Probabilistic Graph Model for Pure Data

1 Methodology of the Graph Probabilistic Model

In this paper, we describe the methodology of a graph-based probabilistic model for Pure Data (PD). We treat each PD file as a graph, and for each node in the graph, we replace the node with a blank and predict the missing element. This prediction is based on the subgraphs the node belongs to and our corpus of one, two, and three-node subgraphs along with their frequencies. We save the top 10 prediction values for each node and record the rank of the correct prediction to calculate the Mean Reciprocal Rank (MRR) score.

Our code is available in GitHub [1]. The methodology is as follows:

Data preparation: We partition the PD projects into training and testing sets and extract parsed PD files for each dataset.

Graph Construction: We build a graph from the list of nodes and connections in a parsed PD file. After that, we extract the unique nodes such as `msg`, `tgl`, etc and count the frequency of each node; and save them in our corpus.

Subgraph Analysis: We extract 2 and 3 node subgraphs from our training set graphs and calculate the frequencies of the 2 and 3-node subgraphs, and store them in our corpus. We also gather data on the three-node combinations observed in our dataset, irrespective of the connections between them.

Research Questions: Given our corpus of unique tokens in the training set PD files, the 2- and 3-node subgraphs along with their frequencies, and a list of 3-node combinations, we build a prediction model for Pure Data graphs. This model uses the probabilities of the subgraphs to predict the content of an empty position in a PD graph, based on all the subgraphs in our training set and their frequencies. With our model, we can address the following research questions:

1. Given a collection of subgraphs along with their respective probabilities, what is the likelihood of a particular subgraph?
2. In the scenario of a graph featuring an empty node, what is the most probable node to occupy the `BLANK` position?

Evaluation: Finally, we evaluate the performance of our graph based model using our test dataset and calculate the MRR score for each of the test graphs.

We explain the stages in the following sections.

1.1 Data Preparation

We partitioned the PD projects into an 80-20 split for training and testing purposes, utilizing the `train_test_split` [2] function from the `scikit-learn` [3] library in Python.

We gathered the SHA-256 hashes corresponding to the revisions of the PD file contents for both the train and test projects from our dataset. Following this, we refined the test hashes by eliminating any matches found in the training set to assess our model's performances on paths generated from previously unseen parsed contents. This process guarantees that our training and test data originate from distinct projects, minimizing overlaps between them. Our training set consists of 163,912 PD files (graphs), while our test set includes 39,412 graphs.

1.2 Graph Construction

We utilized the parsed contents of the PD files from our dataset [4] to generate graphs from the PD source code. We extracted a list of connections between the PD file objects using the `Contents` table from our dataset for each parsed content. We constructed a directed graph using the connection information extracted from each parsed content. For instance, if there was a connection from `object_0` to `object_1`, we added both `object_0` and `object_1` as nodes in the graph, if not already added, and created a directed edge between them. In this context, `object_0` represents the `source` of the connection, while `object_1` is the `destination`.

Following that, we computed the frequency of each unique node, pairs and triplets of nodes within the graph and stored the results. Our training set comprises 34,565 unique nodes, with the most frequent ones being `msg`, occurring 2,880,151 times across all parsed PD programs in the training set; `r`, occurring 1,311,419 times; and `t`, appearing 1,195,537 times. Message boxes are containers for one or more messages, which are transmitted to their designated outlets or destinations upon activation of the box [5, 6]. Notably, `r` stands for `receive`, responsible for message reception, while `t` denotes `trigger`, facilitating message sequencing and conversion [5, 7, 6].

1.3 Subgraph Analysis

We collected data on 2-node and 3-node subgraphs from each PD file graph to build our training subgraph corpus. For the 2-node subgraphs, we examined the graph's edges to understand their connectivity. Then, we applied a depth-first search of length two from each node in the undirected version of the graph to identify subgraphs containing 3 nodes.

For each connected 2-node and 3-node subgraph, we arranged the nodes based on their object type (e.g., `msg`, `tgl`) and constructed the adjacency matrix of the subgraph. In this matrix, the first row represents connections originating from the first sorted node, the second row represents connections from the second sorted node, and so forth. Our training corpus includes 217,803 unique 2-node subgraphs and 1,285,324 unique 3-node subgraphs.

