2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Speeding Up Assembly Sequence Planning Through Learning Removability Probabilities

Alexander Cebulla, Tamim Asfour, and Torsten Kröger

***Abstract*— Industry 4.0 facilitates a high number of product variants, posing significant challenges for modern manufacturing. One of them is the automatic creation of assembly sequences. This can be achieved with the assembly-by-disassembly (AbD) approach, which is currently highly inefficient. We aim at speeding up AbD by leveraging deep learning. AbD relies on iteratively testing parts for removal, which makes the order in which parts are tested highly relevant for its run-time. We optimize this order by training a graph neural network (GNN) based on the shape of parts and the shape of local part connections. For each part, it predicts a removability probability. We use these probabilities to optimize the order in which parts are tested for removal. This reduces the number of parts tested by approximately 64%–90%, depending on the tested product. Further improvements are achieved by combining our approach with bookkeeping, another approach for speeding up AbD. Finally, we separately analyze the impact of the parts and their connections on the removability probabilities predicted by the GNN. We found that most of the important information regarding a part's removability can be derived from its connections alone.**

I. Introduction

The manufacturing of customized product variants, as facilitated by industry 4.0, poses a number of challenges to the production process. A crucial one is assembly sequence planning (ASP), which refers to finding a suitable sequence of assembly steps to put the final product together. Traditionally done by hand, this process needs automation to handle the increasing number of product variants. An existing approach for automating ASP is assembly-by-disassembly (AbD). This approach tries to find a suitable assembly sequence by finding a disassembly sequence for the fully assembled product and inverting it. Without further improvements, AbD is very inefficient, as it iteratively tests, in simulation, for each part if it can be removed. In the worst case, this can lead to $(n-1)+(n-2)+\cdots \in \mathcal{O}(N^2)$ removal attempts for an assembly with N parts, hence becoming highly time-consuming [1].

To speed up AbD, it is crucial to reduce the number of removal attempts, either by reducing unnecessary ones and/or by optimizing the order in which parts are tested for removal. An existing approach for reducing unnecessary removal attempts is a bookkeeping heuristic [2]. It keeps track of parts that could not be removed in previous attempts and only tests them again if there is a chance that they have become removable. However, the order in which parts are tested remains the same. This leaves room for complementary approaches aimed at optimizing the part order.

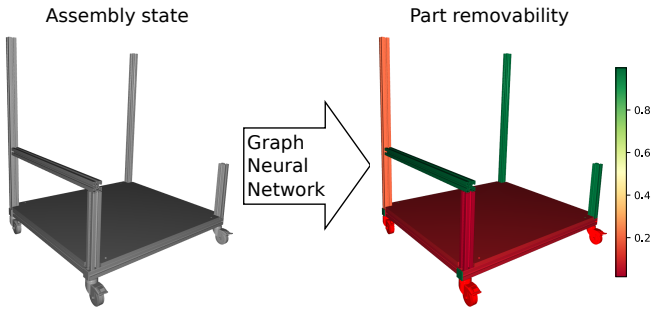


Fig. 1: Goal of our approach: Predict for an assembly the removability probabilities of its parts.

We propose such a part order optimization by employing a graph neural network (GNN) to learn and predict the removability probabilities of individual parts in an assembly as shown in Fig. 1. Optimization is performed by reordering the parts according to their removability. We leverage that products often share similarly designed sub-assemblies as well as that a part's removability depends on its type and the presence or absence of parts in its local environment. We assume that these similarities extend to the removability of individual parts, meaning that similar parts in similar environments have similar removability. This relationship can be learned by GNNs, as parts and their connections can be represented through graphs, where nodes correspond to parts and edges to their connections.

Throughout this paper, we first introduce how we derive data suited for GNN training by transforming assemblies into a graph representation. Next, we describe our GNN approach and its incorporation in an existing AbD framework [1]. With our experiments we first validate our assumption: for products using similar designs, we are able to predict a part's removability with over 85% accuracy. Then, we investigate how our approach can improve an existing AbD framework [1]. An additional comparison between our approach and bookkeeping [2] is provided, together with a combination of both approaches.

In summary, our main contributions are:

1) Development of a graph-shaped representation for assemblies.
2) The construction of a dataset using this representation.
3) Development and training of a GNN for predicting part removability probabilities.
4) Speeding up AbD by testing parts in order of their predicted removability.

The research leading to these results has received funding from the Carl Zeiss Foundation.

Institute for Anthropomatics and Robotics - Intelligent Process Automation and Robotics Lab (IAR-IPR), Karlsruhe Institute of Technology (KIT) {alexander.cebulla, asfour, torsten}@kit.edu.

