on the CPU. In contrast, the graph learning models (GCN, GraphSAGE, Heterogeneous GraphSAGE, and NodeFormer) took around 2-3 hours to reach convergence.

5.6.3 ***MVI Results on DNHG.*** Due to the significantly smaller number of nodes and edges in DNHG compared to a TG, all graph learning models can process the entire graph simultaneously, eliminating the need for sampling in the DNHG. Table 4 shows the results for the DNHG using EmbDI and various graph learning models. As shown in the table, almost all the models perform better on the DNHG compared to the TG. This improved performance can be attributed to the models' ability to load the entire graph at once (since the graph is very small as compared to the TG), allowing each node to interact with all its adjacent nodes during message passing in the training process. It is also worth noting that GraphSAGE on the weighted DNHG outperforms other models at P@1, P@3, and P@5 on the DNHG. This demonstrates that our novel edge-weighting criteria add meaningful information to the graph. Additionally, the results on this graph are comparable to TURL, except for P@10, where EmbDI* outperforms TURL without meta information with the original checkpoint. whereas the results on this graph outperform TURL without meta information trained on low resource settings.

Generating random walks for EmbDI* on this graph took approximately 2 hours 30 minutes, which is less than the time taken in the TG due to the lesser number of nodes. Further, training embedding took another 8 hours. In contrast, the graph learning models (GCN, GraphSAGE, Heterogeneous GraphSAGE, and NodeFormer) took around 2-3 hours to reach convergence.

5.6.4 ***MVI Results on DNHCG.*** This graph is more dense than the DNHG as the number of edges increases significantly in this graph due to adding new edges between entities in the same column. Due to this, graph learning models, more specifically GCN and GraphSAGE, can't be used off the shelf, and sampling is required to run these models on this graph. On the other hand, NodeFormer can process the entire graph at once.

The results for EmbDI* and various graph learning models on this graph are shown in Table 4. Almost all the models performed better on the DNHCG compared to the TG. However, these models performed poorly on the DNHCG compared to the DNHG, except for NodeFormer. The poorer performance on the DNHCG is due to the use of samplers, as these models cannot handle the entire graph. NodeFormer performs better on this graph because it can train on the whole graph. From the experiments, it can be seen that adding edges for entities in the same column provides meaningful information about the table and its structure. Another insight from these experiments is that graph learning models perform better when they can run on the entire graph rather than using samplers. Therefore, more innovation is needed in constructing graphs from tables so that the resulting graph remains small while capturing all relevant information from the tables.

The training time of graph learning models on this graph is similar to the Dual-node heterogeneous graph.

5.7 System Configuration

We run the EmbDI model and inference on all the GNN-based models on Intel(R) Xeon(R) x86-64 machines with 64 CPUs and 337 GB memory. All attention-based models and GNN models are trained with NVIDIA GTX 1080 Ti GPUs having 11 GB Memory.

6 CONCLUSIONS AND FUTURE WORK

Our hypothesis in this paper was that in addition to the sophistication of GNNs employed, the choice of the graph representation of tabular data impacts the performance significantly. We observe that GCN and GraphSAGE achieve their best performance on DNHG as it allows processing of the entire graph without the need for sampling. Even in cases where sampling is used, the performance remains closely comparable to that without sampling, demonstrating that our approach can be easily scaled to large corpora without significant degradation in performance. We find that assigning weights to the edges to encode the frequency of co-occurrence of entities provides a slight improvement over the unweighted version.

In terms of computational performance, TURL takes approximately 110 hours to train to convergence on our hardware, while all GNN-based models reach convergence in about 2-3 hours. Thus, we achieve comparable or superior performance to transformer-based models, which require significantly more training time than GNNs. We achieve slightly better performance than TURL without table meta-information and comparable performance to TURL trained with double the batch size on high-end hardware.

Our work in this paper not only paves way for conducting similar evaluation on other tasks on tabular datasets with careful choice of graph representation, but also opens up an interesting direction of research for automatically selecting appropriate graph transformation for the given tabular data and task. Another important aspect that is missing in our work which we plan to explore in future, is the inability to incorporate meta-data (e.g., caption) associated with each table. Finally, we plan to work on building a end to end question answering framework based on graph representation of tabular data. While all transformer-based approaches currently handle Q&A tasks with questions directed at only one table at a time, graph-based approaches could be used to develop a pipeline for Q&A across multiple tables.

REFERENCES

[1] Louie Andre. 2024. 53 Important Statistics About How Much Data Is Created Every Day in 2024. https://financesonline.com/how-much-data-is-created-every-day. Online; Accessed On: 22 July 2024.

[2] Gilbert Badaro, Mohammed Saeed, and Paolo Papotti. 2023. Transformers for Tabular Data Representation: A Survey of Models and Applications. *Transactions of the Association for Computational Linguistics* 11 (2023), 227–249. https://api.semanticscholar.org/CorpusID:253736389

[3] Jinze Bai, Jialin Wang, Zhao Li, Donghui Ding, Ji Zhang, and Jun Gao. 2021. ATJ-Net: Auto-Table-Join Network for Automatic Learning on Relational Databases. In *Proceedings of the Web Conference 2021* (Ljubljana, Slovenia) *(WWW '21)*. Association for Computing Machinery, New York, NY, USA, 1540–1551. https://doi.org/10.1145/3442381.3449980

[4] Chandra Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. TabEL: Entity Linking in Web Tables. In *International Workshop on the Semantic Web*. https://api.semanticscholar.org/CorpusID:14265783

[5] L. Breiman. 2001. Random Forests. *Machine Learning* 45 (2001), 5–32. https://api.semanticscholar.org/CorpusID:89141

