Putting Parts in Perfect Places

Aaron Tacke, Inken Grüner, Sebastian Schneider

March 1, 2024

Abstract

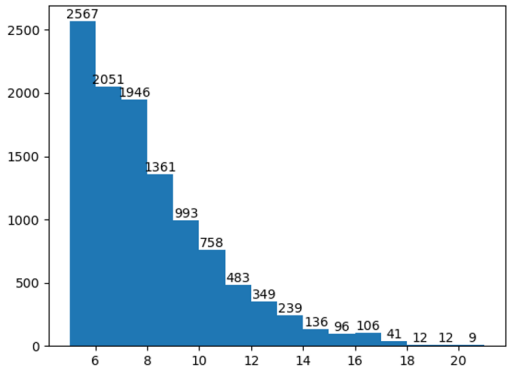
Developing a Model Implementation for creating an undirected Graph given a Set of Parts.

1 Fundamentals

1.1 Statistics

1. **Total Graphs:** 11159
2. **Splits:**
   - *Train Set:* 7811 (70%)
   - *Validation Set:* 1674 (15%)
   - *Test Set:* 1674 (15%)

   In accordance with the lecture nodes [Car23], the validation and test set are sized equally. We chose 15% since we consider 1674 instances sufficient to determine the quality of the models.
3. **Amount of Graphs with Sizes:**



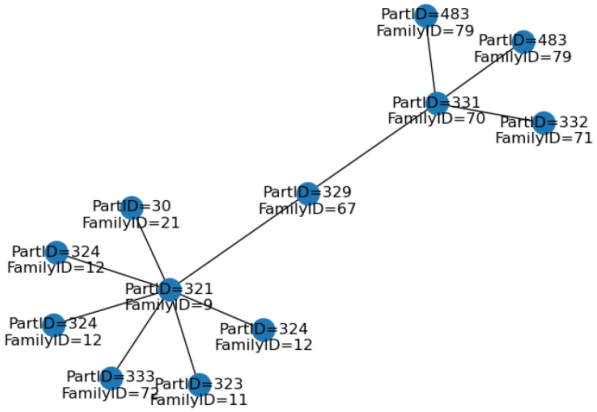
By using overviews like this and by looking at individual graphs, we tried to get a feeling for the data.

1.2 Ordering of Parts

Since all graphs are trees, we can create them by appending new nodes sequentially with one edge each. This method requires ordering parts according to a given sorting criterion, which allows the tree to be built correctly. For each part, we consider all appearances in the training set. For each appearance, we extract the degree of the node of the part. Then, we use the following functions (see Ordering in Table) to aggregate the degrees for every part and sort the parts by the resulting corresponding value.

Ordering	Maximum Achievable Graph Accuracy
Average	ca. 0.9918
Maximum	ca. 0.9808
Median	ca. 0.7135
Minimum	ca. 0.4809
-Length	ca. 0.4990
-Sum	ca. 0.3583

Using the average degree in the training set for ordering parts, 99.18% of trees can be built sequentially. Interestingly, the graphs that cannot be constructed sequentially all look similar:



As you can see, some parts (here 321 and 331) with high degrees are connected by another node. Since the high-degree parts are the first ones in the ordering, they are connected when building the tree, leading to a wrong graph.

By artificially increasing the average degree of the connecting parts by 0.1 for every appearance of this phenomenon, we reach an achievable graph accuracy of 100% on the training set after only two epochs. For the validation set, the achievable graph accuracy for this trained version of the ordering by average degrees is 99.94%.

1.3 Metrics

1.3.1 Edge Accuracy

Edge Accuracy produces misleading results due to the nature of underlying graphs. Speaking in concrete terms, 80% of Edges would be predicted correctly if no edge is set between any two parts in the set of parts. As a result, the evaluation of approaches could be problematic, as high values are more easily achievable, making mediocre solutions look overly promising. Due to this, we have chosen to adopt **Graph Accuracy** as the primary metric for evaluating models. Hence, only graphs containing all nodes and all edges between nodes are determined to be correct. This penalizes "almost" correct graphs a lot harsher than Edge Accuracy; however, we feel this benefits choosing better models in the long run. Also, high graph accuracy implies high edge accuracy, as we will see in the results at the end.