II. Related Work

An important aspect of AbD is how parts are tested for removal. Several strategies for these tests exist. The mating vector approach tries to remove each part along pre-computed directions. These might be parallel to the coordinate axes [3] or are extracted from the parts' surface normals [1]. In a more sophisticated approach [4], they are derived from analyzing the configuration space obstacles for each part. Testing removability via mating vectors is fast, however, it fails if a part cannot be removed along a linear path. An alternative is to search for paths via sampling-based motion planers [1], [5] such as Rapidly-exploring Random Trees (RRT) [6]. The 'Expansive Voronoi Tree' algorithm [7] is another approach that first computes a general Voronoi diagram for the whole assembly, which is then used to efficiently compute removal paths for each of its parts.

All of the work discussed so far focuses on how to efficiently test if a part can be removed. Another important aspect of AbD is how to select a part for testing. In [2] a bookkeeping heuristic is suggested. This is done by keeping track of parts for which no removal path was found during previous removal attempts. For these parts, the approach assumes that they are surrounded by other parts blocking them. Therefore, they are only tested again, if at least one of the parts they collided with during previous removal tests has been removed. Additionally, a wide range optimization algorithms has been applied to ASP [8] with the goal to find a sequence that minimizes criteria such as tool changes. Examples are genetic algorithms [9], particle swarm optimization [10], and deep reinforcement learning [11]. However, all these methods optimize a specific assembly.

In contrast, approaches based on case-based planning (CBP) re-use assembly sequences of sub-assemblies from related objects. For example, [12] retrieved assembly plans for sub-assemblies from a database based on similarities between graphs of part connections, mating directions as well as mating constraints. In [13] a genetic algorithm was used to first find correspondences between parts of a new assembly and a reference assembly. Based on these correspondences, it then generates a reference assembly sequence. In [14], CBP was used to first generate assembly sequence candidates which were then evaluated based on geometric constraints extracted from CAD models. Closest to us are the works [15] and [16]. In [15] – similar to our approach – a graph-based representation with nodes that encode parts via spherical harmonic features [17] was used. However, to encode connections between parts, collinear and coplanar relations between them were computed. This requires manual work of a human to define the axis and plane elements for each part. Based on the similarities between their representations, assembly sequences were then transferred from a reference assembly to the new assembly. The assumption was made that all assemblies are of the same type – in this case, chairs. In [16] graph-shape topologies were specified for a well-defined set of assemblies. These assemblies consisted of three or four aluminium profiles, and various assemblies could have the same topology. Through clustering, a set of typical topologies was then identified. Furthermore, each topology defined a sequence of features that determined a set of assembly rules. Then, for a new assembly, first its topology was computed and the corresponding sequence of features was extracted. Finally, a classifier was used to determine the correct set of assembly rules.

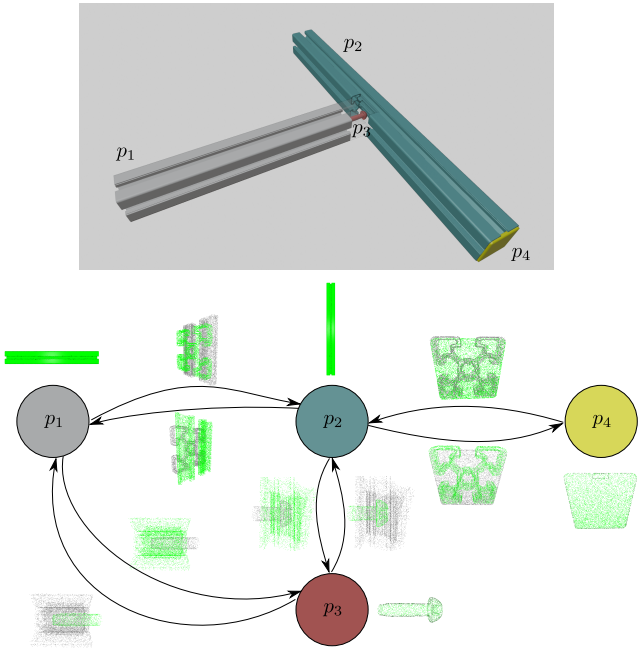


Fig. 2: Representation of a 3D assembly model with four parts as graph. Not shown is the computation of the 3D shape descriptors from the extracted point clouds.

Compared to [15], our approach does not rely on human input at all, and the product assemblies are broader in perspective: They belong to different product groups and only share the property that they are constructed from aluminum profiles together with accompanying parts. In contrast to [16], we work on complex real-world assemblies adapted from [18], [19], [20], [21], [22] with up to 68 parts instead of small assemblies using only 3-4 aluminum profiles.

III. Assembly Representation

Our primary objective for improving ASP via AbD is to preferably test parts for removal that have a high removability probability. We assume, that a part's removability primarily depends on the type of part and its local neighborhood, and not the specific assembly it is employed in. Therefore, we propose the use of a graph-based assembly representation, which explicitly encodes these factors: the individual parts are represented as nodes, while their connections with other parts are represented as directed edges as shown in Fig. 2. In the following, we first give a formal definition of an assembly and show how it can be represented as a graph. Then, we introduce how we calculate node and edge attributes for the graph from an assembly.

