Graph Machine Learning for Assembly Modeling

Anonymous Author(s)
Anonymous Affiliation
Anonymous Email

Abstract

Assembly modeling refers to the design engineering process of composing assemblies (e.g., machines or machine components) from a common catalog of existing parts. There is a natural correspondence of assemblies to graphs which can be exploited for services based on graph machine learning such as part recommendation, clustering/taxonomy creation, or anomaly detection. However, this domain imposes particular challenges such as the treatment of unknown or new parts, ambiguously extracted edges, incomplete information about the design sequence, interaction with design engineers as users, to name a few. Along with open research questions, we present a novel data set.

1 Assembly Modeling

Assemblies are groups of parts that make up a product (see Figure 1). In computer-aided design (CAD), *assembly modeling* refers to designing a new product based on existing parts – think of a cabinet that consists of screws, doors and hinges, or a bike that consists of a frame, wheels, etc [1]. The connection type (e.g., welding or fastening using bolts) may contain geometric information or constraints that are also part of the assembly model. By its very nature, assembly modeling gives rise to a number of interesting novel applications for graph machine learning. Note that assembly modeling in this paper refers to the act of (iteratively) *designing* a new product using the same library of existing parts whereas other lines of work emphasize the computer vision perspective of *perceiving* physical parts (e.g. [2]) or the 3D perspective of *constraining* pairs of parts according to their position and relative movement (e.g., [3]) – also using geometric deep learning. Our goal is to support design engineers, e.g., by suggesting next parts to insert or categorizing the existing parts by their usage.

Some challenges that manufacturing companies face are:

- Assemblies similar to existing ones frequently need to be designed and adjusted – in accordance to customer specifications (e.g., in special mechanical engineering).
- Knowledge about proven part combinations (e.g., particular hinges and doors, screws and bolts, . . .) is available to senior design engineers and may follow a desirable part management but not made explicit and enforced in CAD software.
- Assembly models are produced in an *arbitrary sequence* which depends on the designer's individual preferences (e.g., start working on the front or back wheel of a bicycle is arbitrary); moreover, this insertion ordering is not stored in the final design by common CAD tools.
- Extracting a useful graph structure from CAD assembly models to begin with is not obvious. Although design engineers can define so-called "mates" relations between parts in a design to, e.g., define the rotation of a hinge, they are sometimes used for convenience in the CAD tool (cf. grouping elements) instead of actually denoting a physical connection or meaningful co-occurrence that could be reused.

In this extended abstract, we highlight opportunities for the graph machine learning community to work on CAD assembly modeling as a novel application along with an accompanying assembly data set [4]. Due to the symmetry properties in the data, graph ML is particularly suitable: as the CAD parts have no inherent order and their insertion sequence is not given, permutation invariance is crucial e.g., for part recommendation.

Submitted to the First Learning on Graphs Conference (LoG 2022). Do not distribute.

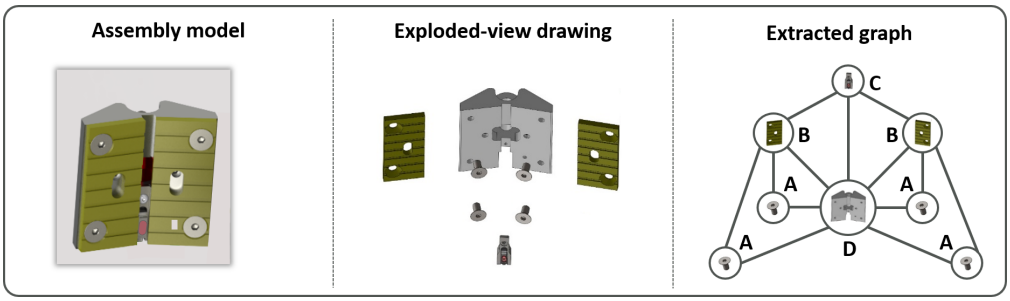

Figure 1: Assembly models (here, a jaw of a gripper) contain the structure of the included parts. Multiple instances of the same part type (here, A, B, C, D) may occur multiple times.

