SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

GNNs for Knowledge Transfer in Robotic Assembly Sequence Planning

Matan Atad

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

GNNs for Knowledge Transfer in Robotic Assembly Sequence Planning

GNNs für Wissentransfer von Robotern bei der Planung von Montagereihenfolgen

Author:	Matan Atad
Supervisor:	Rudolph Triebel, PD Dr. habil.
Advisors:	Jianxiang Feng, M.Sc.
	Maximilian Durner, M.Sc.
	Ismael Rodriguez Brena, M.Sc.
Submission Date:	15.01.2023

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich,   Matan Atad

Abstract

Automated Assembly Sequence Planning (ASP) is a crucial task for robotic systems in manufacturing settings as it increases flexibility and efficiency. However, ASP is still largely performed manually, which can be time-consuming and prone to errors. The ASP process consists of two main steps: determining possible assembly sequences and confirming the feasibility of these sequences with the capabilities and restrictions of the target robot system. While several methods have been developed to automate ASP in recent years, they have limitations such as a lack of flexibility with regard to the properties of the assembly, a long training process, and a dependence on both positive and negative examples for feasibility detection.

To address these issues, we propose a graph-based approach that divides the ASP problem into two tasks: Sequence Prediction and Feasibility Prediction. For the Sequence Prediction task, we model assemblies as graphs of part surfaces and apply a Graph Neural Network (GNN) to efficiently extract meaningful information from the input. An expert demonstrator guides us through the sequence prediction process, step-by-step, predicting which parts can be placed in their position at each state of the assembly. Our method is flexible and able to transfer knowledge between different assembly tasks.

For Feasibility Prediction, we frame the task as an Anomaly Detection (AD) problem and use our GNN as a feature extractor to create latent representations for each assembly graph. We train a Normalizing Flows (NF) model to model the distribution of feasible assemblies and detect infeasible assemblies as Out-of-Distribution (OoD). Although our method performs better than a baseline one-class classifier, it still lags behind a binary classifier trained on both feasible and infeasible assemblies. However, it is not trivial to obtain a sufficient amount of infeasible assemblies for training. To better understand the limitations of our approach, we conduct ablation studies.

Our approach requires only a few seconds to derive a feasible assembly sequence and could be integrated into future ASP frameworks to dramatically reduce planning time while using fewer computational resources compared to previous methods. To our knowledge, we are the first to use NF for graph-level Anomaly Detection. Overall, our method has the potential to significantly improve the efficiency and effectiveness of ASP in manufacturing settings.

Contents

1. Introduction

1.1. Motivation

Recent disturbances to global supply chains following the COVID-19 pandemic and the war in the Ukraine are requiring manufacturers to become agile, for instance by relocating their production lines to other countries in order to mitigate shortages in resources (Shih, 2020). To cut down costs, producers are introducing automation at a greater pace, but rapid changes in market needs also demand often adaptions and customization of product design (Shih, 2020). These changes translate to often modifications in assembly lines, requiring their time-consuming and resource-intensive re-planning.

Automated assembly planning is one of the most important objectives of robotic systems to increase manufacturing flexibility since decreasing planning time and eliminating errors can facilitate greater product customization. Though product design is commonly executed with the assistance of Computer-Aided Design (CAD) systems, robotic assemblies are still mostly planned manually by experts (Rodrıguez et al., 2019).

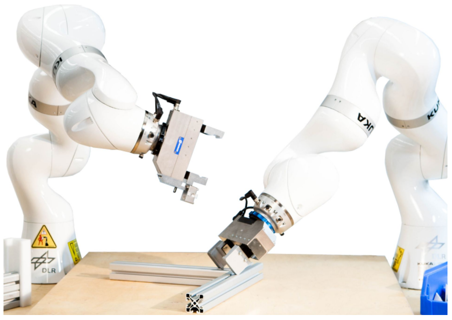
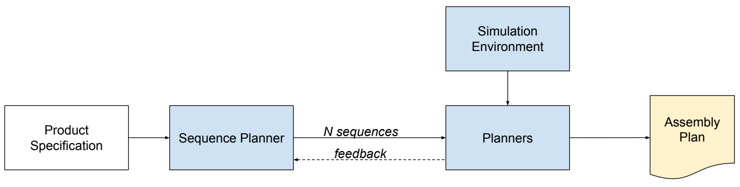
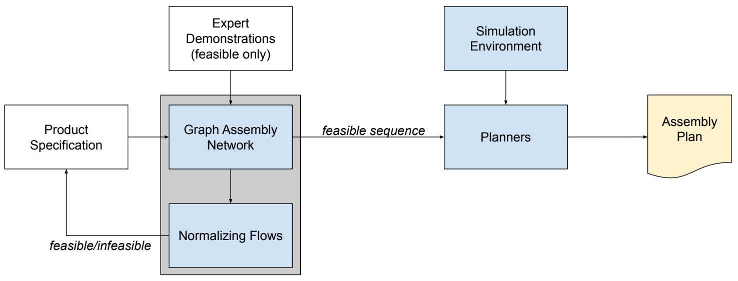


Figure 1.1.: A two-arm robotic system, similar to the one simulated by our environment, assembling a metal structure (Rodriguez et al., 2020).

Robotic Assembly Sequence Planning (ASP) includes two major steps which are interconnected. First, generate all possible assembly sequences, meaning, the order in which individual parts are put in place. This step is affected first and foremost by the geometrical properties of the assembly structure, i.e. how the different parts relate to



(a)



(b)

Figure 1.2.: Traditional (a) vs. our (b) method for Assembly Sequence Planning (ASP). In (a), multiple geometrically possible sequences are given to planners and analyzers, which verify their feasibility with the help of a simulation environment and can provide feedback. In (b), our Graph Assembly Network is trained on expert demonstrations and provide a known to be feasible sequence to the planners, reducing the overall planning time. Our Normalizing Flow (NF) model predicts the feasibility of the assembly based only on feasible examples.

each other. Second, confirm the *feasibility* of the candidate assembly sequences with the capabilities of the target robotic system. For instance, the Degrees of Freedom (DoF), the size of the working environment and the number of arms influence the capacities of the system. The system with two robotic arms in Fig. 1.1 can assemble a larger variety of products than one with a reduced skill-set since it can hand over parts from one gripper to the other (Rodriguez et al., 2020).

Fig. 1.2a depicts the planning process: the Sequence Planner provides geometrically possible sequences to downstream planners and analyzers, such as motion and grasp planners, inverse kinematics and workspace analyzers. These, in turn, reject infeasible sequences with the help of a simulation environment, potentially provide feedback to guide the sequence planners (Thomas et al., 2015) and finally derive an assembly plan.

Each of these two planning stages has its share of challenges. Finding candidate assembly sequences is an $\boldsymbol{NP}$-hard combinatorial problem since the amount of possible solutions grows with the factorial of the assembly parts (Rashid et al., 2012). Feasibility checks, confirming if an individual assembly sequence can be successfully executed on a specific robotic system, are also computationally expensive, as they require the plan to be executed in the robotic system or at least in its simulation (Rodrıguez et al., 2019). In the work by Suárez-Ruiz et al. (2018), 11 minutes were required for the assembly motion planning of an IKEA chair, more time than the actual execution of the plan. A fully automated assembly planner will likely require testing of many candidate sequences, accumulating potentially to hours.

Recently attempts have been made to automate the tedious planning process by using Neural Networks (NN) to directly predict feasible assembly sequences (Watanabe & Inada, 2020; M. Zhao et al., 2019) or to infer the underlying rules guiding their creation (Rodriguez et al., 2020). However, these methods suffer from several limitations. First, they use representations that do not support complex structures or are dependent on the number of parts in the assembly, therefore they could not generalize the knowledge gained. Second, they are dependent on multiple pre-processing steps (e.g. graph matching). Another line of works aims at predicting if a structure is feasible at all in a given robotic system (Bouhsain et al., 2022; Driess et al., 2020; Noseworthy et al., 2021; Wells et al., 2019; L. Xu et al., 2022), yet they require the inclusion of infeasible assemblies in the training set, demanding extra efforts in data collection.

1.2. Approach

We propose to divide the ASP problem into two sub-tasks. The first, *Feasibility Prediction*, classifies the feasibility of the assembly structure, and the second, *Sequence Prediction*, infers a known to be feasible assembly sequence. Aiming at addressing the limitations of existing methods, this thesis employs a graph-based approach for solving these two tasks.

Graph representation is intuitive for assemblies, as it is used to depict complex structures in many other domains, for instance, chemistry (Lim et al., 2019) and physics (Battaglia et al., 2016). We refined an existing representation of assemblies as graphs, suggested by Rodriguez et al. (2020), such that it will include a full specification of the structure parameters and its assembly state, i.e. which parts are in their target position. We set the assembly part surfaces as graph nodes and encode their corresponding geometrical distances as graph edges. This representation supports the depiction of assemblies with a varying number of differently shaped parts. In contrast to representations that use absolute part positions, ours is agnostic to rotations and mirroring of the assembly structure. The graph also contains part nodes, which encode the current state of the assembly, instead of using an external data structure for this purpose.

To extract useful information from the graph input, a Graph Neural Network (GNN) (Scarselli et al., 2008) is a popular and appealing method, as it allows for high computation-space efficiency (Wu et al., 2020). We train our Graph Assembly Network to follow an expert demonstrator (Lin et al., 2022) and solve the Sequence Prediction task in a *step-by-step* setting, predicting in each state of the assembly which parts could be placed in their target position. We can employ this setting since assembly sequences exhibit the Markovian property (Bellman, 1957), in which each step is independent of all previous ones, given the current state. Apart from its simplicity and training efficiency, this approach allows our model to be independent of the number and length of predicted sequences. Indeed, we demonstrate that it can transfer knowledge between different assembly tasks.

We frame Feasibility Prediction as an Anomaly Detection (AD) problem, allowing us to detect infeasible assemblies in a single-class setting. We employ our Graph Assembly Network as a feature extractor and use its output graph latent as an input to a Normalizing Flows (NF) (Dinh et al., 2014) model, which models the distribution of feasible assemblies. Since we assume infeasible assemblies are drawn from a different distribution, we can use an Out-of-Distribution (OoD) detection setting to identify them (Yang, Zhou, et al., 2021), since the NF assign them with a lower likelihood. We conducted ablation studies to investigate the intrinsic driving factors that contribute to the performance of our model.

1.3. Research Scope

The scope of this work is developing a method to solve the two presented ASP tasks using GNN and NF models.

Fig. 1.2b presents how our approach could be integrated into an Assembly Planning framework. Our Graph Assembly Network and NF models are first trained on expert demonstrations of feasible sequences represented as Assembly Graphs. Given a query assembly, the feasibility of its graph latent is predicted by our NF model. If it's feasible, an assembly sequence is derived from our Graph Assembly Network by traversing the assembly state tree and provided to downstream planners and analyzers in order to derive an assembly plan.

We define the following three research questions:

1. Can we represent assemblies in a way that is flexible and supports differently-sized structures?
2. Can we generalize knowledge and correctly predict assembly sequences?
3. Can we predict the feasibility of an assembly only based on positive examples, i.e. other feasible assemblies?

1.4. Thesis Structure

This work includes six other chapters. Chapter 2 covers related work in robotic Assembly Sequence Planning, the way graphs are used in the field of Task Planning and previous methods for graph-level Anomaly Detection. Chapter 3 presents the theoretical background required for the development of our method. Chapter 4 describes our approach, the architecture of the model and its implementation. In Chapter 5, we present the experimental setting, the experiments we conducted and their results. We summarize our approach and key contributions in Chapter 6. Finally, in Chapter 7, we discuss the limitations of our work and present directions for future research.