A. Formal Definition of an Assembly

An assembly is defined by the set of its parts $\mathcal{A} := \{p_1, p_2, \dots p_N\}$ as well as their connections $\mathcal{C} := \{\mathcal{C}_{p_1}, \mathcal{C}_{p_2}, \dots \mathcal{C}_{p_n}\}$. Each part p_i is defined by its geometric shape and 6D pose. The connections $\mathcal{C}_{p_i}$ associated with p_i include all parts p_j that share a physical connection with p_i. A connection between p_i and a potential neighbor p_j exists, if the minimal distance between both parts is smaller than a threshold ϵ. Formally, let X_{p_i} and X_{p_j} be sets of surface points on p_i and p_j respectively, then the connections $\mathcal{C}_{p_i}$ of p_i are given by

$$\mathcal{C}_{p_i} := \{p_j | p_j \in \mathcal{A}, \min_{x_i \in X_{p_i}, x_j \in X_{p_j}} ||x_i - x_j||^2 < \epsilon\}$$

B. Representing Assemblies as Graph

We represent an assembly $\mathcal{A}$ as directed graph $G_{\mathcal{A}} = (N_{\mathcal{A}}, E_{\mathcal{A}})$, with the node set $N_{\mathcal{A}}$ and the edge set $E_{\mathcal{A}}$. Thereby, each part $p_i \in \mathcal{A}$ is represented by a node $n_i \in N_{\mathcal{A}}$, while connections between parts p_i and p_j are represented by directed edges $e_{i,j} \in E_{\mathcal{A}}$ connecting the nodes. Formally, a connection $e_{i,j} \in E_{\mathcal{A}}$ exists exactly when $p_i, p_j \in \mathcal{A}$ and $p_j \in C_{p_i}$. In the following, the subscript for G_A will be omitted if it is clear which assembly the graph represents.

C. Calculation of Node and Edge Attributes

Next, a suitable representation for nodes and edges in the graph is needed. Given our assumption that a part's removability primarily depends on the part itself and its connections, such a representation should be invariant to scale, translation and rotation. Through these invariances, it is no longer important in which way and size a part is employed in an assembly - the resulting representation will always be the same, keeping the focus on the part and its connections. To obtain such a representation, we employ 3D shape histograms [23] with a spherical harmonic representation [17]. This approach works based on point clouds of the parts. First, the point cloud is centered and re-scaled to the unit cube to achieve and translation and scale invariance. To obtain rotation invariance, spherical harmonics are utilized next. Therefore, the re-scaled and centered point cloud is transformed into spherical coordinates, which characterize each point by a radius r and two angles θ, φ. For these values, a 3D histogram is calculated, where each bin is characterized by a combination of value-ranges for r, θ, φ. Through fixing a value-range for radius r we can obtain a discrete spherical function $f(\theta, \varphi)$ from the 3D shape histogram. Further, according to spherical harmonics theory each spherical function can be decomposed as the sum of its harmonics [17]:

$$f(\theta, \varphi) = \sum_{\ell=0}^{\infty} \sum_{m=-\ell}^{\ell} a_\ell^m Y_\ell^m(\theta, \varphi), \tag{1}$$

where $Y_\ell^m(\theta, \varphi)$ are the spherical harmonic basis functions of degree ℓ and order m and a_ℓ^m are coefficients computed through projecting $f(\theta, \varphi)$ onto these basis functions. An important property of spherical harmonics is that for a fixed degree ℓ the L_2 norm of the coefficient vector is invariant against rotating the spherical function $f(\theta, \varphi)$. Therefore, through computing the norm of these vectors for every radius bin, i.e., every value-range for radius r, and then concatenating all norms, we obtain a rotation-invariant shape descriptor $\boldsymbol{h} \in \mathbb{R}^D$. We used six radius bins and 30 bins each for θ and φ, resulting in $D = 90$.