Formally, in our problem setting we assume a set of part types $\mathcal{P}$ (e.g., a particular type of screw, hinge, etc.) serving as vocabulary on which a data set of N assemblies $\{A_i\}_{i=1}^{N}$ is based. An assembly A_i specifies its containing parts (multiple instances of the same type are possible) as nodes $\mathcal{N}^{A_i}$, and information about connected parts as edges $\mathcal{E}^{A_i} \subseteq \mathcal{N}^{A_i} \times \mathcal{N}^{A_i}$. Each part $p \in \mathcal{N}^{A_i}$ is an instance of a part type, referred to as $\mathcal{T}(p) \in \mathcal{P}$. Consequently, each assembly is represented as an undirected, unweighted graph where the nodes are heterogeneous (different part types) and the edges are homogeneous (only one type of edges – expressing connectivity in a design – is allowed). The parts can be represented by an one-hot encoding or pretrained embeddings which may get extended in the future by additional features denoting geometric information, their material (steel, aluminium, or plastics), or properties of the parts such as conductivity or temperature resistance.

2 Graph ML Use Cases in Assembly Modeling

Given a data collection of assemblies as described, there are several application scenarios for graph machine learning. With respect to the distinction into structural (graph structure is explicit as in molecule generation) and non-structural scenarios (graphs are implicit and derived from text or images) given in [5], the proposed applications fall into a "semi-structural" category. The extraction of edges is a vital challenge of this application scenario as some CAD mates can be extracted canonically whereas others may have been defined by designers out of convenience with no actual meaning, so that some edges may need to be extracted from, e.g., geometric proximity.

2.1 Part Recommendation

In assembly modeling, design engineers can choose from a variety of existing parts, making selecting the right ones a cumbersome task. Past assemblies contain information both on the collection of used parts and on their combination to solve a specific task. We assume that parts which are used together frequently are causally related and, therefore, parts that are likely to be inserted can be predicted using graph machine learning. In previous work [6], recommending next required parts during construction based on GNNs has already been investigated showing promising results – i.e., learning $P(\mathcal{P} \mid A_i^t)$ where A_i^t refers to the state of an assembly at time step t. For the gripper jaw illustrated in Figure 1, next needed parts could be a handle and a screw to continue the construction of the entire gripper. Formulated as a graph classification problem where each class corresponds to a part type, the top-k rate may be used as performance metric.

However, the presented approach does not incorporate *where* to assemble the recommended part to the partial design. This may be acceptable for small assemblies, but becomes unwieldy for large assemblies with plenty of possible extension nodes. A natural next step is to predict applicable parts for every already added part of the assembly – i.e., learning $P(\mathcal{P} \mid c \in A_i^t)$ to localize the part recommendation within an assembly. Due to this autoregressive nature of part recommendation, it bears similarity to graph generation although the focus is on incremental steps with user interaction as opposed to learning an assembly distribution $P(A)$ all at once, as will be discussed in Section 3. Since the main goal of part recommendation is to provide a design engineer with a small set of relevant parts to reduce the cognitive burden, approaches using conformal prediction [7] could prove useful.

2.2 Anomaly Detection of Mismatching Parts

A second (unsupervised) use case consists of detecting anomalies in assembly models such as an unexpected choice of particular part types (e.g., screws from a different manufacturer) or rarely used substructures that could hint at an unconventional way of solving a design task. From a business perspective, companies might limit their procurement to a set of well-known part types (better contracts with manufacturers, more reliable during the product life-cycle) within their strategic part management. Anomalous assembly models might emerge, e.g., from starting a new model based on a much earlier project with some part types having become obsolete or simply a lack of experience/knowledge on behalf of the design engineer. From a graph ML perspective, both identifying anomalous graphs in a database as well as identifying anomalous graph objects (nodes, edges such as, e.g., unexpected part types for the screws in the gripper in fig. 1) needs to be addressed [8], in particular to show to design engineers or procurers *where* and *how* the assembly is deviating.

2.3 Creating a Taxonomy of Parts

Third, using node embeddings h_p of the parts $\{p \in A_i\}_{i=1}^{N}$ such as node2vec [9], DeepWalk [10], or comp2vec [6] for visualization and clustering could aid companies as well in their part management. The availability of well-curated, hierarchical taxonomies of part types depends on the level of maturity of a company and traditionally requires significant manual effort. A data-driven solution that exploits usage patterns in assembly models could organize a company's frequently used part types better. There has been an interest in making these embeddings (or latent representations) of nodes more interpretable to humans [11] which is what needs to be done for this task. However, the graphs retrieved from assemblies do not show homophily – two parts that are connected are most likely *not* similar but rather complementary (e.g., door and hinge) – which is an underlying assumption of many existing node embedding techniques. Here, synonymity in terms of usage (on which comp2vec is based) tends to be a better replacement, i.e., parts are similar if they can be used in the same contexts.