2. Related Work

The focus of this work is Assembly Sequence Planning (ASP), a variant of the Task and Motion Planning (TAMP) problem. In Section 2.1, we review previous approaches to ASP. In section Section 2.2, we discuss recent advancements in TAMP, mainly the use of GNNs, which inspired our approach. Finally, in Section 2.3, we present current approaches for graph Anomaly Detection (AD) for feasibility prediction.

2.1. Assembly Sequence Planning (ASP) in Robotics

Assembly Sequence Planning (ASP) is a process that receives a geometric description of a final assembly product and derives a sequence of operations, according to which the target product can be assembled step-by-step. A principle problem for robotic assembly sequence planning is that the number of parts orderings is factorial in the number of parts, making it an $\boldsymbol{NP}$-hard problem (Rashid et al., 2012).

We define a *feasible* sequence as an ordering of all assembly parts which is possible in a specific robotic system and will put the structure together without collision while considering geometrical restrictions and ones originating from the motion planners (Jiménez, 2013). A second question concerns the *optimality* of the assembly sequence, which could be defined in terms of the overall assembly time, the number of required tool changes and more (Jiménez, 2013). In this work, we focus on feasibility, though we believe our approach could be extended to incorporate optimality as well[1]. Nonetheless, we mention here works considering the question of optimality for completeness.

Traditional approaches for ASP incorporate the following steps (Jiménez, 2013):

1. Definition of the precedence constraints between parts.

2. Generation of all feasible assembly sequences.

3. Searching for the optimal solution.

Commonly, a graph depicting the assembly process is created in which nodes represent individual assembly steps. Finding a solution in this setting means searching a path

[1]In order to classify assemblies as feasible, our method learns to follow expert demonstrations. These could potentially be restricted only to optimal ones (see more in Chapter 7).

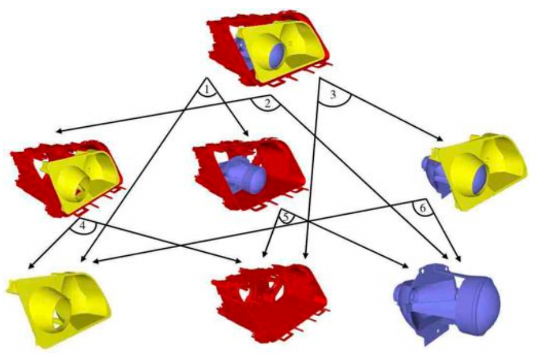


Figure 2.1.: An example of an AND/OR Graph (Thomas et al., 2015). The root of the graph is the complete assembly and the leaves are the individual parts.

between nodes representing initial assembly states to the one representing the final product.

2.1.1. Graph Search Algorithms

A popular assembly graph representation is the AND/OR Graph (De Mello & Sanderson, 1990) (Fig. 2.1), in which the root is the full assembly structure and the leaves are the individual parts. Each node represents a state of the assembly and an edge connecting two nodes means a single disassembly operation[2] leading from one state to its predecessor.

The AND/OR Graph could be created using the *Disassembly For Assembly* strategy (De Fazio & Whitney, 1987), which converts the task of finding how to assemble a given product to a simpler equivalent problem- finding how it could be disassembled. Starting from a single complete target assembly, it recursively removes parts till an initial state is reached, in which all of the assembly parts are disconnected. In an alternative strategy, in which the graph is built bottom-up from the set of individual parts, there are numerous target configurations that could be reached, meaning an infinite branching factor[3] (De Mello & Sanderson, 1990), making the former strategy simpler.

As an example of an application of this method, Thomas et al. (2003) inferred the AND/OR Graphs out of product CAD files. Later they devised a method to prune the

[2]Note that assembly operations are not necessarily reversible (e.g. drilling a part). The expression *Disassembly Operation* refers to a reverse of an assembly operation.

[3]The *Branching Factor* of a graph node is the number of its children or out-degree. High branching factors make search algorithms that follow every branch computationally expensive, potentially leading to combinatorial explosion (Edelkamp & Korf, 1998).

geometrically infeasible paths out of these graphs using computer vision techniques (Nottensteiner et al., 2016; Thomas et al., 2015). Nevertheless, their method still requires validation with a grasp planner to enforce the restrictions of the robotic system, considerably increasing the time required for finding a feasible sequence. This is a major disadvantage in comparison to our approach, which requires detailed planning only for a single, known to be feasible, sequence.

Work	Graph Representation	Method	Limitations
Thomas et al. (2015) Nottensteiner et al. (2016)	Complete assembly process	Graph search	Prolonged validation with planers
Iwankowicz (2016) Ab Rashid (2017) B. Li et al. (2022)	N/A	Heuristics	No actual learning
Sinanoğlu and Börklü (2005) Chen et al. (2008)	N/A	NN	Inflexible internal representations
M. Zhao et al. (2019) Watanabe and Inada (2020)	N/A Complete assembly process	Deep RL	Inflexible internal representations Limited application to other problems
Rodriguez et al. (2020)	Complete assembly process Assembly structure	Rules Inference, NN, Graph matching	Inflexible internal representations Limited application to large structures

Table 2.1.: Discussed approaches for Assembly Sequence Planning (ASP)

2.1.2. Heuristics

Although exhaustive search is the simplest strategy ensuring the detection of the optimal sequence, it is impractical for larger assemblies. Therefore, multiple meta-heuristic methods have been proposed to reject infeasible sequences and find ones close to the optimal (Jiménez, 2013). These include Ant Colony Optimization (ACO) (Ab Rashid, 2017), Genetic Algorithms (GA) (B. Li et al., 2022) and Evolutionary Algorithms (EA) (Iwankowicz, 2016). Though these approaches reduce the search time, they still have to be executed separately per assembly, since no actual learning from experience is performed.

Few works (Chen et al., 2008; Sinanoğlu & Börklü, 2005) suggested NN as a heuristic, using Supervised Learning to infer a mapping from the geometric features to a complete assembly sequence and as such, they are early predecessors of our work. Nevertheless, they input feature vectors and thus do not scale to assemblies with different sizes and different objects. Also, they are evaluated only on a few samples and are dependent on multiple pre-processing steps.

More recently, M. Zhao et al. (2019) and Watanabe and Inada (2020) applied deep Reinforcement Learning (RL) for ASP, to learn a policy for assembly sequencing.

Though both methods initialize the policy with results from past solved tasks, they still require retraining for every new structure. This is the case since their internal representations are based on feature vectors which are dependent on the number of parts in the assembly and its specific characteristics. Though they can gain insights from known past sequences, their learning is limited and they can not be easily applied to other problems.

2.1.3. Rule Inference

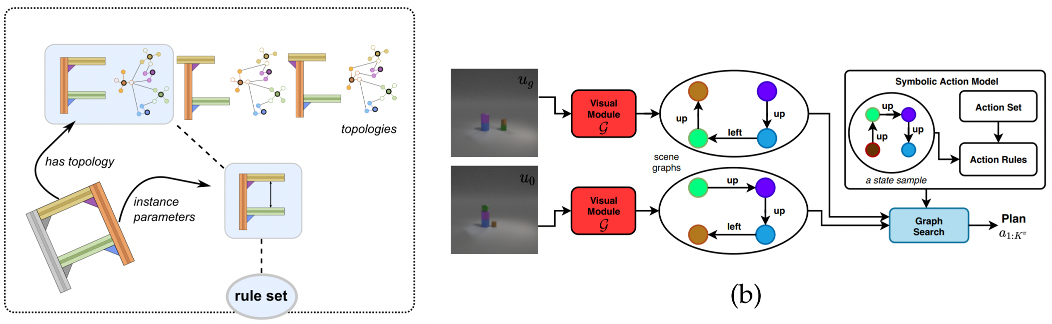
In a recent work, motivating ours, Rodriguez et al. (2020) suggested inferring assembly rules (e.g. a specific part type should be assembled before another), thus allowing knowledge transfer of workspace limitations between different assemblies (Fig. 2.2a). They define a concept of an assembly *Topology*, a graph representation of the assembly layout, depicting its parts and corresponding surfaces as nodes. Edges in this graph mean either association of a surface to a part or relations between surfaces (e.g. touching, screwed). The topology graph does not include any specific geometrical characteristics of the assembly (e.g. distances between surfaces), and therefore many different assemblies, or *instances*, could be associated with the same topology. Using graph sub-matching, they associate each specific assembly with its matching topologies from a predefined database of known ones. Finally, a topology-specific classifier is used to predict the assembly rules given the instance-specific geometrical parameters.

The presented approach has several limitations. First, it is highly coupled to the number of parts in the assembly, as training a different rules classifier is required per topology. In addition, for larger assemblies, the sub-graph matching step would likely result in pairing with multiple smaller topology classes. As mentioned by Rodriguez et al. (2020), it may be hard to consolidate these and infer rules covering the constraints of the larger assembly.

2.2. Graphs for the Task Planning Problem

ASP can be seen as a variant of the Task Planning problem (in itself a sub-task of TAMP), which plans robot operations in an environment with complex, often long-horizon, objectives, involving different objects and manipulators (Garrett et al., 2021). Although this problem setting is more general than ours, we draw inspiration from works in this field.

Task Planning algorithms define symbolic rules for the states, actions and constraints used by the planners. Their output action sequences are usually provided to a motion planner which checks the kinematic feasibility in the geometric world (Thomas et al.,



(b)

(a)

Figure 2.2.: Overview of methods by Rodriguez et al. (2020) (a) and S. Nguyen et al. (2020) (b). In (a), graph matching is used to infer a higher level topology for an assembly. A classifier then predicts assembly rules given the specific instance parameters. In (b), graph search is performed to find steps leading from a source to target graphs.

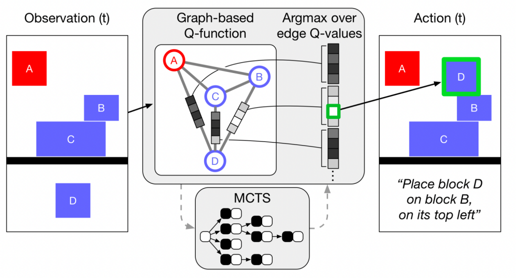
2003). Unfortunately, motion planning is computationally demanding and its usability as a feasibility checker is restricted in the real world.

A vast body of task-planning approaches were suggested over the years (Garrett et al., 2021). We will focus here on approaches that represent the environment where the robot operates as a graph since this representation is scalable and can take advantage of relational priors between environment objects (Y. Zhu et al., 2021). In this setting, the graphs commonly incorporate nodes for manipulated objects (Bapst et al., 2019; S. Nguyen et al., 2020; Y. Zhu et al., 2021), their target positions (Funk et al., 2022; Lin et al., 2022) and the robot gripper (Ye et al., 2020). Edges can represent high-level relations between objects (S. Nguyen et al., 2020; Y. Zhu et al., 2021).

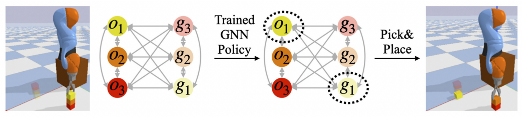
2.2.1. Sampling and Optimization

Classical methods for task planning incorporate *sampling* or *optimization* (K. Zhang et al., 2022). Given a task with a description of an initial and a goal state, sampling methods sample possible intermediate states leading to the goal state from a continuous infinite state space. Later, searching algorithms or heuristics are used to find a sequence of feasible actions between these intermediate states.

A recent trend in these approaches is to directly work on visual input instead of specifications of the environment. This has its benefit in a TAMP setting, in which robots are expected to interact with a dynamic and ever-changing environment. In our setting, though, a static specification of the end product could be provided from CAD


Observation (t)
Graph-based Q-function
Argmax over edge Q-values
Action (t)
A
B
C
D
MCTS
"Place block D on block B, on its top left"

(a)



(b)

Figure 2.3.: Overview of methods by Bapst et al. (2019) (a) and Lin et al. (2022) (b). In (a), a graph is fed to a GNN, producing updated edge embeddings. The output action is obtained with an argmax across all edges coordinates, each defines the probability of performing an action between the source and target nodes. In (b), GNNs select the next object and its corresponding goal position, by applying softmax on the respective node embeddings.

files. Moreover, working with structured inputs presents substantial advantages in the number of computational resources required[4].