1.4 Auto-Encoder

Approaches like the LSTM require the input to reflect specific features of the parts in the different time steps. Using Family IDs in addition to parts has degraded the performance of the approach. However, providing the family ID as a feature should give the model meaningful additional information. To provide this information effectively, we want to use Embeddings that offer both PartID and FamilyID in a denser format and ideally reflect the closeness of parts in graphs through the similarity of generated Embeddings.

2 Approaches

2.1 Long Short-Term Memory

First, we tried to use a Long Short-Term Memory approach without using the tree-based sequential building. Even after tweaking the loss function accordingly, it exploited the edge accuracy (See section 1.3.1) by consistently predicting too few edges. Initial attempts to exploit the tree property of the graphs far exceeded the previous performance, which is why we concentrated on this.

By leveraging the tree-based nature of the Graphs, we can use an RNN with LSTMs as presented in the lecture [Car23] to have the network generate the edge between the new node at the current time step and the other nodes from the previous time steps. This gives us a graph with all nodes and edges for each node. Implementation can be found in LSTM.ipynb. Evaluation of Sequences of PartIDs and Embeddings leads to the following Results in terms of Graph Accuracy (GA) and Edge Accuracy (EA):

Experiment	Achieved Graph and Edge Accuracy
1000 Epochs on PartIDs	GA: 32% Train / 30,9% Val; EA: 58,8%
2000 Epochs on PartIDs	GA: 32,4% Train / 31,4% Val; EA: 58,6%
1000 Epochs on Embeddings	GA: 30,7% Train / 28,55% Val; EA: 55,9%

Results for training on PartIDs as Input
Using an LSTM-based Approach seems promising, as we can use Softmax to ensure that each timestep receives one additional node and has to predict a new edge connecting the new node to a different node in the graph. Initial approaches using LSTM require extended learning to achieve satisfactory results. Unfortunately, even training for up to 2000 Epochs, the Graph Accuracy of the predictions remains comparatively low.
⇒ Graph Accuracy remains low for the current Input → Output Combination. We will postpone further testing until the completion of Embeddings.

Results for training on Embeddings for PartIDs as Input
Based on the Results of Training the LSTM on a Sequence of Embeddings associated with the PartIDs, Graph Accuracy fails to improve while Loss Plateaus. As a result, we surmise the RNN fails to leverage the additional information found in the embeddings to more effectively predict Graphs.

2.2 Graph Recurrent Attention Network

Based on the paper by Liao et al. [ea20], an Attention-based Network is used to create b (Block Size) nodes at a time (including the edges between these new nodes). In a subsequent step, the network can generate edges between the new block and the already-developed parts of the graph. Data has to be transformed to fit the required format of the official repository `https://github.com/lrjconan/GRAN` to include the following:

- Edges of the Graph (Target)
- Nodes of the Graph
- Attributes of Nodes
- Graph Identifier for each node

Results
Training the model on the Training Set (70%) for 500 Epochs yields unpromising results. The reason for that lies in the fact that the intended application of GRAN is to find recurring patterns in the Training Data and use it to generate entirely new graphs that are only meant to exhibit similar patterns shown in the Training set. Providing a set of Nodes on which to construct the Graph is, at least in the context of the original intention of the paper, not meant as part of the application flow. In theory, removing the step of the model in which the parts are generated should be possible; however, training an attention-based network does entail some hurdles as more significant amounts of data would likely be required, and using the approach of the paper for an entirely different application would likely lead to unsatisfactory results.

2.3 Instance-Based

2.3.1 The evolution of our Instance-Based approaches

Again, we started without building the graphs sequentially, using what we call the **Dendrogram Approach:**

Considering all parts as their own set, we successively link two parts from different sets and union the respective sets until a tree is created. To choose which parts to connect next, we compared the number of edges between pairs of parts in the training set. This approach achieved a graph accuracy of 0.15% on the training set, which was higher than the 0% graph accuracy achieved with unordered LSTM approaches.