3 Related Work

The task of part recommendation bears some similarities to graph generation, i.e., approximating $P_{\text{data}}(G)$ with a parametrizable $P_{\text{model}}(G \mid \theta)$. The graph can be generated either all at once, for example using variational autoencoders or generative adversarial networks, or incrementally by so-called *autoregressive* models that predict single or multiple nodes or edges step by step – conditioned on an intermediate state of the graph. Since our goal is to support design engineers by presenting suggestions instead of taking over the whole task, we go with the second approach. During CAD modeling, we want to allow changes on the partial assembly by designers. Therefore, we need a model that can generate arbitrarily large graphs which is typically not the case for non-incremental models.

Generating graphs with matching structural characteristics to the training data along with handling only one node type (i.e., $|\mathcal{P}| = 1$) as done using recurrent neural networks in GRAN [12] or GraphRNN [13] is not sufficient for our use case: the relevant information for predicting next needed parts lies both in the already used parts, i.e., the node features, and the structure of the graph. The focus is mainly on the type of part that should be added and only secondarily where to insert it into the existing graph. Since these approaches only evaluate the final graph structure (in particular, in terms of aggregated graph statistics such as degree distributions) without incorporating its intermediate states, canonical numbering of nodes can be performed for generating training instances, keeping their number small as no node permutations need to be considered. Common choices for GRAN or GraphRNN are breadth-first or depth-first traversals starting from the most connected node or random orderings. For assemblies, however, designers can start with any part or subgraph, followed by a generation sequence depending on the designer's preferences. Therefore, the authors in [6] create instances for every possible creation sequence of an assembly by iteratively cutting off nodes that serve as labels for the resulting partial assemblies. Unlike [14], in these approaches newly added nodes are always connected to the previous graph structure, which we want to enforce during construction.

Molecule graph generation refers to generating valid molecules with desired chemical properties, incorporating various types of nodes (i.e. different atoms) and even various types of connections (which is not necessary for using the current representation of assemblies). While guaranteeing the validity of the generated graph (like in [15] concerning the chemical structure of the generated molecule) may be assumed to be an important aspect for assembly modeling as well, this check-up

turns out be not this obvious as the number of connection points of an part is typically not available or misleading since design engineers may adapt their geometry, e.g., by drilling holes, in order to assemble additional parts. However, this application domain seems to be the most similar to assembly modeling in terms of data representation. Nevertheless, again only the final generated graphs are relevant for evaluating the molecule generation model – as expounded above, also the intermediate steps matter for part recommendation.

4 Open Questions

The domain of assembly modeling imposes particular challenges that can stipulate further research in graph machine learning as described in detail in the following.

How to deal with evolving data sets? Over time, the part catalogs may get updated as well as new catalogs and part types may be incorporated to a company's part library. In particular at test time, we might be confronted with part types in assemblies that were not available during training. This setting confronts us with so-called *attribute-missing* graphs where all attributes of a subset of nodes are missing, opposed to *attribute-incomplete* graphs [16] that are composed of nodes all with non-empty attribute sets, typically treated by value imputation techniques either in a preprocessing step (e.g., [17] or [18]), or during processing the graph in the model (e.g., [19] or [20]). Methods based on the homophily assumption are not applicable since the assumption of connected nodes been similar is clearly violated in the assembly modeling use cases as connected parts typically serve different purposes. Initial work has been done on handling attribute-missing graphs, e.g., [16] that make a shared-latent space assumption on graphs resulting in a new form of GNN called SAT – its applicability on assembly modeling needs to be investigated.

How to handle ambiguous edges? As introduced in Section 1, extracting a graph structure from an assembly model is an ambiguous task as only some mating relations may be given in the CAD system – some of them even serving other purposes than denoting meaningful connections (e.g., to simply support the designer's workflow). Consequently, extracting a graph structure from an assembly is not straightforward and when based on geometric proximity of parts an computational expensive approach. Even in a perfect world, where all parts are connected by mates in a meaningful way, this graph structure may be insufficient for the learning task (as mentioned in [21] and [22]) because parts that are far away from each other according the graph structure may have a certain relationship which is relevant for recommending next parts. *Graph rewiring* may be a promising solution for this issue, transforming the initial graph structure by adding and removing edges to improve information processing. Moreover, due to the novelty of the application domain, it is unclear whether a graph structure is really helpful for solving assembly modeling tasks, possibly a set-based approach could perform as well – this can also be investigated using graph machine learning.