Y. Zhu et al. (2021) created a *geometric* graph representing the positions and poses of objects in the environment before and after a robot manipulation from two input images in a fully-supervised setting. Node features for this graph originate in a pose estimation pipeline. Next, they use heuristics to convert this graph into a *symbolic* scene graph, in which edges represent high-level relations between objects (*On*, *In* etc.). Starting from the graph representing the goal state, they let a GNN act as a task planner, predicting a series of states leading back to the initial state (Backward Search). Then, they sample some of these candidates and provide them to a second GNN, which predicts which of these would be feasible in the current state of the environment.

Working in a similar setting, Ye et al. (2020) use sampling on their graph representations, starting from an initial state and enumerating possible next steps (Forward Search). As part of their training, they fit a multi-dimensional Gaussian on sequences of actions in the training set, allowing them to sample feasible action sequences during inference. However, this limits their applicability to same-length sequences.

In optimization-based methods, solutions are found by minimizing a constrained cost function and enforcing the geometrical constraints of the robotic system. Similarly to the previous method, S. Nguyen et al. (2020) infer from two input images a graph representing objects in an environment before and after a robot manipulation. They

[4]Our GNN pipeline has $\approx 50K$ parameters vs. many millions required by ones processing images.

then perform sampling and a search to find action sequences that will transform the source graph to the target (Fig. 2.2b). Finally, they let a non-linear program eliminate sequences that do not adhere to environment constraints.

The major difference between the problem setting of these works to ours is the fact they operate in an infinite state space, making them dependent on external components, such as optimizers, to retrieve feasible action sequences. This in turn restricts them to simplified environments, made of identical manipulated objects. In our setting, the state space is finite, though factorial, and known, allowing us to model it directly and also incorporate complex objects with intricate relations.

2.2.2. Learning

Reinforcement Learning methods learn policies that map the current state of the environment to a projected next state by maximizing a numerical reward signal (K. Zhang et al., 2022). Bapst et al. (2019) (Fig. 2.3a) and follow-up work by Funk et al. (2022) and R. Li et al. (2020), use GNNs to learn policies for the construction of target structures using given building blocks. Bapst et al. (2019) graph representation includes nodes for each object, while Funk et al. (2022) differentiate between placed and unplaced objects nodes and also add nodes that define the target structure layout. The graphs are forwarded to GNNs and the output policy action is obtained through edges (Bapst et al., 2019) or node-pair embeddings (Funk et al., 2022).

The main disadvantage of RL methods is their long training time and the vast amount of data required. For instance, the model by Funk et al. (2022) required 5000 epochs to converge and R. Li et al. (2020) needed hundreds of millions of training samples. Our model is trained on a few thousand examples for as little as 35 epochs.

As a direct inspiration to our work, instead of using RL, Lin et al. (2022) (Fig. 2.3b) use learning from demonstrations (or *Imitation Learning*) to train two GNNs, one that selects objects in the scene and another that selects a suitable goal state from a set of possible goal positions. Their graph representation holds nodes for objects as well as goal positions and the network predictions are obtained by applying softmax over the nodes embeddings. Imitation learning reduces the amount of training data required, however, their graph representations lack geometrical information about manipulated objects. They assume an environment made of identical blocks with simplified relationships, therefore their ability to take motion planning feasibility into account is limited. We aim to incorporate knowledge of the restrictions of the robotic environment into our model.

Work	Graph Representation	Method	Limitations
Y. Zhu et al. (2021) Ye et al. (2020) S. Nguyen et al. (2020)	Separate graphs for source & target environment states	Sampling, heuristics Sampling, search Sampling, search, optimization	Simplified, unscalable
Bapst et al. (2019) R. Li et al. (2020) Funk et al. (2022)	Single graph with nodes for objects, gripper, targets etc.	RL	Prolonged training
Lin et al. (2022)	Single graph with nodes for objects & their goal positions	Imitation Learning	Simplified

Table 2.2.: Discussed approaches for Task and Motion Planning (TAMP)

2.3. Graph Anomaly Detection for Feasibility Prediction

We represent assemblies as graphs and frame the feasibility prediction problem as graph-level Anomaly Detection. This setting allows us to detect infeasible assemblies based only on feasible ones and thus eliminate the effort required in collecting a dataset of infeasible assemblies.

2.3.1. Plan Feasibility

We start by presenting several previous methods, centered around the question of plan feasibility. Rodrıguez et al. (2019) show how the order of expensive feasibility checks executed with a planner could be optimized, for instance by pruning sequences that are expected to fail and performing time-consuming evaluations only if quicker ones have passed successfully. Wells et al. (2019) trained a feature-based Support Vector Machines (SVM) (Cortes & Vapnik, 1995) model to directly predict the feasibility of an action sequence based on experience, but it can not be scaled to scenes with a different number and types of objects. Driess et al. (2020) (Fig. 2.4) and two recent follow-up works (Bouhsain et al., 2022; L. Xu et al., 2022) use a NN to predict if a mixed-integer program can find a feasible motion for a required action based on visual input. Noseworthy et al. (2021) take an active learning approach to this problem, guiding the dataset acquisition step. Their method choose action sequences that are deemed most informative (reduce the entropy the most) and learns from their successes or failures.

Interestingly, all these methods work in a fully-supervised setting, requiring failing action sequences to be included in the training set and then use binary classifiers. We devise a single-class method by modeling the distribution of feasible assemblies with Normalizing Flows, thus feasibility prediction translates to Out-of-Distribution detection (see Section 4.1.2).

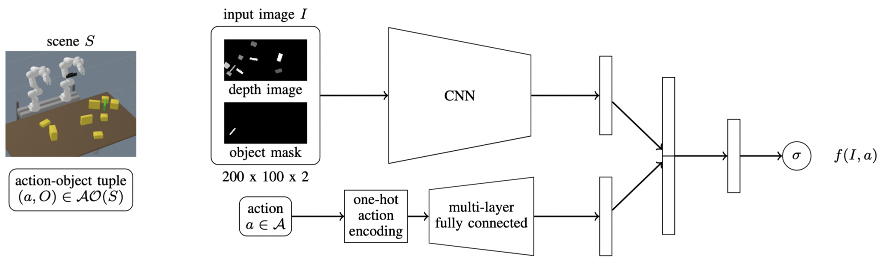


Figure 2.4.: Overview of method by Driess et al. (2020). A NN predicts the feasibility of an action on a specific object in the scene input image. The object in question is specified using a mask.

2.3.2. Graph-Level Anomaly Detection

Graph AD is a widely researched topic, though most works focus on identifying anomalous nodes and edges in a single graph, while only a few methods tackle the graph-level setting (X. Ma et al., 2021), i.e. identifying graphs abnormal to the ones seen during training. Moreover, when it comes to the graph-level problem, many studies aim to detect abnormal changes in sequences of time-dependent graphs (Cui et al., 2019; Eswaran et al., 2018; Lagraa et al., 2021), yet these are difficult to generalize to settings with large variations in structure between non-related graph instances.

Considering AD in static graphs, some approaches (H. T. Nguyen et al., 2020; L. Zhao & Akoglu, 2021) suggested first acquiring graph embeddings, for instance with Graph2Vec (Narayanan et al., 2017) or Weisfeiler-Leman graph kernel (Shervashidze et al., 2011), and then apply existing anomaly detectors, such as Isolation Forest (Liu et al., 2008), Local Outlier Factor (Breunig et al., 2000) or One-class SVM (OC-SVM) (Schölkopf et al., 1999), to derive an anomaly score. The main problem with these methods is that they are dependent on the quality of the graph embeddings, which were inferred in the previous step, and may not be optimal for the AD task.

Few methods utilize GNNs for frameworks optimized end-to-end. L. Zhao and Akoglu (2021) suggested a model consisting of a Graph Isomorphism Network (GIN) (K. Xu et al., 2018), a node pooling layer and a Deep Support Vector Data Description (SVDD) (Ruff et al., 2018) one-class classifier. A different approach is followed by R. Ma et al. (2022), who use Knowledge Distillation for capturing the training data patterns.

Work	Graph Embeddings	Classifier	Limitations
H. T. Nguyen et al. (2020)	Graph2Vec Graph kernels	Isolation Forest Local Outlier Factor OC-SVM	Graph embeddings not optimal
L. Zhao and Akoglu (2021) R. Ma et al. (2022)	GNN	Deep SVDD Difference between Predictor and Random networks	No exact density estimation
Gomes-Selman and Demir (2019)	GNN	Predicted likelihood of Auto-regressive model	Limited to simplified graphs

Table 2.3.: Discussed approaches for Graph Anomaly Detection

They train a Predictor GNN to follow the output embeddings of another Target GNN, fixed with random weights. Finally, their anomaly score is defined as the difference between the two networks' predictions.

Generative models and especially Generative Adversarial Network (GAN) (Goodfellow et al., 2014) are commonly used for AD in other domains (Kwon et al., 2019; Rani et al., 2020; Yang, Xu, et al., 2021). These methods are based on the idea that anomalies cannot be generated since they do not exist in the training data. In this setting, a NN learns the patterns of the training data by reproducing it from its corresponding latent representation. The anomaly score is derived from a Reconstruction Error, which compares the original and reproduced data instances.

Unfortunately, generative models for graphs are quite limited in their reconstruction abilities from a latent representation, especially in complex settings such as ours (i.e. heterogeneous graphs with a varying number of nodes) (Wu et al., 2020). Gomes-Selman and Demir (2019) use Graph Auto-regressive models (You et al., 2018) to compare the predicted scores of training and anomalous graphs. However, their experiments are limited to highly simplified synthesized graphs.

All the methods presented so far use their anomaly score as a rough approximation of the true density of the abnormal samples. As a direct inspiration to our method, several works have explored the usage of Normalizing Flows (NF) (Dinh et al., 2014) for AD in other domains (Nachman & Shih, 2020; Rudolph et al., 2021; Wellhausen et al., 2020). Similar to the two stages approach above, a latent representation of the sample is first obtained from a feature extractor. Then, this latent is transformed through a flow to obtain its density. A major advantage of this approach is its capability to compute the density of a sample directly, without approximation. However, as mentioned before, it is dependent on the ability of the feature extractor to extract semantics relevant to the task (Kirichenko et al., 2020). To the best of our knowledge, we are the first to apply NF to the graph-level AD problem.

3. Background

This chapter presents the theoretical background required for the development of our method. We elaborate on the principles of Graph Neural Networks (GNNs) (Section 3.1), Heterogeneous Graph Neural Networks (Section 3.2) and Normalizing Flows (NF) (Section 3.3), as these are fundamental to our approach. In addition, we briefly describe the Task Planning formulation (Section 3.4), as we use it in our problem definition.

3.1. Graph Neural Networks (GNNs)

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph with nodes $\mathcal{V}$ and undirected edges $\mathcal{E}$, where every node $v \in \mathcal{V}$ is assigned with a feature vector $\phi(v)$. Graph Neural Networks (GNNs) (Scarselli et al., 2008) are neural networks operating on graphs, capable of updating node features by exchanging information between neighboring nodes. This is done using a *Message Passing* layer (Gilmer et al., 2017), where commonly multiple message-passing layers are used. For every message passing layer l, a three-step process to update the features of node $v \in \mathcal{V}$ is defined:

1. *Gather* the feature vectors, or messages, of neighbouring nodes: $\{h_j^{l-1}\}_{j \in \mathcal{N}_i}$.
2. *Aggregate* the neighbouring nodes messages: $m_i^l = g_\omega(\{h_j^{l-1}\}_{j \in \mathcal{N}_i})$.
3. *Update* the features of the node: $h_i^l = f_\theta^l(h_i^{l-1}, m_i^l)$.