Learnings

- The prevalence of edges in the training set can be used to determine which edges should be connected.
- The individual decisions are sensible when only looking at two parts at the same time, but models without context information are minimal
- Looking at the falsely created graphs, many of them couldn't even be made during a sequential creation using a correct ordering (see section 1.2).

These learnings led us to try the **sequential creation of graphs** (comparable to the LSTM approach in section 2.1) with our Instance-Based method.

By again comparing the number of edges between two parts in the training set, we achieved a graph accuracy of 61.1% on the training and 59.6% on the validation set.

We tested various metrics for choosing the next edge. We found that the relative prevalence (number of edges in the training set divided by the total number of occurrences of the two parts connected by an edge) achieved better results (graph accuracy of 62.3% and 61.2% for training and validation sets, respectively). Despite different approaches, we could not improve the approach by using the family IDs of parts as part of the metric.

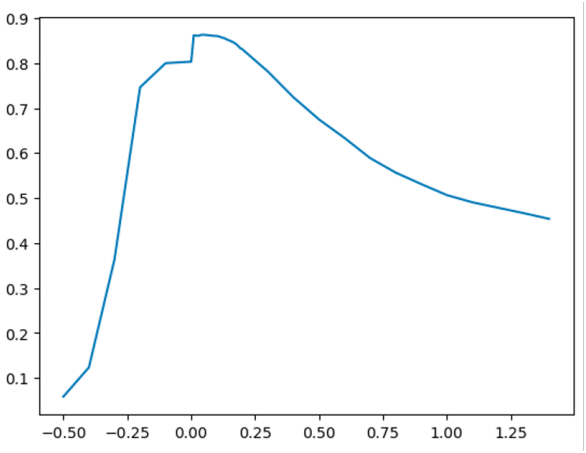
Learnings

- Sequentially building graphs harmonizes with the given graph problem
- The family ID could be useful in rare cases if the part id is unknown, but is not generally beneficial given the current state of this approach
- Looking at the falsely created graphs, many are (almost) star-shaped, since high-degree parts are sometimes wrongly favored. Too little context information is available to avoid this.

To improve our model based on these learnings, we **added context information during creation:**

By extracting the maximum degree of parts in the training set, we can prohibit adding edges to already saturated nodes (reached the maximum degree). This increased the graph accuracy on the training and validation set to 80.1% and 77.3%, respectively.

Since we already know the average degree from learning a useful order (see section 1.2), we tried to reward parts reaching the average degree and penalize overshooting it, weighed using a hyperparameter. We used a zooming grid search to find a good value for this hyperparameter:



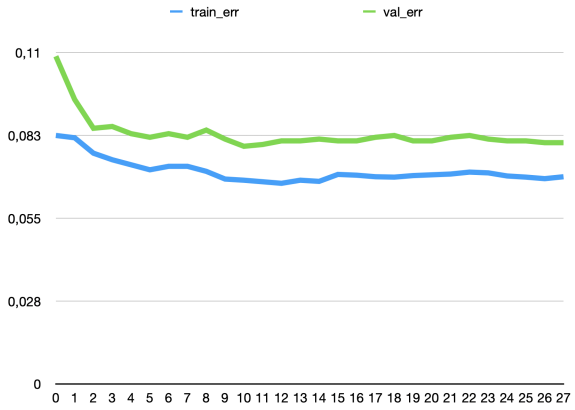
The resulting graph accuracy was 86.3% for the training set and 83.5% for the validation set, using 0.05 as the hyperparameter.

Learnings

- Now, when looking at falsely created graphs, it's hard to say what went wrong → Obvious low-hanging fruits seem to be picked.
- Weighted restrictions using hyperparameters are useful for including context information in choosing edges. Thus, we also changed exceeding the maximum Degree from a hard restriction to a weighted penalty.