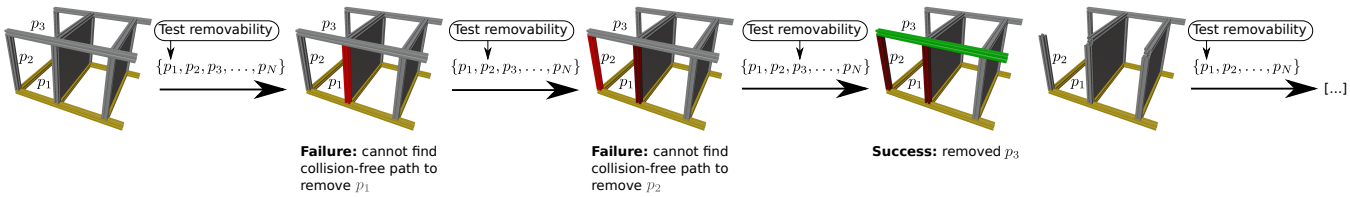
To be independent of manual annotations, we want to automatically extract node attributes $\boldsymbol{a}_n \in \mathbb{R}^D$ and edge attributes $\boldsymbol{a}_e \in \mathbb{R}^{2D}$ from an assembly. This can be achieved by using the aforementioned shape descriptor $\boldsymbol{h}$. Specifically, each node n_i will store the 3D shape descriptor, computed for the point cloud X_i of its corresponding part p_i, as its attribute $\boldsymbol{a}_{n_i} := \boldsymbol{h}_{n_i}$. For edge attributes, as shown in Fig. 2, the contact areas between parts will be used. In detail, for edge $e_{i,j}$ and fixed distance threshold ϵ, we define $X_i^j = \{x_i \in X_{p_i} | \exists x_j \in X_{p_j} : ||x_i - x_j|| < \epsilon\} \subseteq X_{p_i}$ as the points of p_i that are in contact with p_j. X_j^i is defined accordingly. Then, $\boldsymbol{h}_i^j$ and $\boldsymbol{h}_j^i$ are the 3D shape descriptors computed for X_i^j and X_j^i. The edge attribute for the directed edge $e_{i,j}$ is then given by the vector $\boldsymbol{a}_{e_{i,j}} := \left[\boldsymbol{h}_i^j, \boldsymbol{h}_j^i\right]$ and for $e_{j,i}$ it is $\boldsymbol{a}_{e_{j,i}} := \left[\boldsymbol{h}_j^i, \boldsymbol{h}_i^j\right]$, respectively.

IV. Speeding up ASP via AbD

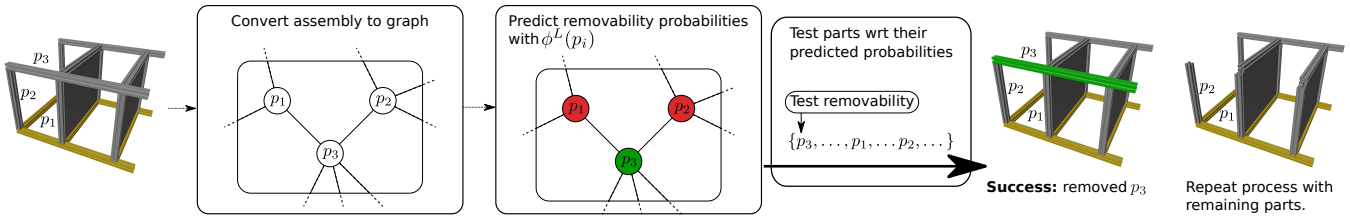
The basic idea behind ASP via AbD is to find a suitable assembly sequence by first disassembling the product. As shown in Fig. 3a, it starts from a fully assembled product and then tests for each part if it can be removed without collision and if potential additional constraints (e.g. for gravity) are fulfilled. This creates a disassembly sequences, which is inverted to get an assembly sequence (assuming the sequence is invertible). While general, this approach suffers from a combinatorial explosion. That is, the number of removal attempts is in $\mathcal{O}(N^2)$, where N is the number of parts. We propose to mitigate this problem through reducing the number of removal tests. In detail, for an assembly represented as graph as described in Sec. III, we use a GNN to predict the probability of parts being removable. Then, parts are tested according to their removability from high to low. The whole process is illustrated in Fig. 3b In the following, we first describe a general AbD framework [1] and how we used it to create our dataset. Then, we describe the GNN architecture. Finally, we show how we used a GNN to intelligently reordering parts before testing them for removability.

A. General AbD Framework

We use [1] as general AbD framework. It suggests a general workflow, but leaves the implementation of its individual steps open, see below. The idea is to organize the search for a disassembly sequence through constructing a directed disassembly graph DG. Thereby, its root node contains the fully-assembled product. Starting from the root, the graph is then iteratively expanded with new nodes, where each node contains a (sub-)assembly of the original assembly. Specifically, from a node to its child node, exactly one part is



(a) **Vanilla AbD:** Test the parts of assembly $\mathcal{A} := \{p_1, p_2, p_3, \ldots, p_N\}$ for removability in an *arbitrarily pre-defined order*.



(b) **Our approach:** Test parts according to their predicted removability probabilities ϕ^L.

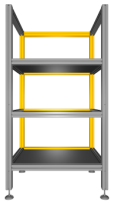
Fig. 3: Comparison between **vanilla AbD** and **our approach**. Parts highlighted in yellow are fixated in the xy-plane, with gravity acting along the $-z$-axis, and act as a stable base.