Where $\mathcal{N}_i$ are the neighbours of node v_i, the function g_ω can be learned during training or constant (e.g. sum) and f_θ is a NN with learned weights θ. f_θ and g_ω are shared across all nodes in the graph, making GNNs efficient and independent of the number of nodes in the graph. We set $h_i^0 = \phi(v_i)$, i.e. the input features.

Numerous GNN architectures were proposed in recent years (Wu et al., 2020), suggesting different ways to aggregate neighbor features, combining message-passing layers with activation functions and more. We will present here the two most prevalent architectures which incorporate a notion of edge weight or features into the formulation[1] since these are required in our setting (see Section 4.2).

[1]Both of these assume the graphs include self-edges.

Graph Convolution Networks (GCN)

In Graph Convolution Network (GCN) (Kipf & Welling, 2016), the neighbor's contribution to the node features is convoluted using the weight of the edge connecting the two nodes:

$$m_i^l = \sum_{j \in \mathcal{N}_i} \left(\frac{e_{i,j}}{\sqrt{d_i d_j}} \cdot h_j^{l-1} \right) \tag{3.1}$$

$$h_i^l = \mathbf{W} \cdot \frac{e_{i,i}}{d_i} h_i^{l-1} + \mathbf{W} \cdot m_i^l \tag{3.2}$$

Where $\mathbf{W}$ is a learned weight matrix, $e_{i,j}$ is the weight of the edge connecting the edge v_i to v_j and d_i is an edge weight normalization term: $d_i = 1 + \sum_{j \in \mathcal{N}_i} e_{i,j}$.

Graph Attention Networks (GAT)

In Graph Attention Network (GAT) (Brody et al., 2021; Veličković et al., 2017), the neighbor's contribution to the node features is a relative weight, learned using an attention mechanism:

$$m_i^l = \sum_{j \in \mathcal{N}_i} \left(\alpha_{i,j} \cdot h_j^{l-1} \right) \tag{3.3}$$

$$h_i^l = \mathbf{W}_1 \cdot \alpha_{i,i} h_i^{l-1} + \mathbf{W}_1 \cdot m_i^l \tag{3.4}$$

$$\alpha_{i,j} = \frac{\exp\left(\mathbf{a} \cdot \sigma\left(\mathbf{W}_2 [h_i^{l-1} \parallel h_j^{l-1} \parallel e_{i,j}]\right)\right)}{\sum_{j \in \mathcal{N}_i \cup \{i\}} \exp\left(\mathbf{a} \cdot \sigma\left(\mathbf{W}_2 [h_i^{l-1} \parallel h_k^{l-1} \parallel e_{i,k}]\right)\right)} \tag{3.5}$$

Where $\mathbf{W}_1$, $\mathbf{W}_2$ and $\mathbf{a}$ are learned, σ is a Leaky ReLU activation function (Maas et al., 2013) and $\parallel$ is a vector concatenation operator.

3.2. Heterogeneous Graph Neural Networks

A Heterogeneous Graph (Sun & Han, 2013) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ generalizes graphs to multiple types of nodes and edges. Each node $v \in \mathcal{V}$ belongs to one particular node type $\psi(v)$ and analogously each edge $e \in \mathcal{E}$ to an edge type $\phi(e)$. We further define for each node v a set of distinct node and edge types connected to it $\Phi(v) = \{(\psi(v), \phi(e))\}$.

Both GCN and GAT models were extended to a heterogeneous graph setting (Wang et al., 2019; C. Zhang et al., 2019). This is accomplished by obtaining for each node a different updated representation per group of specific neighboring source node and edge types, and aggregating the different representations to obtain a single result, for instance using a sum. This formulation is essential, as every type of neighbouring node may have a different dimension.

For example, Eq. 3.2 is replaced for Heterogeneous GCN with:

$$h_i^l = \mathbf{g}\left((h_i^l)^k\right)_{k\in\Phi(v_i)} \tag{3.6}$$

$$(h_i^l)^k = \mathbf{W}^k \sum_{j\in\mathcal{N}_i^k}\left(\frac{e_{i,j}}{\sqrt{d_i d_j}}\cdot h_j^{l-1}\right) \tag{3.7}$$

Where $\mathbf{g}$ is an aggregation function (sum, average etc.), $\mathcal{N}_i^k$ is a sub-set of v_i neighbours with a specific node type and connected to v_i with a specific edge type. Notice how the general weight matrix $\mathbf{W}$ in Eq. 3.2 is replaced here with $\mathbf{W}^k$, to match the specific feature dimensions of the nodes $\mathcal{N}_i^k$.

3.3. Normalizing Flows

Normalizing Flows (NF) (Dinh et al., 2014) are transformations of simple base probability distributions into complex ones using a sequence of invertible and differentiable transformations. Given data samples from an unknown distribution we can map them into a simple distribution (e.g. Gaussian) in which it is possible to evaluate their densities.

Change of Variables Formula

Let $X, Z \in \mathbb{R}^D$ two random variables such that Z has a known and tractable probability density function $p_Z : \mathbb{R}^D \to R$, called the base distribution. Let $f : \mathbb{R}^D \to \mathbb{R}^D$ an invertible and differentiable (*valid*) transformation such that $f(Z) = X$. Using the change of variables formula, one can compute the probability density function of the random variable X:

$$p_X(X) = p_Z(f^{-1}(X))\left|\det\left(\frac{\partial f^{-1}(X)}{\partial X}\right)\right| \tag{3.8}$$

$$= p_Z(Z)\left|\det\left(\frac{\partial f(Z)}{\partial Z}\right)\right|^{-1} \tag{3.9}$$

Where $\det(A)$ is the determinant of the square matrix A and $J(f) = \frac{\partial f(Z)}{\partial Z} \in \mathbb{R}^{D\times D}$ is the Jacobian matrix of partial derivatives such that $J(f)_{i,j} = \frac{\partial f(Z)_i}{\partial Z_j}$. The requirement for $f(\cdot)$ to be invertible and differentiable ensures that the Jacobian matrix has an inverse and its determinant is tractable.

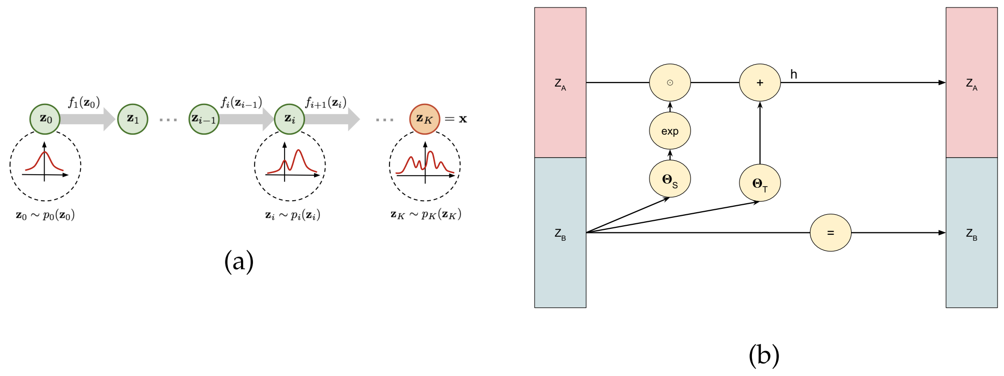


Figure 3.1.: Transformation Stacking (a) and Affine Coupling Flows (b). In (a), stacking of multiple flows transforms a simple distribution $p_0(Z_0)$ to a complex one $p_K(Z_K)$ (Weng, 2018). In (b), the *coupling function* $h(\cdot|\theta)$ applies "scale-and-shift" to the upper part of the input Z^A, while its parameters depend on the output of the NNs Θ_S and Θ_T on the lower part Z^B.

Transformation Stacking

Given an arbitrary complex transformation f, one can generate any distribution p_X from any base distribution p_Z with Eq. 3.8 (Bogachev et al., 2005). However, constructing these mapping such that they are convenient to compute, invert, and calculate the determinant of their Jacobian is a difficult task. Instead, we can stack multiple simple transformations to create arbitrary complex ones (Fig. 3.1a).

Let $f_1, \ldots, f_K$ a set of K valid transformations, we denote $Z_i = f_i \circ \cdots \circ f_1(Z)$, $i \in [1, K]$, and $Z_K = X$. We also define $Z_i \sim p_i(Z_i)$. The stacking, or composition, $f = f_K \circ \cdots \circ f_1$ can be shown to be valid and has the following determinant of the Jacobian (Kobyzev et al., 2020):

$$\det J(f) = \prod_{i=1}^{K} \left| \det \left(\frac{\partial f_i^{-1}(Z_i)}{\partial Z_i} \right) \right| \tag{3.10}$$

Substituting this term into Eq. 3.8 we get:

$$p(X) = p_0(Z_0) \prod_{i=1}^{K} \left| \det \left(\frac{\partial f_i^{-1}(Z_i)}{\partial Z_i} \right) \right| \tag{3.11}$$

Coupling Flows

Given an input $Z \in \mathbb{R}^D$, we define its disjoint partition $(Z^A, Z^B) \in \mathbb{R}^d \times \mathbb{R}^{D-d},\ d < D$. Let the *coupling function* $h(\cdot|\theta) : R^d \to R^d$ be a valid transformation parameterized by θ. We define the *coupling flow* (Dinh et al., 2014):

$$\begin{aligned} X^A &= h(Z^A|\Theta(Z^B)) \\ X^B &= Z^B \end{aligned} \tag{3.12}$$

Where h is parameterized by a *conditioner function* $\Theta(Z^B)$, an output of some function on Z^B alone. Assuming h is invertible, one can show that the coupling flow is a valid transformation (Dinh et al., 2014).

For increased complexity, it is common practice to use transformation stacking (Eq. 3.11) and compose several coupling flow blocks. However, in every coupling flow layer, $D - d$ input dimensions remain unchanged. Therefore, the ordering of input dimensions is reversed after each block; Z^A is alternately replaced with Z^B.

Affine Coupling Flows

Several methods were proposed in regards to $\Theta()$ and $h()$. In the RealNVP architecture (Dinh et al., 2016), it is suggested to use *Affine Coupling* (or "scale-and-shift") (Fig. 3.1b):

$$h(Z^A) = Z^A \odot \exp\Big(\Theta_S(Z^B)\Big) + \Theta_T(Z^B) \tag{3.13}$$

Where $\Theta_S, \Theta_T : \mathbb{R}^{D-d} \to \mathbb{R}^d$ are scale and translation NNs and $\odot$ is an element-wise product.

Density Estimation with Normalizing Flows

Let f_θ be a transformation parameterized with θ and let p_Z a given base distribution parameterized with ϕ. Given a set of samples $D = \{X^i\}_{i=1}^M$ from an unknown distribution, we can perform likelihood-based estimation of the parameters $\Theta = (\theta, \phi)$:

$$\log p(D|\Theta) = \sum_{i=1}^{M} \log p_X(X^i|\Theta) \tag{3.14}$$

$$= \sum_{i=1}^{M} \log p_Z(f_\theta^{-1}(X^i)|\phi) + \log \left|\det\Big(J(f_\theta^{-1}(X^i)\Big)\right| \tag{3.15}$$

During training, the parameters of the transformation θ and of the base distribution ϕ are updated to maximize the log-likelihood.

3.4. Task Planning Formulation

Task Planning specifies a tuple $\{\mathcal{S}, \mathcal{A}, \sigma_0, \mathcal{S}_*\}$ to describe the environment in which a robot operates. The tuple is comprised of a set of environment states $\mathcal{S}$ and a set of all possible actions, or transitions, $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$, describing changes leading from one state of the environment to the next. Each action $\alpha =< \sigma, \hat{\sigma} >\in \mathcal{A}$ moves the environment from state σ to state $\hat{\sigma}$. In addition, an initial state $\sigma_0 \in \mathcal{S}$ and a set of possible goal states $\mathcal{S}_* \subseteq \mathcal{S}$ are defined.