[6] L. Breiman, Jerome H. Friedman, Richard A. Olshen, and C. J. Stone. 1984. Classification and Regression Trees. *Biometrics* 40 (1984), 874. https://api.semanticscholar.org/CorpusID:29458883

[7] Erick Cantú-Paz and Chandrika Kamath. 2003. Inducing oblique decision trees with evolutionary algorithms. *IEEE Trans. Evol. Comput.* 7 (2003), 54–68. https://api.semanticscholar.org/CorpusID:748190

[8] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. 2020. Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks. *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (2020). https://api.semanticscholar.org/CorpusID:267920445

[9] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016). https://api.semanticscholar.org/CorpusID:4650265

[10] Milan Cvitkovic. 2020. Supervised Learning on Relational Databases with Graph Neural Networks. *ArXiv* abs/2002.02046 (2020). https://api.semanticscholar.org/CorpusID:211043697

[11] Li Deng, Shuo Zhang, and Krisztian Balog. 2019. Table2Vec: Neural Word and Entity Embeddings for Table Population and Retrieval. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval* (2019). https://api.semanticscholar.org/CorpusID:70017945

[12] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL. *ACM SIGMOD Record* 51 (2020), 33 – 40. https://api.semanticscholar.org/CorpusID:220128303

[13] Vijay Prakash Dwivedi and Xavier Bresson. 2020. A Generalization of Transformer Networks to Graphs. *ArXiv* abs/2012.09699 (2020). https://api.semanticscholar.org/CorpusID:229298019

[14] Yoav Freund and Robert E. Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*. https://api.semanticscholar.org/CorpusID:6644398

[15] Garima Gaur, Rajat Singh, Siddhant Arora, Vinayak Gupta, and Srikanta J. Bedathur. 2023. Teaching Old DB Neu(ral) Tricks: Learning Embeddings on Multi-tabular Databases. *Proceedings of the 6th Joint International Conference on Data Science & Management of Data (10th ACM IKDD CODS and 28th COMAD)* (2023). https://api.semanticscholar.org/CorpusID:255416724

[16] Martin Grohe. 2020. word2vec, node2vec, graph2vec, X2vec: Towards a Theory of Vector Embeddings of Structured Data. *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (2020). https://api.semanticscholar.org/CorpusID:214713780

[17] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Neural Information Processing Systems*. https://api.semanticscholar.org/CorpusID:4755450

[18] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. TaPas: Weakly Supervised Table Parsing via Pre-training. In *Annual Meeting of the Association for Computational Linguistics*. https://api.semanticscholar.org/CorpusID:214802901

[19] Tin Kam Ho. 1995. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, Vol. 1. 278–282 vol.1. https://doi.org/10.1109/ICDAR.1995.598994

[20] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Neural Information Processing Systems*. https://api.semanticscholar.org/CorpusID:3815895

[21] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). https://api.semanticscholar.org/CorpusID:6628106

[22] Thomas Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *ArXiv* abs/1609.02907 (2016). https://api.semanticscholar.org/CorpusID:3144218

[23] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *International Conference on Learning Representations*. https://api.semanticscholar.org/CorpusID:5959482

[24] Sreerama K. Murthy. 1998. Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey. *Data Mining and Knowledge Discovery* 2 (1998), 345–389. https://api.semanticscholar.org/CorpusID:207741345

[25] Sebastian Nowozin. 2012. Improved Information Gain Estimates for Decision Tree Induction. *ArXiv* abs/1206.4620 (2012). https://api.semanticscholar.org/CorpusID:2355737

[26] Liudmila Ostroumova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2017. CatBoost: unbiased boosting with categorical features. In *Neural Information Processing Systems*. https://api.semanticscholar.org/CorpusID:5044218

[27] Jie Ouyang, Nilesh V. Patel, and Ishwar K. Sethi. 2008. Chi-Square Test Based Decision Trees Induction in Distributed Environment. *2008 IEEE International Conference on Data Mining Workshops* (2008), 477–485. https://api.semanticscholar.org/CorpusID:9011011

[28] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Conference on Empirical Methods in Natural Language Processing*. https://api.semanticscholar.org/CorpusID:1957433

[29] J. Ross Quinlan. 2004. Induction of decision trees. *Machine Learning* 1 (2004), 81–106. https://api.semanticscholar.org/CorpusID:13252401

[30] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* 20 (2009), 61–80. https://api.semanticscholar.org/CorpusID:206756462

[31] Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C. Bayan Bruss, and Tom Goldstein. 2021. SAINT: Improved Neural Networks for Tabular Data via Row Attention and Contrastive Pre-Training. *ArXiv* abs/2106.01342 (2021). https://api.semanticscholar.org/CorpusID:235293989

[32] Suryakanthi Tangirala. 2020. Evaluating the Impact of GINI Index and Information Gain on Classification using Decision Tree Classifier Algorithm*. *International Journal of Advanced Computer Science and Applications* 11 (2020). https://api.semanticscholar.org/CorpusID:212657021

[33] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Neural Information Processing Systems*. https://api.semanticscholar.org/CorpusID:13756489

[34] Qitian Wu, Wentao Zhao, Zenan Li, David Paul Wipf, and Junchi Yan. 2023. NodeFormer: A Scalable Graph Structure Learning Transformer for Node Classification. *ArXiv* abs/2306.08385 (2023). https://api.semanticscholar.org/CorpusID:258509408

[35] Pengcheng Yin, Graham Neubig, Wen tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. *ArXiv* abs/2005.08314 (2020). https://api.semanticscholar.org/CorpusID:218674345

[36] Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir R. Radev, Richard Socher, and Caiming Xiong. 2020. GraPPa: Grammar-Augmented Pre-Training for Table Semantic Parsing. *ArXiv* abs/2009.13845 (2020). https://api.semanticscholar.org/CorpusID:221995589