(a) Workbench (68 parts) [18]

(b) Lectern (59 parts) [19]

(c) Whiteboard (29 parts) [20]

(d) Cart (33 parts) [21]



(e) Shelf (53 parts) [22]

Fig. 4: The used real-world assemblies adapted from the given source. Parts highlighted in yellow are fixated in the xy-plane, with gravity acting along the $-z$-axis, and act as a stable base. They are not included in the number of parts.

removed from the sub-assembly. Child nodes are only added, if a part is removable. In the end, the leaf nodes of the graph contain empty sub-assemblies and each path from the root to a leaf node describes a valid disassembly sequence. The framework allows to flexibly implement the following steps:

1) *The strategy used to search through the disassembly graph DG.* Search strategies suggested in [1] are depth-first search (DFS) or breadth-first search (BFS).
2) *The order, in which assembly parts of a selected node are tested.* In [1], an arbitrarily pre-defined order is used.
3) *The used removal test.* [1] suggested to use two tests: first, all parts are tested via mating vectors (fast removal tests), then via RRT [6] (slower removal tests).

We make the following choices: 1) We use DFS, because we want to find a single suitable disassembly sequence as fast as possible. 2) We use a GNN to predict the removability probability for each part in a node's sub-assembly. Then we reorder the parts for testing according to these probabilities from high to low to optimize the part order. In contrast, [1] used an arbitrarily pre-defined order – we will refer to this as vanilla AbD. 3) We follow the suggested approach and first test via mating vectors, and, if no removable part is found, with RRT. As a further improvement, we only use RRT on parts that could be moved at least along one mating vector.

As an additional constraint for a part being removable, we employ a simple gravity constraint. This is done to account for the stability of an assembly. We assumed that each assembly is build upon a fixed frame of profiles, which is highlighted yellow in Fig. 3. Each remaining part must be directly or indirectly (over a chain of physically connected parts) in contact with a part belonging to this frame.

B. Dataset

For comparing our approach against the vanilla AbD approach, we used five real-world assemblies shown in Fig. 4. Additionally, these assemblies were also used to construct a dataset $\mathcal{D} = \{(\mathcal{A}_i, l_i) | i \in 0, 1, \ldots, N\}$ for training and evaluating our GNN. Hereby, $\mathcal{A}_i$ denotes the set of parts contained in the sub-assembly and l_i the labels associated with the parts in $\mathcal{A}_i$. 1 is used as label if the part is removable, 0 otherwise. To obtain valid sub-assemblies and labels, we use the vanilla AbD framework as described in Sec. IV-A with the following alteration: For each state, all contained parts are tested for removal to obtain l_i. Then, the part that is actually removed is randomly chosen to form the next disassembly state.

C. GNN architecture

GNNs are designed for processing graphs with variable node and edge count [24]. They employ layers with shared

weights for nodes as well as layers with shared weights for edges. For each layer l, a GNN Φ^l takes a graph $G^l = (N^l, E^l)$ as input, applies various transformations and outputs another graph $G^{(l+1)} = (N^{(l+1)}, E^{(l+1)})$ of the same shape with updated attributes. Formally:

$$G^{(l+1)} = \Phi^l \left(G^l\right), \forall l \in [0, L], \tag{2}$$

where $L+1$ the total number of layers. Updating node and edge attributes in layer l consists of three steps: First, the edge attributes are updated, using the edge attributes plus the attributes of the two adjacent nodes as input for a feed-forward neural network (FNN). Next, the node attributes are updated in a two-step process: First, attributes from all adjacent edges and nodes are aggregated. Second, these aggregated attributes, together with the node attributes, are processed by another FNN. Formally,

$$\boldsymbol{a}_{e_{i,j}}^{(l+1)} = \phi_e^l(\boldsymbol{a}_{n_i}^l + \boldsymbol{a}_{e_{i,j}}^l + \boldsymbol{a}_{n_j}^l) \tag{3}$$

$$\bar{\boldsymbol{a}}_{n_i}^{(l+1)} = \sigma^l \left(\{\text{ReLU}\left(\boldsymbol{a}_{n_i}^l + \boldsymbol{a}_{e_{i,j}}^{(l+1)}\right) + \epsilon | e_{i,j} \in E_{\mathcal{A}}\}\right) \tag{4}$$

$$\boldsymbol{a}_{n_i}^{(l+1)} = \phi^l \left(\boldsymbol{a}_{n_i}^l + \bar{\boldsymbol{a}}_{n_i}^{(l+1)}\right), \tag{5}$$

where the exponent l denotes the layer. ϕ_e^l and ϕ^l denote layer-specific FNNs, each consisting of two layers with 64 neurons and ReLU activation functions. The same networks are applied to all edges/layers of l. Last, σ denotes a learned *SoftMax* aggregator, which was shown [25] to perform better than other aggregation methods such as *mean()* or *max()*. The GNN contains two layer-specific adjustments: In the first layer, a linear transformation is applied to the edge attributes to align their channel count with the node attributes. In the last layer L, ϕ^L produces only a single output using the sigmoid activation function. We interpret this output as the removability probability of the part p_i represented by the respective node n_i, ranging from 0.0 (not removable) to 1.0 (removable). The GNN output for node n_i will be denoted as n_i^L in the following. For training, the binary cross-entropy loss together with the ADAM [26] optimizer is used.