The objective of the Task Planning problem is to find a policy π, which is a sequence of ordered actions, that moves the initial state σ_0 into a goal state $\sigma_* \in \mathcal{S}_*$.

4. Method

Given an input assembly, we consider two tasks:

- *Sequence Prediction*: Infer sequences of possible placement actions, leading to the complete assembly.
- *Feasibility Prediction*: Infer the feasibility of the assembly in the target robotic system.

For this purpose, we encode the assemblies as graphs and design a pipeline comprising of two major components: a GNN, predicting the assembly sequence in a step-by-step setting, and an NF model, which uses a latent representation created by the GNN pipeline to predict the assembly's feasibility likelihood.

This chapter is structured as follows. We start by describing the formal problem setting for both tasks in Sections 4.1.1 and 4.1.2. We continue and present our Assembly Graph representation (Section 4.2) and Graph Assembly Network pipeline (Section 4.3) and show how we use these for Sequence Prediction in Section 4.4. Finally, we present our NF model for Feasibility Prediction in Section 4.5.1.

4.1. Problem Setting

4.1.1. Sequence Prediction

Our objective is to predict an ordering, or a sequence, of the assembly parts, such that it can be followed by a robotic system. Meaning, this predicted sequence should be feasible, i.e. take into account the kinematic constraints of the system. We use a simulator to generate a training set consisting of expert demonstrations, where each assembly is accompanied by all its possible feasible sequences.

Assemblies

We follow Rodriguez et al. (2020) and describe assembly A as a finite set of N parts $P = \{p_1, \ldots, p_N\}$, where each part is composed of a set of K_i surfaces $S_i = \{s^i_1, \ldots, s^i_{K_i}\}$. Both parts and surfaces are specified by their type and an additional unique numeric identifier. Surfaces are further characterized by their relationship to each-other $R(s_a^i, s_b^j)$, with $s_a^i \in S_i$ and $s_b^j \in S_j$. The surface distance relationship defines the relative position of parts in the assembly.

Decision Process

Following Lin et al. (2022) and using Task Planning formulation (Section 3.4), we describe the Sequence Prediction task for assembly A as a Markov Decision Process (MDP) (Bellman, 1957) $\{\mathcal{S}, \mathcal{A}, p\}$[1]:

$\mathcal{S} = \{\sigma_t \in \{0,1\}^N\}$ is the state set. State σ_t defines which parts are already in their target position (i.e. assembled), while the rest are pending their placement:

$$\sigma_t[i] = \begin{cases} 1 & \text{if part } p_i \text{ is in its target position} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

$\mathcal{A} = \{\alpha \in N\}$ is the part placement action set. The action α place the part p_α in its target position.

The transition function p defines the probability that action α in state σ_t will lead to state σ_{t+1}:

$$p(\sigma_{t+1}) = Pr(\sigma_{t+1} | \sigma_t, \alpha_t) \tag{4.2}$$

p satisfies the Markov property (Bellman, 1957), since given (σ_t, α_t), the state σ_{t+1} is conditionally independent of all previous states and actions.

Starting from the *initial* state σ_0, multiple different sequences of placement actions leading to the *final* state σ_T, in which all N parts are in place, might be possible. However, for some assemblies, no sequence of actions can successfully lead to the final state due to the constraints of the robotic system. We refer to these assemblies as *infeasible*. Our objective is to learn a policy $\pi_\theta : \sigma_t \to \{\alpha_1, \dots, \alpha_j\},\ j \leq N$ which maps each state to a set of placement actions leading to a final state by following an expert.

Expert demonstrations

The expert demonstrations are given as trajectories τ for each assembly A in the dataset. The trajectory τ defines a set of possible placement actions for each of the states $\sigma_1, \dots, \sigma_T$:

$$\tau(A)_i = \left\{\sigma_i, \{\alpha_{i,k}^{exp}\}_{k=1}^{K_i}\right\},\ K_i \leq N,\ \forall i \in [1, T] \tag{4.3}$$

[1]We omit the definition of rewards $\mathcal{R}$, as these are redundant in our setting.

In this setting, our objective is to minimize the fully-supervised loss between the expert action set and the predicted action set:

$$\min_{\theta} \mathbb{E} \left[\sum_{i=1}^{T} \left\| \{\alpha_{i,k}^{exp}\}_{k=1}^{K_i} \setminus \{\alpha_{i,z}^{pred}\}_{k'=1}^{K_i'} \right\| \right] \tag{4.4}$$

Where $\mathbb{E}$ is the expectation over all training samples and $\setminus$ is the set difference operator.

As we will see later, our method is able to generalize and predict placement actions leading to the successful construction of unseen assemblies.

4.1.2. Feasibility Prediction

Our second objective is to predict the feasibility of a given assembly. For this purpose, we use Anomaly Detection (AD), which aims to detect anomalous samples by scoring their deviation from the *normal* instances observed during training (Yang, Zhou, et al., 2021). In our setting, the normal samples are the feasible assemblies and the abnormal are the infeasible ones.

Our underlying assumption is that abnormal samples are drawn from a distribution different from the training sample distribution and therefore could be detected using likelihood estimation (OoD detection) (Yang, Zhou, et al., 2021). We model the feasible assemblies distribution $P_{feasible}$, and then given an assembly A we can confirm if $P_{feasible}(A) < \delta$ for some threshold $\delta > 0$.

4.1.3. Assumptions

Similar to Rodriguez et al. (2020), we exploit the fact that related products are typically comprised of parts from a shared catalog. In our setting, this catalog is made of just a few metal atomic sub-types that could be assembled to create numerous structures. Consequently, reusing knowledge acquired by assembling specific products can expedite planning for future related products comprised of parts from the same catalog.

We consider metal assemblies positioned on a 2D plane (i.e. on a flat surface) and assume their parts to be mutually aligned. Assembly parts surfaces are either orthogonal or parallel to each other and geometrical distances are defined between parallel surfaces. We only consider relationships defined by the notion of distance between surfaces, excluding more advanced interactions such as "screwed" or "inserted"[2].

[2]Thus simplifying the representation used by Rodriguez et al. (2020), though these advanced interactions may introduce additional restrictions on the sequencing of the assemblies.

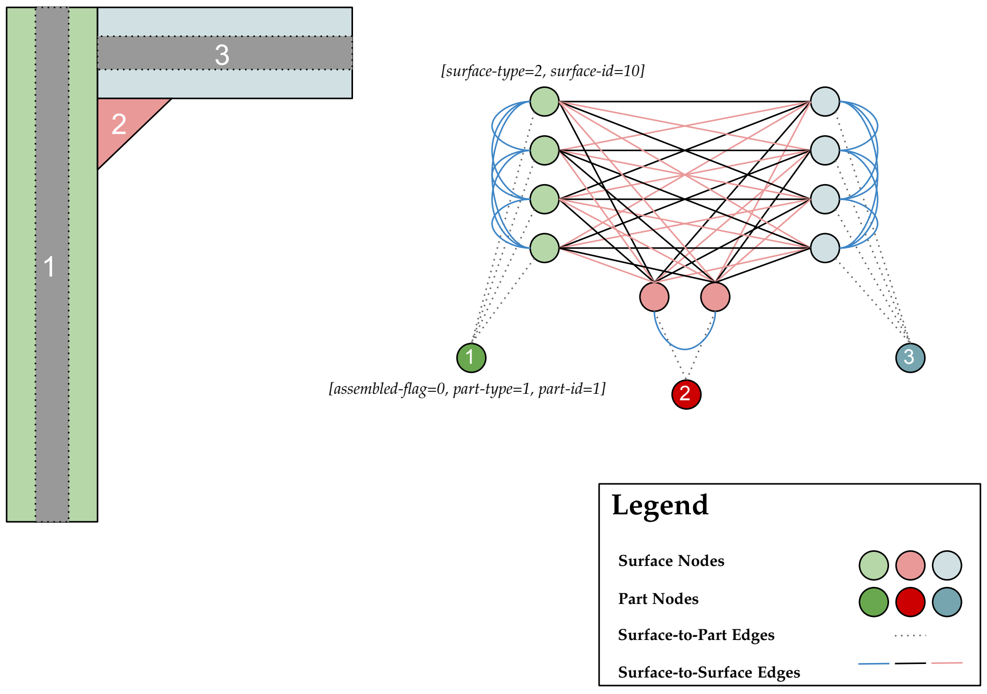


Figure 4.1.: A 3-part assembly (left) and a matching assembly graph (right) with example feature vectors for two nodes. Best viewed in color.

4.2. Assembly Graphs

We represent the overall structure of an assembly using *distances* between its part surfaces. Using only relative distances instead of absolute positions in the plane makes this representation agnostic to the rotation and mirroring of the assembly structure.

Given an assembly A, we represent it at state σ_t as an undirected heterogeneous graph $\mathcal{G}_t = (\mathcal{V}, \mathcal{E})$ (Section 3.2) containing two types of nodes: part nodes $\mathcal{V}^p$ and surface nodes $\mathcal{V}^s$, and two types of edges: $\mathcal{E}^{s\text{-to-}s}$, connecting all surface edges in the graph to each other and $\mathcal{E}^{s\text{-to-}p}$, connecting each surface node to its respective part node. Nodes and edges are optionally associated with additional multi-dimensional feature vectors $\phi(v) \in \mathbb{R}^{d_v}$ and $\phi(e) \in \mathbb{R}^{d_e}$ respectively[3]. An example assembly graph is depicted in Fig. 4.1.

[3]We abuse the notation and reuse $\phi()$ to symbolize the input feature vectors of different graph entities.

Part Nodes

Part nodes are the ones responsible for encoding the current state of the assembly. A part node $v_i^p \in \mathcal{V}^p$ is associated with a matching vector $\phi(v_i^p) \in \mathbb{R}^3$ comprising of the features *[assembled-flag, part-type, part-id]*:

- *assembled-flag*: a $1d$ binary value, indicating if the respective part is placed in its target position at the current state of the assembly. For example, in the initial state of the assembly graph, σ_0, all these flags are set to zero, while in the final state σ_T all are set to one.
- *part-type*: numeric value field which is assigned with one of the following $1d$ values: 0 for *short-profile*, 1 for *long-profile* and 2 for *angle-bracket*.
- *part-id*: natural numeric value in the range $[0, N-1]$, uniquely identifying a part in the given assembly.

Surface Nodes

A surface node $v_i^s \in \mathcal{V}^s$ is associated with a matching vector $\phi(v_i^s) \in \mathbb{R}^2$ comprising of the features *[surface-type, surface-id]*:

- *surface-type*: assigned with with $1d$ natural numeric value in the range $[0, 4]$ according to the following keys: *short-profile-short-surface*, *short-profile-long-surface*, *long-profile-short-surface*, *long-profile-long-surface*, *angle-bracket-lateral-surface*. We do not distinguish between parallel surfaces of the same part (for instance, the two lateral surfaces of the bracket), to make our approach agnostic to mirroring of individual parts.
- *surface-id*: similar to *part-id*, each surface is assigned with an identifier in the range $[0, \text{number-of-surfaces-in-assembly} - 1]$.

Positional Encoding

Both the *part-id* and *surface-id* fields are encoded with a d-dimensional Sinusoidal Positional Encoding (Vaswani et al., 2017). Let t be the desired part or surface id, we define its encoding $p_t \in \mathbb{R}^d$ as:

$$\forall i \in [1, d] : \ p_t[i] = \begin{cases} \sin(\omega_k \cdot t) & \text{if } i = 2k \\ \cos(\omega_k \cdot t) & \text{if } i = 2k+1 \end{cases} \tag{4.5}$$

$$\omega_k = \frac{1}{10000^{2k/d}} \tag{4.6}$$

We number surfaces clockwise, starting from the top surface, while we number parts from the one closest to the environment origin.