To uniquely identify the 2-node and 3-node subgraphs, we generated a key for each subgraph using the index of the elements in the subgraph concatenated with their adjacency matrix. For example, for a subgraph msg $\rightarrow$ floatatom, where the index of

msg in our corpus of unique tokens is 100, and the index of floatatom is 95, the key to uniquely identify this subgraph would be `95,100,0010`, where 0010 represents the adjacency matrix between these two nodes.

We recorded the frequency of each subgraph to facilitate probability calculations. Additionally, we stored information on all three-node combinations seen in the training graphs to assist in our prediction stage.

1.4 Research Questions

After constructing our corpus of subgraphs, we try to answer two research questions:

RQ1: Given a collection of subgraphs along with their respective probabilities, what is the likelihood of a particular subgraph?

RQ2: In the scenario of a graph featuring an empty node, what is the most probable node to occupy the `BLANK` position?

1.4.1 RQ1: Calculating the probability of a given subgraph based on our corpus of subgraphs

We determine the probability of a subgraph based on the probabilities of the 1, 2, and 3 node subgraphs in our corpus. The probability of a 3-node subgraph is calculated as the frequency of that subgraph divided by the total frequency of all 3-node subgraphs. Similarly, the probability of a 2-node subgraph is calculated as the frequency of that subgraph divided by the total frequency of all 2-node subgraphs. For a 1-node subgraph, the probability is determined by dividing the frequency of that node by the total frequency of all unique nodes. The algorithm is as follows:

1. If all the nodes of a given test subgraph are present in our corpus, we generate the subgraph's key following the methodology in Section 1.3. If the key exists in our corpus, the score is assigned based on the probability of the 3-node subgraph.

2. If any node of the subgraph is absent from our corpus or the 3-node subgraph key does not exist, implying that this particular subgraph has not been seen before, we query the 2-node subgraphs using the same procedure mentioned in Section 1.3. For each 2-node subgraph in our input graph, if its key exists in the corpus, we multiply the final probability score by the probability of the 2-node subgraph.

3. If the 2-node subgraphs are not present in our corpus or any node of the 2-node subgraphs is absent, we adjust the score by the probability of the individual nodes within the subgraph. Additionally, if the node is not present in our corpus, the score is multiplied by a very small value, such as $0.5 \times$ (1/total frequencies of all unique nodes).

4. For each missed 3-node and 2-node subgraph, the score is multiplied by a small discount factor to indicate that these subgraphs are unseen in our corpus, thus their probabilities should be lower compared to those observed. We use a discount factor of 0.05.

Finally, we return the score computed through the aforementioned algorithm to estimate the likelihood of a subgraph.

1.4.2 Predicting an empty node of a given graph

To predict an empty node in a given graph, we first extract all the three-node subgraphs in that graph that include the node and then score these subgraphs using the procedure outlined in Section 1.4.1. We aggregate the predictions of all subgraphs that this node is a part of to select the top 10 highest-scoring candidates to occupy the blank position of the graph.

To identify potential candidates for a blank node in a subgraph, we search our corpus for nodes previously observed with the other two non-blank nodes. For example, in a subgraph `BLANK` → `msg`; `BLANK` → `floatatom`, we examine our corpus for nodes previously seen with `msg` and `floatatom`. We compile a list of these nodes as potential candidates, place them in the blank position, and score the subgraph.

We maintain a heap of length 10 to store the candidates with the highest scores from the subgraphs containing the empty node. Finally, we record the index of the correct prediction from the heap to calculate the MRR score, assigning -1 if the correct node is not in the top 10 positions.