D. Intelligently Reordering Parts for Removal Tests

To speed up the AbD process, we first train a GNN Φ on a dataset $\mathcal{D} = \{(\mathcal{A}_i, l_i) | i \in 0, 1, \ldots, N\}$ collected as described in Sec. IV-B. Then, as shown in Fig. 3b, given a new assembly $\mathcal{A}$ we first convert it into the graph representation $G_{\mathcal{A}} = \{N_{\mathcal{A}}, E_{\mathcal{A}}\}$ as described in Sec. III. Next, $G_{\mathcal{A}}$ is passed through Φ to obtain the removability probability n_i^L for each node $n_i \in N_{\mathcal{A}}$. Finally, the parts $n_i \in N_{\mathcal{A}}$ are ordered according to their removability probability from highest to lowest, before they are tested for removability until a removable part is found and removed. This results in a new sub-assembly with its own graph representation. The process is repeated with each new sub-assembly, until a full disassembly sequence is found.

V. Experiments

Our experiments aim to answer three questions: (i) Is the proposed graph representation in combination with a GNN suitable for inferring part removability? (ii) How does our approach compare to vanilla AbD as proposed in [1] and to AbD with a bookkeeping heuristic [2]? (iii) Does our approach yield further improvements when combined with a bookkeeping heuristic?

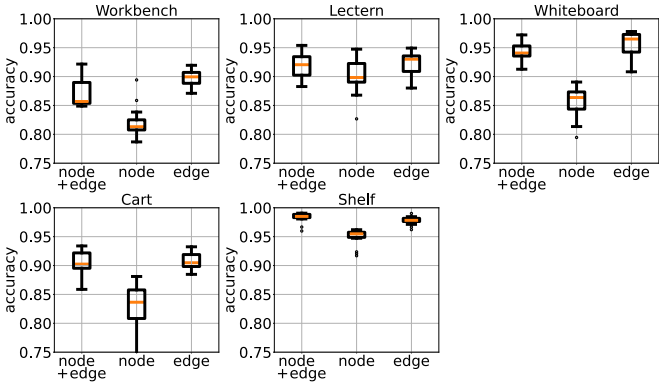


Fig. 5: Prediction accuracy averaged over ten runs for GNNs trained on four assemblies and tested on the fifth.

We answer these question through two sets of experiments. The first one addresses the impact of different parts of the graph representation on the final removability prediction. The second one evaluates our approach with and without bookkeeping against the AbD approach proposed in [1] with and without bookkeeping. For both, a dataset is created as described in Sec. IV-B. For each assembly, we start fully assembled and stop when it is fully disassembled. This is repeated 20 times per assembly to form the final dataset.

We used Python and relied on the Open3D [27] library for handling meshes, the Flexible Collision Library [28] to test for part collisions, and on PyTorch Geometric [29] for implementing GNNs. All experiments were performed on Ubuntu with an Intel Core i9-10980XE, 64 GB of memory, and a GeForce RTX 2080 Ti.

A. Assembly Representation Analysis

We performed experiments to evaluate the importance of node and edge attributes of our assembly representation for learning part removability probabilities. The GNN was trained for 50 epochs and evaluated using five-fold cross-validation, i.e., each assembly was evaluated using a GNN trained on the other four assemblies. For each assembly, we repeated this 10 times. We evaluated the performance using averaged prediction accuracy acc $= \frac{\text{\#correct prediction}}{\text{\#all samples}}$. A part is removable, if its predicted removability probability was > 0.5. The results are shown Fig. 5. It can be seen that the node attributes contain useful information regarding removability, achieving > 0.8 median accuracy for all experiments. However, they are outperformed by the edge attributes as well as the the combined node + edge attributes. This was expected, given that there are various scenarios that the classifier cannot differentiate based on node attributes alone, e.g., when the same parts are connected differently. In addition, one can see that most of the information required to learn removability probabilities is stored in the edge

TABLE I: Comparision of the total disassembly time and number of removal tests required by the AbD framework described in Sec. IV-A without (**Vanilla**) and with (**BK**) a bookkeping heuristic [2] against our approach described in Sec. IV-D without (**GNN**) and with (**GNN + BK**) a bookkeping heuristic.