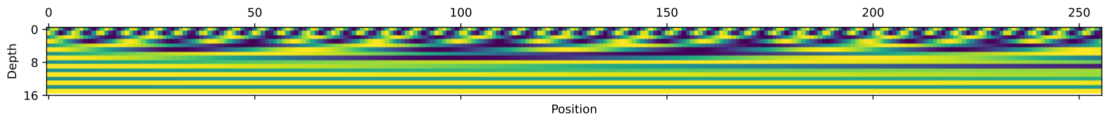


Figure 4.2.: Sinusoidal Positional Encoding (Vaswani et al., 2017) used by our model. Each column represent a single 16d embedding.

Surface-to-Surface Edges

The set of surface nodes $\mathcal{V}^s$ is a complete graph. Meaning, each surface node pair $(v_i^s, v_j^s) \in \mathcal{V}^s \times \mathcal{V}^s$ is connected with a surface-to-surface edge $e_{ij}^{s\text{-to-}s} \in \mathcal{E}^{s\text{-to-}s}$. This edge is assigned with a 1d feature vector $\phi(e_i) \in \mathbb{R}$, indicating the relation between the two surfaces (color coded in Fig. 4.1). If the two surfaces belong to the same part, the value of this feature is 1 (blue edges). If the two surfaces are orthogonal, the value of this feature is -1 (light red edges). Otherwise, the surfaces are parallel and the value is their respective geometric distance in centimeters (black edges). Surface nodes also have self-loop edges with a feature of 0 (removed from the figure for brevity).

Surface-to-Part Edges

Each surface and part node pair $(v_i^s, v_j^p) \in \mathcal{V}^s \times \mathcal{V}^p$, where surface v_i^s belongs to the part v_j^p, is connected with a surface-to-part edge $e_{ij}^{s\text{-to-}p} \in \mathcal{E}^{s\text{-to-}p}$. This edge is not associated with a feature vector.

4.3. Graph Assembly Network

Our model inputs an assembly A at state σ_t represented as a graph $\mathcal{G}_t$ (Fig. 4.3). Inspired by Lin et al. (2022), we model the long-horizon assembly sequencing problem as a step-by-step binary classification per each part in the assembly. At each step, the model outputs a score per part, reflecting the probability of placing it in its target position in the current state of the assembly.

Surface and Part Blocks

The architecture is made of identical blocks, which are applied sequentially to obtain updated node representations. Each block is made of a GAT GNN (Brody et al., 2021;

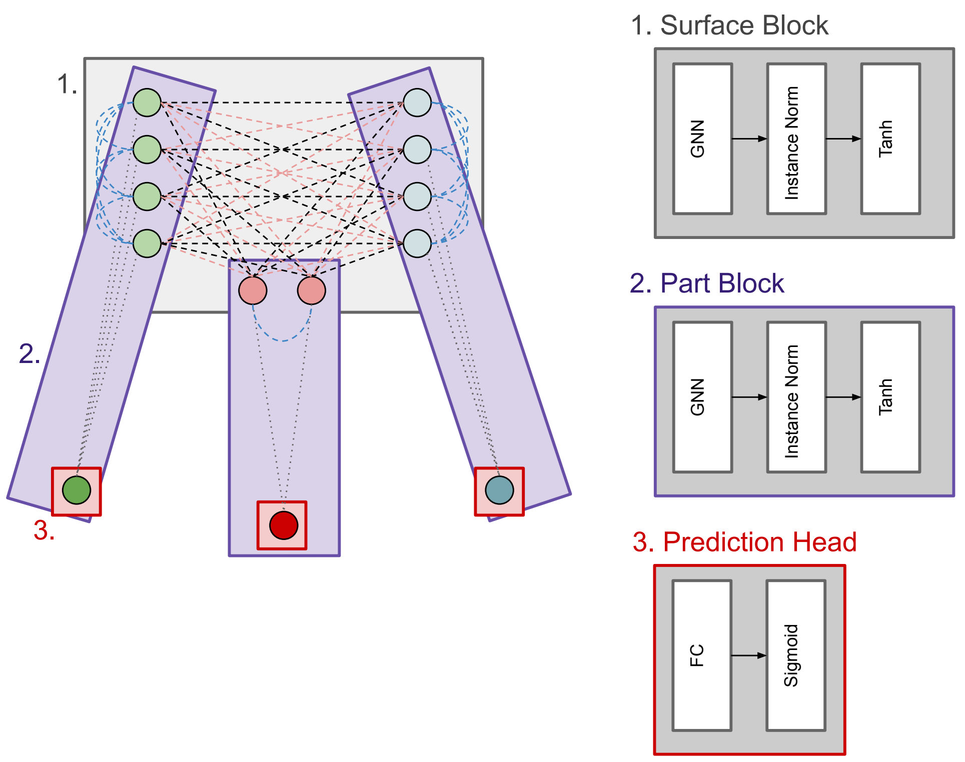


Figure 4.3.: The Graph Assembly Network. First, the Surface Block (1) is applied on the surface nodes (gray box). Next, the Part Block (2) is used to pool information from the surface nodes into the part nodes (purple boxes). Finally, the Prediction Head (3) is applied on the part nodes (light red boxes) to obtain a probability score per part.

Veličković et al., 2017) (Section 3.1), an Instance Normalization layer (Ulyanov et al., 2016) and a Tanh function.

Surface Blocks are applied on surface nodes and surface-to-surface edges and output updated surface node features. Then Part Blocks are applied on surface nodes, part nodes and surface-to-part edges to obtain updated part node features.

Instance Normalization is computed as follows:

$$y = \frac{x - \mathbb{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta \tag{4.7}$$

Where x and y are the input and output node features respectively, the mean and variance are computed per dimension of the nodes in the particular graph of the batch, γ and β are learned parameters and ϵ a small constant.

As an additional regularization mechanism, we apply *Dropout* (Srivastava et al., 2014) to prevent the network from over-fitting the training set during knowledge transfer experiments. This is performed by letting a Bernoulli distribution randomly set the GNN weights to zero during training.

Prediction Head and Loss Function

To obtain a score per part, a fully-connected layer followed by a Sigmoid function is applied on each part node.

During training, we compare the network output to the expert trajectories using binary cross-entropy. Our objective function (Eq. 4.8) includes an additional regularization term (Eq. 4.9) aimed at encouraging the network not to predict the placement of parts that are already in their target position (as reflected in the part nodes *assembled-flag* input feature).

$$L_\Theta = \sum_{i=1}^{N}\sum_{j=1}^{N_i} \left(\hat{y}_{ij} \cdot \log(y_{ij}) + (1-\hat{y}_{ij})\log(1-y_{ij})\right) + \delta L_{\text{reg}} \tag{4.8}$$

$$L_{\text{reg}} = \sum_{i=1}^{N}\sum_{j=1}^{N_i} a_{ij} \cdot y_{ij} \tag{4.9}$$

Where N is the number of graphs in the dataset, N_i is the number of nodes in the i-th graph, y_{ij} and $\hat{y}_{ij}$ the output probability score of the model and the ground-truth for the j-th node in the i-th graph respectively, δ a weighing coefficient and lastly a_{ij} the value of the *assembled-flag* in the input features of the t-th node of the i-th graph.

Training Setting

As described in Section 4.1.1, we collected M demonstrations of the expert solving assembly sequencing problems. At each state of the assembly σ_t, we gather the set of placement actions performed by the expert $\alpha_t = \{\alpha_t^{exp}\}_{k=1}^{K}$ across all of its demonstrations. We assemble a dataset made of assemblies, their state and the matching expert action ground-truths:

$$D = \{\tau_j\}_{j=1}^{M},\ \tau_j = \{(A, \sigma_t, \alpha_t)\}_{i=1}^{T} \tag{4.10}$$

In the training stage, we randomly sample a batch of these τ tuples and build their matching assembly graph $\mathcal{G}_t$, representing the state of assembly A at state σ_t. We also create a ground-truth one-hot vector $\hat{y} = \mathbb{1}[\hat{y}_i \in \alpha_t]$ based on the expert actions, in which 1 is assigned to all parts chosen by the expert in α_t and 0 for all others. Finally, we compare the model predictions to the ground-truth using the objective in Eq. 4.8.

We used PyTorch Geometric (PyG) (Fey & Lenssen, 2019) to build the model and PyTorch Lightning (Falcon & The PyTorch Lightning team, 2019) to train it. Our model includes 51.7K trainable parameters. Table 4.1 specifies the hyper-parameters we used for training.

Parameter	Value
Batch Size	256
Learning Rate	0.002182
Optimizer	Adam
Training epochs	33
Regularization δ	0.3
Positional Encoding Length	16
Hidden Channels	94
# Surface Blocks	3
# Part Blocks	1
Surface/Part Blocks Leaky ReLU Slope	0.15
Surface/Part Blocks Dropout Probability	0.02
Surface/Part Id Positional Encoding Size	16d

Table 4.1.: Hyper-parameters used for model training.

4.4. Sequence Prediction

In the setting presented in Section 4.3, our model predicts in each state of the assembly the possible next placement actions. To infer complete assembly sequences (i.e. of length N), we apply the model on the initial-state assembly graph while repeatedly placing parts in their target positions following the model predictions. We define this process in Algorithm 1 in which we traverse the assembly state tree using Depth First Search (DFS).

For each assembly A in the test set, we execute Algorithm 1 on the graph in its initial state $\mathcal{G}_0$, meaning, when all part nodes *assembled-flag*s are set to zero. First, we check the

Algorithm 1 Assembly State Tree Traversal

```
function TRAVERSE-TREE(Model $M$, Assembly Graph $\mathcal{G}_t = (\mathcal{V}, \mathcal{E})$, Threshold $\lambda$)
    $S \leftarrow$ list()
    if $(\forall v \in \mathcal{V} :\ v.assembled\text{-}flag == 1)$ then
        return $S$                                        ▷ Exit: all parts assembled
    end if
    $\mathbf{y}_{\text{pred}} \leftarrow M(\mathcal{G}_t)$
    for $i \leftarrow 1$ to $|\mathcal{V}|$ do
        if $\mathbf{y}_{\text{pred}}[i] < \lambda$ then
            continue
        end if
        $\mathcal{G}_{t+1} \leftarrow$ copy$(\mathcal{G}_t)$
        $[\mathcal{V}_{t+1}]_i.assembled\text{-}flag \leftarrow 1$          ▷ Set part node $i$ as assembled
        $S_* \leftarrow$ TRAVERSE-TREE$(M, \mathcal{G}_{t+1}, \lambda)$                 ▷ Recursion call
        for $s$ in $S_*$ do
            $s_* \leftarrow [i] + s$               ▷ Add current part at the head of the sequence
            $S$.append$(s_*)$
        end for
    end for
    return $S$
end function
```

exit condition of the recursion, if all parts are already in place. Next, we call the model on the graph to retrieve its prediction per part node $\mathbf{y}_{pred} \geq 0$. We then iterate over the part nodes while skipping ones for which the score is lower than a predefined threshold λ. This threshold helps reduce the overall run-time; as we only follow promising paths. For each node v, we follow the model prediction and set it as assembled. We then call the recursion on the altered graph to retrieve possible sequences starting with the chosen node. Finally, we add v to the head of each returned sequence.

We define the set of predicted complete sequences as:

$$\mathcal{S} = \{\, s \in \textsc{Traverse-Tree}(M, \mathcal{G}_0, \lambda) \mid |s| = N \,\} \tag{4.11}$$

Where N is the number of parts in A.

4.4.1. Sequence Score

For a given sequence $\mathscr{s}$ predicted by the algorithm, we define its *Sequence Score* $\gamma_{\mathscr{s}}$ as the minimal step probability along the predicted sequence:

$$\gamma_{\mathscr{s}} = \min(\mathbf{y}_{\mathscr{s}}) \tag{4.12}$$

This score measures the confidence in the "weakest link" along the predicted sequence; the action about which the model is least certain. This quantity is appropriate in our setting since its magnitude is independent of the sequence length, allowing evaluation of the model performance on differently sized assemblies[4].