Since it's hard to say manually which problems lead to some graphs not being created correctly, we now rely on **ML-based improvements** similarly to the trained ordering (see section 1.2):

We first tried to artificially decrease the metric (edge count divided by pair occurrences in the training set) for pairs of parts chosen incorrectly, leading to worse results. Then, we tried to increase the metric for pairs of parts that should have been chosen instead, which was beneficial.



After ten epochs, increasing the metric by 0.1% for the disregarded pair of parts for every incorrect instance, a graph accuracy of 93.2% for the training set and 92.1% for the validation set was achieved.

Learnings

- Throwing ML on an existing approach can greatly benefit the accuracy. However, we don't understand what the exact problem might have been in previous iterations
- During training, the same "weights" (metric values for pairs of parts) are increased and decreased since the edge is required in some graphs but not allowed in others. Again, there is too little context information to distinguish between these cases.

- When analyzing the graphs still created incorrectly, the problematic choices that lead to wrong edges are often repeating, regarding the parts and their neighbors.

We want to add context for previously incorrect decisions by **storing problematic edges** from the training set including the better alternatives and their neighbors:

If, while choosing an edge, this exact situation (with regards to the new part, the chosen existing part, and its neighbors) led to an incorrect graph during training, and the part that would have been the correct choice during training also exists, it decision is overwritten.

Although this leads to incorrect results for 1.5% of the graphs that were previously created correctly, it allows 35% of the previously incorrect graphs to be now generated correctly. After updating some outdated hyperparameters, the resulting graph accuracy is 96.5% and 93.5% for the training and validation set, respectively.

Learnings

- Compared to the previous approaches, we are now starting to overfit the training set.
- The same set of situations (with regards to all neighbors of the potential neighbors of a part) can lead to correct choices for hundreds of graphs while being the incorrect steps for hundreds of other graphs. At the same time, many of these individual situations only happen once or twice in training data set →. To make further distinctions with more context information useful, more data would be required.

In the following table, all improvements are listed again with their respective graph accuracy.

Feature	Achieved Graph Accuracy
Use edges in training set as instances	0.15% Train
Build trees sequentially using order	61.1% Train / 59.6% Val
"Edge-Probability" as metric	62.3% Train / 61.2% Val
Prohibit exceeding Maximum Degree	80.1% Train / 77.3% Val
Reach Average Degree	86.3% Train / 83.5% Val
Improve metric by learning	93.2% Train / 92.1% Val
Avoid known incorrect decisions	96.5% Train / 93.5% Val

2.3.2 The Final Approach

Given a set of parts, we try to build the correct graph containing these parts.

First, we order the parts based on each part's previously extracted average degree. If the part connects other high-degree parts, its average degree is artificially boosted to allow for this connection.

Now, we consider the first part of our graph and add all other nodes sequentially by always connecting the new part to exactly one of the existing parts. To choose the correct part, we determine the "edge-probability" for each possible pair: The "edge-probability" is the number of previously occurring edges between the two parts divided by the total occurrences of both parts together. It was artificially boosted if the "edge probability" was too low to create the target graph during training. Also, edges leading to parts getting closer to their average degree are rewarded, and parts overshooting the average or maximum degree are penalized slightly or strongly, respectively. The chosen existing part, all its neighbors, and the new parts are compared to the set of known incorrect decisions. If the edge is a known incorrect decision and the correct alternative is possible, it is chosen instead.

Results:

	Graph Accuracy	Edge Accuracy
Training set	96.543%	unknown
Validation set	93.548%	99.456%
Test set	92.891%	99.439%

We didn't calculate the edge accuracy for the training set because of a lack of computing resources. Again, the edge accuracy is hard to interpret since the difference of 0.017% could lead one to believe that the test set behaves as well as the validation set, but in reality, 0.657% fewer graphs are constructed correctly by the model. Due to the relatively manual approach and the amount of decisions based on the validation graph accuracy, we expected a more significant difference and were pleasantly surprised by the result.

References

[Car23] Carola Lenzen. Artificial intelligence. URL: https://digicampus.uni-augsburg.de/, 10 2023.

[ea20] Renjie Liao et al. Efficient graph generation with graph recurrent attention networks. *Conference on Neural Information Processing Systems*, 2020.