		Time (s) (mean + std. over 10 runs)	Number of removal tests (mean + std. over 10 runs)		
			Total	Failed (collision)	Failed (gravity)
Work-bench	**Vanilla**	816 ± 90	1153 ± 107	1032 ± 113	53 ± 16
	BK	439 ± 75	604 ± 46	483 ± 45	53 ± 16
	GNN	310 ± 34	420 ± 47	344 ± 46	8 ± 2
	GNN + BK	224 ± 28	281 ± 30	205 ± 29	8 ± 2
Lectern	**Vanilla**	591 ± 190	725 ± 74	600 ± 84	66 ± 25
	BK	348 ± 89	499 ± 56	374 ± 61	66 ± 25
	GNN	72 ± 13	123 ± 25	60 ± 21	4 ± 9
	GNN + BK	57 ± 12	85 ± 14	23 ± 8	4 ± 9
White-board	**Vanilla**	217 ± 115	186 ± 36	144 ± 33	13 ± 7
	BK	184 ± 120	157 ± 25	115 ± 21	13 ± 7
	GNN	66 ± 32	35 ± 5	1 ± 1	5 ± 5
	GNN + BK	66 ± 32	35 ± 5	1 ± 1	5 ± 5
Cart	**Vanilla**	191 ± 53	252 ± 36	197 ± 40	22 ± 8
	BK	130 ± 33	174 ± 12	119 ± 16	22 ± 8
	GNN	51 ± 15	90 ± 11	49 ± 11	7 ± 2
	GNN + BK	54 ± 10	88 ± 10	47 ± 10	7 ± 2
Shelf	**Vanilla**	258 ± 49	579 ± 85	477 ± 80	49 ± 19
	BK	185 ± 50	447 ± 79	345 ± 71	49 ± 19
	GNN	35 ± 5	58 ± 3	6 ± 3	0 ± 0
	GNN + BK	34 ± 6	58 ± 3	6 ± 3	0 ± 0

attributes as the edge feature performance is always very close to the edge + node feature performance.

B. Improving the AbD Process

To investigate how our approach improves the AbD process, we conducted four experiments for each assembly:

1) *Vanilla:* The vanilla AbD framework [1] is used as described in Sec. IV-A.
2) *Bookkeeping (BK):* We extended the vanilla AbD framework with a bookkeping heuristic described in [2] to avoid unnecessary removal tests.
3) *Graph Neural Networks (GNN):* This uses our approach as described in Sec. IV-D. We trained a GNN on samples created for the other four assemblies as described in Sec. IV-B. We then used it to predict parts' removability probabilities and ordered the parts accordingly from high to low removability probability.
4) *Graph Neural Networks + Bookkeeping (GNN + BK):* A combination of 2) and 3) is used.

For all four experiments, we used the same pre-computed initial sequence in which parts were tested for removability. Each experiment was repeated with ten different initial part orderings. For each repetition, a GNN was trained on the dataset of the other four assemblies, each for 50 epochs. For comparability, the same trained GNNs were used for 3) and 4). The used assembly representation utilized edge as well as node attributes as described in Sec. III. The averaged results from all four experiments for each assembly, showing total execution time and the number of removal attempts, are displayed in Tbl. I. Wrt the execution time it is important to note that this property is only comparable for experiments on the same assembly, because removal attempts for differently shaped parts have different duration. For example, the AbD process for the cart is faster than for the whiteboard, despite the total number of removable parts and number of removal attempts being higher as shown in Fig. 4.

The results show, that the Vanilla approach always performed worst. Bookkeeping always reduced the number of needed removal attempts due to collision but could not reduce the number of removal attempts due to gravity constraints. This is expected, as the approach does only keep track of parts that are impossible to remove due to collisions. Our approach reduced the number of needed removal attempts further, not only for those that failed due to collision but also for those that failed due to gravity constraints. Especially high improvements were observed for the whiteboard and the shelf. During the preliminary study, we achieved over 95% accuracy for those, which led to an average of only six failed removal attempts for both. Overall our approach reduced the total number of removal attempts between 64% and 90% when compared to vanilla AbD and between 30% and 87% when compared to AbD with bookkeeping. This leads to speed ups between 2.6 and 7.4 compared to vanilla AbD and 1.4 and 5.3 compared to AbD with bookkeeping. The highest improvements were obtained for the shelf assembly and the smallest ones for the workbench.

Combining our approach with bookkeeping led to further improvements for three out of five assemblies. The highest further reduction of removal attempts compared to the GNN alone was achieved for the workbench assembly with 33%. This illustrates how our approach can be combined with complementary methods for further improvements.

VI. Discussion and Future Work

In our work, we focused on speeding up the AbD approach for ASP. It has to iteratively test parts for removal, hence its run-time is highly dependent on a good order for testing parts. We introduced a graph-based assembly representation and proposed to train a GNN on it to optimize the order. In detail, we used the GNN to predict the removability probabilities of parts and then ordered them accordingly, before testing them. We evaluated this approach using five different real-world assemblies, achieving a significant speedup for AbD (between 2.6 and 7.4). Further improvements were achieved by combining our approach with a bookkeeping heuristic. In future work, we want to add additional information to our representation such as an encoding of the spatial neighborhood around parts which would be useful for optimizing the reachability of parts. Additionally, our approach is limited as only the current disassembly state is considered when predicting removability, while potential future states could impact the ideal current choice. Thus, further improvements could be achieved through the use of reinforcement learning, which is able to consider the impact of future states in the disassembly. Other objectives, like the total number of tool changes, could also be optimized through such an approach.