1.5 Evaluation

For each parsed content extracted from our test hashes, we construct graphs using the methodology described in Section 1.2. Then, we replace each node in the graph with a blank and predict the missing node following the procedure outlined in Section 1.4.2. We record the index of the correct prediction from the heap and calculate the MRR score for each test graph using the following formula:

$$\text{MRR} = \frac{\sum_{i=1}^{N} \text{RR}_i}{N}$$

where `N` is the number of total nodes in the graph, and the reciprocal rank (RR) of a node is calculated as:

$$\text{RR} = \frac{1}{\text{rank}}$$

In this context, `rank` represents the index of the correct prediction in the heap. If the index is -1, the RR of a node is 0.

In the initial stage, we calculated the MRR for 7,626 test graphs. Figure 1 illustrates the distribution of MRR across these graphs. As shown in the figure, the mean MRR value for our probabilistic model is 0.43. Additionally, 86 graphs have an MRR of 1.0, while 98 graphs have an MRR of 0, indicating that the model failed to correctly predict the nodes of these graphs, as the correct predictions were not within the top 10 results. Table 1 presents a summary of these statistics.

Table 1: Summary statistics of our test graphs

	Total	**Mean**	**Min**	**Q1**	**Q2**	**Q3**	**Max**
MRR	7,626	0.43	0	0.32	0.42	0.52	1.0

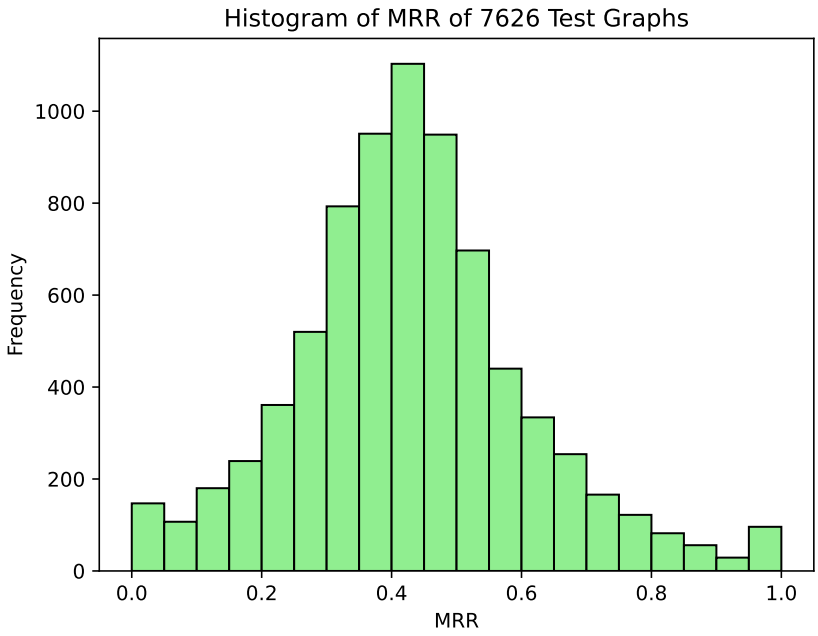

Figure 1: MRR of test graphs

2 Limitations

There are some limitations to our model which are listed below:

2.1 Corpus limited to 2 and 3 node subgraphs

Our corpus is constructed solely from 2-node and 3-node subgraphs. However, the model's performance might have seen improvement had we incorporated larger subgraphs since doing so would have provided additional contexts for predicting unknown tokens.

References

[1] Anisha Islam. Probability Estimator for Visual Code. `https://github.com/anishaislam8/Probability-Estimator-For-Visual-Code`. Accessed: 2024-05-14.

[2] scikit-learn developers (BSD License). sklearn.model_selection.train_test_split. `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`. Accessed: 2024-02-19.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[4] Anisha Islam. Opening the Valve on Pure-Data Dataset. https://archive.org/details/Opening_the_Valve_on_Pure_Data, 2023. Accessed: 2024-02-19.

[5] Miller Puckette et al. Pure Data: another integrated computer music environment. *Proceedings of the second intercollege computer music concerts*, pages 37–41, 1996.

[6] IEM. Pure Data. https://puredata.info/. Accessed: 2024-01-20.

[7] K. Barkati and N. Granieri. Pure Data Reference Card. https://puredata.info/docs/manuals/pdrefcards/pd-refcard-en.pdf/view, 2018. Accessed: 2023-11-12.