For example, let $\mathscr{s} = [4, 2, 3, 1, 5]$ a predicted assembly sequence. Let the respective probabilities for each of the actions $\mathbf{y}_{\mathscr{s}} = [0.9, 0.87, 0.84, 0.81, 0.95]$. The Sequence Score is then $\gamma_{\mathscr{s}} = \min(\mathbf{y}_{\mathscr{s}}) = 0.81$.

4.5. Feasibility Prediction

4.5.1. Normalizing Flows

As outlined in Section 4.1.2, our second objective is to predict the feasibility of assemblies. Since we frame this task as AD and our input is represented as graphs (Section 4.2), we take a graph-level AD approach (X. Ma et al., 2021). We propose a model trained on feasible assembly graphs that can identify infeasible ones as abnormal. For this purpose, we used NF (Section 3.3) in an OoD (Kirichenko et al., 2020) setting to model the distribution of feasible assemblies.

Feature Extractor

We use our Graph Assembly Network (Section 4.3), pre-trained on feasible assemblies, as a graph-level feature extractor by applying a channel-wise mean pooling on each graph part node's embeddings. This setting creates a single latent graph representation whose number of dimensions is independent of the number of assembly parts.

NF model

We used the common RealNVP NF architecture (Dinh et al., 2016) (Section 3.3) comprising multiple layers of affine coupling flows. We pass feasible assemblies latents through

[4]We experimented with other aggregation functions as well, see Section 5.4.1 for more details.

these flows to reach a multivariate Gaussian distribution with a diagonal covariance matrix, parameterized using learned mean and variance.

During training, we optimized the log-likelihood objective in Eq. 3.15. The validation set was comprised of both feasible and infeasible assembly graphs latents, allowing us to select a model which maximized the Area Under the Receiver Operating Characteristic Curve (ROC AUC) score (see Section 5.2.2), i.e. optimized class separation on this set. This is still an AD setting since although the validation set includes both classes, it was only used for model selection.

The NF model predicts for a given test assembly its log-likelihood score. If $LL(A) < \delta$ for some threshold $\delta > 0$, we predict A is infeasible. The model includes $16.8M$ trainable parameters. Table 4.2 specifies the hyper-parameters we used for training.

Parameter	Value
Batch Size	32
Learning Rate	$1e-5$
Training epochs	32
Optimizer	Adam
Hidden Channels	94
# Flows	749
Shift & Scale network layers	4
Shift & Scale hidden channels	94

Table 4.2.: Hyper-parameters used for the Normalizing Flows model training.

4.5.2. Baseline Classifier: Sequence Set Size

As a reference to our proposed NF-based OoD detector, we used our Sequence Predictor (Section 4.4) trained on both feasible and infeasible assemblies to derive the feasibility of assembly graphs, making this a binary classifier (in contrast to the single-class NF model). In this setting, the size of the predicted Sequence Set indicates the assembly feasibility.

We let our Graph Assembly Network pipeline predict the set of sequences $\mathcal{S}_{\mathcal{A}}$ for a given test assembly A. If no sequence was predicted, i.e. $|\mathcal{S}_{\mathcal{A}}| = 0$, the assembly is predicted *Infeasible*, otherwise, it is predicted *Feasible*. We use the Sequence Score (Section 4.4.1) as a threshold controlling the size of the predicted set: $\forall \mathcal{s} \in \mathcal{S}$ require $\gamma_{\mathcal{s}} > \delta$ for some $\delta > 0$.

5. Experiments and Results

In this chapter, we present the details of our experimental setup and summarize our results. We first describe the dataset we created (Section 5.1) and the evaluation metrics we employed (Section 5.2). We then present the results of our experiments on the Sequence Predictor (Section 5.3), including evaluations in various knowledge transfer tasks and an ablation study of the GNN positional encodings. Finally, we cover the Feasibility Predictor (Section 5.4), comparing it to other classifiers, examining the limitations of the graph latent representation, and conducting extensive ablation studies. Additional detailed results can be found in Appendix A. In the following, we note with A_i assemblies with i parts.

5.1. Dataset

5.1.1. Acquiring a dataset

To perform our experiments, a dataset of synthetic assemblies and their matching sequences was created using the in-house simulation software MediView. Each of the assemblies is made of up to three atomic part types: *long profile*, *short profile* and *angle bracket*, the latter allows attachment of profiles to each other. Assembly parts are associated with their matching surfaces; four surfaces for profiles (two *long* and two *short*) and two for brackets[1] (*lateral*).

For example, in Fig. 5.1, the blue and red surfaces are parallel and have a distance of 2 cm, whereas the blue and orange surfaces are in contact and therefore have a distance of 0 cm. The blue and green surfaces are on the same hyperplane and therefore have a distance of 0 cm as well. Finally, the blue and magenta surfaces are orthogonal and therefore not assigned with a distance.

For each assembly, the simulation software was tasked with putting together the structure by iteratively attempting all $N!$ part placements, while considering the restrictions imposed by the capabilities of a *KUKA LBR Med* (Fig. 1.1) target robotic system and restrictions of the working environment.

[1]The bracket's remaining diagonal surface is omitted since it is not aligned with profile surfaces.

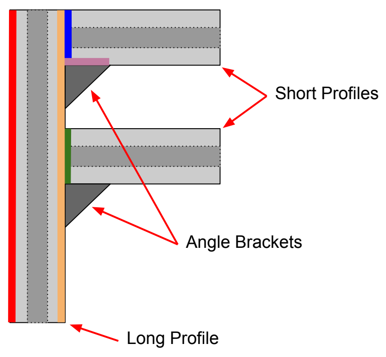


Figure 5.1.: An example 3-part assembly (best viewed in color).

We process the simulation output to obtain two types of supervision signals for assembly A:

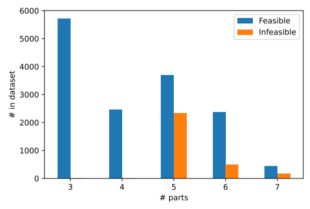
1. **Placement Action Trajectory** In a given state of an assembly σ_t, which placement actions $\{\alpha\}_{k=1}^{K}$, meaning, choices for the next part, led to successful construction.
2. **Assembly Feasibility** If at least one of the simulation sequences is successful, the assembly is labeled as *feasible*. If, on the other hand, no sequence led to success, meaning the simulation is unable to create the structure, the assembly is labeled as *infeasible*.

5.1.2. Dataset characteristics

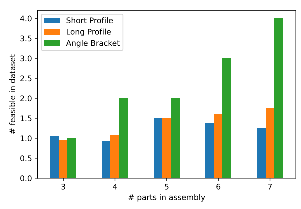
Our dataset is comprised of randomly generated specifications of 18000 feasible assemblies and 2000 infeasible ones. Assemblies are made of between 3 and 7 parts (Fig. 5.2a), with a combination of different atomic types (Fig. 5.2b).

We further examine the number of ground-truth sequence trajectories per assembly (Fig. 5.2c). Most A_5 assemblies have five sequence trajectories each, while some have only two. A lower number of sequences implies a more constrained assembly, one which is harder to put together. For the infeasible assemblies, incomplete sequences attempted by the simulation environment are available, i.e., ones leading to a *failure state*, in which the structure is incomplete, but no remaining part could be placed due to constraints.

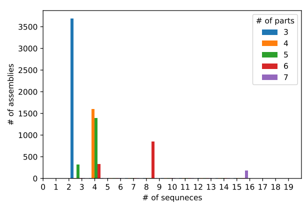
As the overall number of parts in the assembly increases, so does the maximal distance between their surfaces, since the overall size of the structure also grows



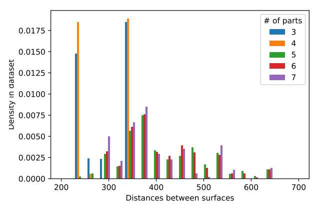
(a)



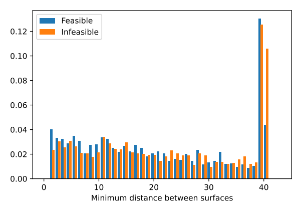
(b)



(c)



(d)



(e)

Figure 5.2.: (a): Parts breakdown in feasible and infeasible assemblies. (b): Number of part types in feasible assemblies. (c): Number of ground-truth sequence trajectories per assembly. (d): Maximal distance between assembly surfaces. (e): Minimal distance between surfaces (excluding zero) vs. assembly feasibility.

(Fig. 5.2d). We have not observed a correlation between the surfaces' minimal distances and the assembly feasibility (Fig. 5.2e).

5.2. Evaluation Metrics

We used the following metrics to evaluate our method:

- **Sequence Prediction:**
    1. **Step-by-Step:** Examine the model's predictive performance to infer the parts that should be assembled next given the current state. Evaluated using a Precision-Recall curve and Area Under Curve (AUC) score.
    2. **Complete-Sequence:** Examine the model's predictive performance to infer the entire set of assembly sequences. Evaluated using Information Retrieval (IR) Precision-Recall curve and AUC score.
    3. **Precision@k:** Measures the model's ability to rate the confidence it assigns to its Top-*k* sequences. Meaning, how likely are the *k*-highest scored sequences in the ground-truth set.

- **Feasibility Prediction:** We evaluate the ability of the different models to separate between the feasibility classes with Area Under the Receiver Operating Characteristic Curve (ROC AUC).

5.2.1. Sequence Prediction

Step-by-Step Metrics

In this setting, we are comparing the model predicted binary class per part node (*assembled-flag*) to the ground-truth and therefore chose to evaluate it using established classification metrics. We apply a threshold on the model predictions and compare it with the ground truths to derive the step-by-step *Precision* and *Recall* metrics:

$$\text{Precision} = \frac{tp}{tp + fp} \tag{5.1}$$

$$\text{Recall} = \frac{tp}{tp + fn} \tag{5.2}$$

For example, for a 5-part assembly with ground-truth $\mathbf{y} = [1, 0, 1, 1, 0]$ (meaning, the expert chooses to assemble parts 1, 3 and 4), the model outputs the prediction $\mathbf{y}_{\text{pred}} = [0.99, 0, 0.3, 0.7, 0.55]$. Notice how the output probabilities do not sum up to 1, as each is an independent result of a Sigmoid function. Assuming a threshold of $\lambda = 0.5$ we get the prediction $\hat{\mathbf{y}}_{\text{pred}} = [1, 0, 0, 1, 1]$, meaning two True Positives (tp) for parts 1 and 4, one True Negative (tn) for part 2, one False Positive (fp) for part 5 and one False Negative (fn) for part 3. We can now report both Precision and Recall of $2/3$.

Note that the step-by-step $\mathbf{y}_{\text{pred}}$ is different from the sequence $\mathbf{y}_{\mathscr{s}}$ we used to derive the Sequence Score in Section 4.4.1. While the former is the prediction of the model per part for a given state of the assembly, the latter is a probability predicted per placement action chosen by the Sequence Predictor along the sequence $\mathscr{s}$.

We compute the macro average for the Precision and Recall metrics per assembly size, meaning calculate it separately and then average the results to prevent biases against smaller assemblies. In addition, we plot a Precision-Recall curve with a varying threshold value and report the AUC score[2] in order to eliminate the dependency on the chosen threshold parameter.

[2]The Area Under the Precision-Recall Curve is also called *Average Precision* in some publications (M. Zhu, 2004). We use the term AUC to prevent confusion.

Information Retrieval Metrics

Metrics used in Information Retrieval (IR) compare sets of predictions vs. a ground truth, e.g., websites retrieved by a search engine (Croft et al., 2010). In our setting, we treat a correctly predicted sequence as one having an identical counterpart sequence in the ground truth, as we are interested in following the expert demonstrations as closely as possible. Partial matches between some steps in the predicted and ground truth sequences are not beneficial as they may be infeasible given incorrectly placed parts[3].