REFERENCES

[1] T. Ebinger, S. Kaden, S. Thomas, R. Andre, N. M. Amato, and U. Thomas, "A general and flexible search framework for disassembly planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 3548–3555.

[2] S. Dorn, N. Wolpert, and E. Schömer, "An assembly sequence planning framework for complex data using general voronoi diagram," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 9896–9902.

[3] M. V. A. R. Bahubalendruni and B. B. Biswal, "An intelligent approach towards optimal assembly sequence generation," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 232, pp. 531 – 541, 2018.

[4] U. Thomas, M. Barrenscheen, and F. M. Wahl, "Efficient assembly sequence planning using stereographical projections of c-space obstacles," *Proceedings of the IEEE International Symposium on Assembly and Task Planning*, pp. 96–102, 2003.

[5] I. Aguinaga, D. Borro, and L. M. Matey, "Parallel rrt-based path planning for selective disassembly planning," *The International Journal of Advanced Manufacturing Technology*, vol. 36, pp. 1221–1233, 2008.

[6] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Iowa State University, Ames, Iowa 50011, USA, Tech. Rep., October 1998.

[7] S. Dorn, N. Wolpert, and E. Schömer, "Expansive voronoi tree: A motion planner for assembly sequence planning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 7880–7886.

[8] B. Deepak, G. B. Murali, M. V. A. R. Bahubalendruni, and B. B. Biswal, "Assembly sequence planning using soft computing methods: A review," *Proceedings of the Institution of Mechanical Engineers, Part E: Journal of Process Mechanical Engineering*, vol. 233, pp. 653 – 683, 2019.

[9] L. M. Galantucci, G. Percoco, and R. Spina, "Assembly and disassembly planning by using fuzzy logic & genetic algorithms," *International Journal of Advanced Robotic Systems*, vol. 1, 2004.

[10] H. Y. Zhang, H. Liu, and L. Li, "Research on a kind of assembly sequence planning based on immune algorithm and particle swarm optimization algorithm," *The International Journal of Advanced Manufacturing Technology*, vol. 71, pp. 795–808, 2014.

[11] K. Kitz and U. Thomas, "Neural dynamic assembly sequence planning," *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pp. 2063–2068, 2021.

[12] A. Swaminathan and K. S. Barber, "Ape: an experience-based assembly sequence planner for mechanical assemblies," *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1278–1283 vol.2, 1995.

[13] Q. Su, "Applying case-based reasoning in assembly sequence planning," *International Journal of Production Research*, vol. 45, pp. 29 – 47, 2007.

[14] K. Lee, S. Joo, and H. I. Christensen, "An assembly sequence generation of a product family for robot programming," *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1268–1274, 2016.

[15] Z. Zhou, R. Xiong, Z. Chen, and Y. Wang, "Assembly sequence generation for new objects via experience learned from similar object," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 1054–1061.

[16] I. Rodríguez, K. Nottensteiner, D. Leidner, M. Durner, F. Stulp, and A. Albu-Schäffer, "Pattern recognition for knowledge transfer in robotic assembly sequence planning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3666–3673, 2020.

[17] M. M. Kazhdan, T. A. Funkhouser, and S. M. Rusinkiewicz, "Rotation invariant spherical harmonic representation of 3d shape descriptors," in *Symposium on Geometry Processing*, 2003.

[18] Kjn workbench 4. [Online]. Available: https://grabcad.com/library/kjn-workbench-4-1

[19] Angled cart/lectern with drawer. [Online]. Available: https://grabcad.com/library/angled-cart-lectern-with-drawer-1

[20] Large rotatable white board. [Online]. Available: https://grabcad.com/library/large-rotatable-white-board-1

[21] Premium panel cart. [Online]. Available: https://grabcad.com/library/premium-panel-cart-1

[22] Aluminium profile shelf. [Online]. Available: https://grabcad.com/library/aluminium-profile-shelf-1

[23] M. Ankerst, G. Kastenmüller, H.-P. Kriegel, and T. Seidl, "3d shape histograms for similarity search and classification in spatial databases," in *International symposium on spatial databases*. Springer, 1999, pp. 207–226.

[24] P. Battaglia, J. B. C. Hamrick, V. Bapst, A. Sanchez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. J. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *arXiv*, 2018. [Online]. Available: https://arxiv.org/pdf/1806.01261.pdf

[25] G. Li, C. Xiong, A. K. Thabet, and B. Ghanem, "Deepergcn: All you need to train deeper gcns," *arXiv:2006.07739*, 2020.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[27] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[28] J. Pan, S. Chitta, and D. Manocha, "Fcl: A general purpose library for collision and proximity queries," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 3859–3866.

[29] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.