How to improve intuitive sequence generation and interactive inference? Especially for part recommendation, the proposed sequence of part insertions needs to be intuitive in the eye of the designers that interact with the assembly modeling tool. There is not an obvious way to extract a sequence from a data set of graphs – as is done in generative graph models such as GraphRNN or GRAN. Either *all* possible insertion sequences (that leave the assembly connected) or a sample thereof need to be considered – as done in [6] – or the data sets need to be augmented with insertion sequences.

Finally, we encourage readers to investigate the data set [4] and identify similarities with their preferred data sets or applicability of their methods that can address the above challenges. It contains graph-based pseudonymized real-world assemblies (cf. Figure 1) and prepared samples for part recommendation consisting of partial assemblies and next needed parts, allowing to perform all presented use cases.

References

[1] Stephen J Schoonmaker. *The CAD guidebook: A basic manual for understanding and improving computer-aided design*. CRC Press, 2002. 1

[2] Guanqi Zhan, Qingnan Fan, Kaichun Mo, Lin Shao, Baoquan Chen, Leonidas J Guibas, Hao Dong, et al. Generative 3d part assembly via dynamic graph learning. *Advances in Neural Information Processing Systems*, 33:6315–6326, 2020. 1

[3] Benjamin Jones, Dalton Hildreth, Duowen Chen, Ilya Baran, Vladimir G. Kim, and Adriana Schulz. AutoMate: A Dataset and Learning Approach for Automatic Mating of CAD Assemblies. *ACM Trans. Graph.*, 40(6), dec 2021. ISSN 0730-0301. doi: 10.1145/3478513.3480562. URL https://doi.org/10.1145/3478513.3480562. 1

[4] Carola Gajek. ECML22 GRAPE Data. 7 2022. doi: 10.6084/m9.figshare.20239767.v1. URL https://figshare.com/articles/dataset/ECML22_GRAPE_Data/20239767. 1, 4

[5] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020. ISSN 2666-6510. doi: https://doi.org/10.1016/j.aiopen.2021.01.001. 2

[6] Carola Gajek, Alexander Schiendorfer, and Wolfgang Reif. A Recommendation System for CAD Assembly Modeling based on GNNs. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2022. 2, 3, 4

[7] Glenn Shafer and Vladimir Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(3), 2008. 2

[8] Xiaoxiao Ma, Jia Wu, Shan Xue, Jian Yang, Chuan Zhou, Quan Z Sheng, Hui Xiong, and Leman Akoglu. A comprehensive survey on graph anomaly detection with deep learning. *IEEE Transactions on Knowledge and Data Engineering*, 2021. 3

[9] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016. 3

[10] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014. 3

[11] Ayushi Dalmia and Manish Gupta. Towards interpretation of node embeddings. In *Companion Proceedings of the The Web Conference 2018*, pages 945–952, 2018. 3

[12] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, William L. Hamilton, David Duvenaud, Raquel Urtasun, and Richard Zemel. *Efficient Graph Generation with Graph Recurrent Attention Networks*. Curran Associates Inc., Red Hook, NY, USA, 2019. 3

[13] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. GraphRNN: Generating realistic graphs with deep auto-regressive models. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5708–5717. PMLR, 10–15 Jul 2018. 3

[14] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning Deep Generative Models of Graphs, 2018. 3

[15] Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. Graphaf: a flow-based autoregressive model for molecular graph generation. 2020. doi: 10.48550/ARXIV.2001.09382. 3

[16] Xu Chen, Siheng Chen, Jiangchao Yao, Huangjie Zheng, Ya Zhang, and Ivor W. Tsang. Learning on attribute-missing graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(2):740–757, 2022. doi: 10.1109/TPAMI.2020.3032189. 4

[17] Joonyoung Yi, Juhyuk Lee, Kwang Joon Kim, Sung Ju Hwang, and Eunho Yang. Why not to use zero imputation? correcting sparsity bias in training neural networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. 4

[18] Indro Spinelli, Simone Scardapane, and Aurelio Uncini. Missing data imputation with adversarially-trained graph convolutional networks. *Neural Networks*, 129:249–260, 2020. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2020.06.005. 4

[19] Hibiki Taguchi, Xin Liu, and Tsuyoshi Murata. Graph convolutional networks for graphs containing missing features. *Future Generation Computer Systems*, 117:155–168, 2021. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2020.11.016. 4

[20] Jiaxuan You, Xiaobai Ma, Yi Ding, Mykel J Kochenderfer, and Jure Leskovec. Handling missing data with graph representation learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 19075–19087. Curran Associates, Inc., 2020. 4

[21] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications, 2020. 4

[22] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. *ArXiv*, abs/2111.14522, 2022. 4