IR-Precision and IR-Recall Two sets are defined: *Retrieved Documents* (RET) and *Relevant Documents* (REL). Retrieved documents could be websites returned for a given search query. In this case, relevant documents are all websites on the internet relevant to the topic. In our setting, the retrieved documents are the set of sequences $\mathcal{S}$ returned by Algorithm 1, and the relevant documents are the set of ground truth sequences.

IR-Precision and *IR-Recall* (Croft et al., 2010) are defined :

$$\text{IR-Precision} = \frac{|RET \cap REL|}{|RET|} \tag{5.3}$$

$$\text{IR-Recall} = \frac{|RET \cap REL|}{|REL|} \tag{5.4}$$

IR-Precision is the proportion of retrieved documents that are relevant, it measures the model's ability to reject irrelevant documents in the retrieved set. *IR-Recall*, on the other hand, is the proportion of relevant documents that are retrieved. Meaning, how well is the model able to find all relevant documents. Similar to the step-by-step precision-recall curve, we report here the AUC.

Precision@k There is an additional benefit in a model that is able to properly rank its retrieved documents set. For instance, in our setting, having more confidence in the top-1 sequence, since in the real world we would be interested in putting the assembly together only once. Precision@k (P@k) measures the proportion of top-k retrieved documents that are relevant (Herlocker et al., 2004)[4]:

$$TOP(k) = \textbf{sort}(RET)[:k] \tag{5.5}$$

$$\text{P@k} = \frac{|TOP(k) \cap REL|}{|TOP(k)|} \tag{5.6}$$

[3]We argue that the predicted sequence $\mathfrak{s} = [4,2,3,1]$ for the ground truth $\mathfrak{z} = [1,2,3,4]$ has no benefits, since, for example, the partial match $[2,3]$ may be infeasible given part 4 is already placed.

[4]An analogous definition of Recall@k (R@k) exists (Herlocker et al., 2004), but is problematic and therefore not used here. It considers the proportion of relevant documents that are retrieved in the top-k and therefore highly dependent on $|RET|$. A perfect model will have a small R@k when k is much smaller than $|REL|$.

Where **sort**() ranks the retrieved sequences by their Sequence Score (see Section 4.4.1).

In practice, k might be larger than $|RET|$, introducing biases into the computation of Eq. 5.6, since we would not be able to distinguish between cases in which the predicted sequences are false to ones in which fewer sequences were predicted in the first place. We argue that the first case is more harmful in our setting and set $k = \min(k, |RET|)$. It does not make sense to compute P@k when $k < |REL|$ because even a perfect model can only achieve $(k/|REL|) < 1$.

For instance, consider the case in which we are interested in computing $P@3$ when there are five ground-truth sequences ($|REL| = 5$) and the model predicted only two sequences ($|RET| = 2$), with one of these correct. In this case, we set $k = min(3, 2) = 2$ and derive $P@3 = 1/2$.

5.2.2. Feasibility Prediction

In this setting, we compare binary classifiers that assign each assembly with one of two possible classes: *Feasible* or *Infeasible*. For this purpose, we plot the commonly used Receiver Operating Characteristic (ROC) curve, which compares the classifiers' True-Positive Rate (*tpr*) to the False-Positive Rate (*fpr*), and derive a matching AUC score, where:

$$tpr = \frac{tp}{p} \tag{5.7}$$

$$fpr = \frac{fp}{n} \tag{5.8}$$

Notice that in this setting, a positive (p) instance is a feasible assembly and a negative (n) is an infeasible one, different from the labeling presented above in Section 5.2.1.

5.3. Sequence Prediction Task

In our first experiment, given an assembly, we are interested in predicting its assembly sequences.

5.3.1. Baseline Setup

We train our Graph Assembly Network (Section 4.3) and test it on the sequence prediction task. We evaluate the model in the step-by-step and complete-sequence prediction settings. As a baseline, we used 4-fold cross-validation and trained the model on a feasible assembly dataset comprised of a random mixture of assembly sizes. We then tested the model separately per assembly size.

In each column of Fig. 5.3 we report the results for a given assembly size, including the Precision-Recall curve in the complete-sequence setting (including the AUC) and P@k scores for $k \in [1 \ldots 10]$ with a threshold of 0.5. Notice that P@k is reported for A_i only when the assembly has at least k ground truth sequences (see discussion in Section 5.2.1).

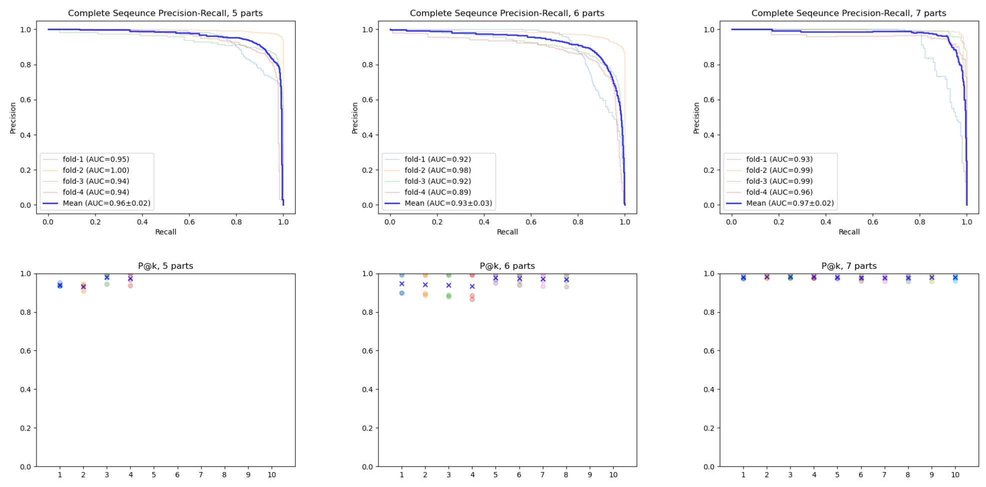


Figure 5.3.: Cross Validation results on the Sequence Prediction task in the baseline setup. In each column the baseline model is evaluated on a different A_i.

When inspecting the results, we see that the model achieves good results in this setting, reflected in a high mean AUC score across the board. Interestingly, the complete-sequence mean AUC score for A_5 and A_6 (0.96 and 0.93) is lower than the one achieved for A_7 (0.97). One would have expected that since A_7 has many more ground truth sequences (Fig. 5.2c), their prediction with higher recall is harder. We claim the opposite is correct: a smaller number of ground truth sequences also reflects restrictive constraints on the assembly. More indications for this are given when comparing Fig. 5.2c to the P@k results of A_5 and A_6, where a rise in precision is observed between $P@2 - P@3$ and $P@4 - P@5$. Some assemblies in A_5 and A_6 have fewer ground truth sequences than the rest since they are more constrained and harder for the model to predict, lowering the corresponding P@k scores.

5.3.2. Knowledge Transfer

A significant ability of our model is to generalize knowledge between different assembly tasks. We demonstrate this by evaluating models trained on differently sized assemblies, as previous methods (Rodriguez et al., 2020; Wells et al., 2019) are incompatible with this setting.

In the first setting, *many-to-one* (Fig. 5.4), we trained a model on a dataset comprised of a mixture of assembly sizes but i, i.e., $A_{\forall j \neq i}$. We then evaluated this model on A_i alone. For instance, the model in the first column of Fig. 5.4 was trained on $A_{\forall j \neq 5}$ and is evaluated on A_5. When comparing the results in Fig. 5.4 to the baseline, we observe a reduction of an average of 4.6 points in AUC between the settings, which could be expected. However, we can also observe $P@1$ and $P@2$ of ≈ 0.9, indicating our method is beneficial in this setting.

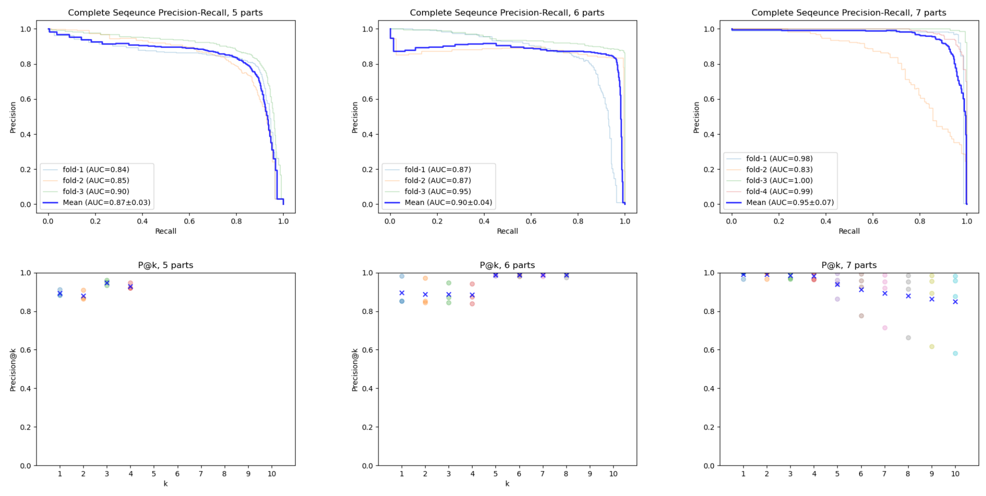


Figure 5.4.: Cross Validation AUC scores on the Sequence Prediction task in the knowledge transfer *many-to-one* setting. In each column a different model, trained on $A_{\forall j \neq i}$, is evaluated on A_i.

We shed light on this generalization in a second experiment setting, *one-to-many* (Fig. 5.5 and Table 5.1), which is reversed to the previous. Here, a model is trained A_i and then evaluated on $A_{\forall j \neq i}$, i.e., all assemblies with a size different than i. Looking into Fig. 5.5, we identify a clear pattern in which each model is able to obtain comparably

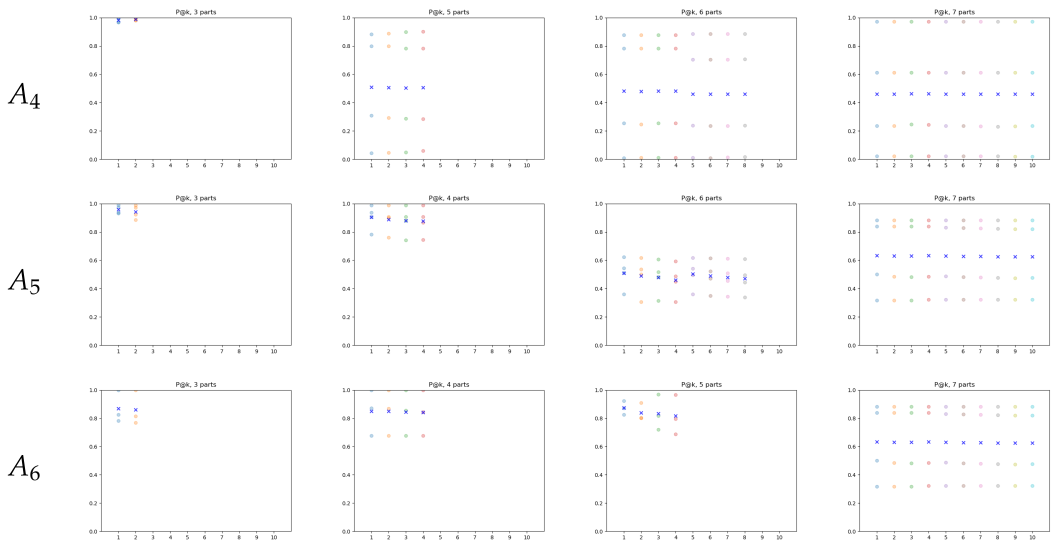


Figure 5.5.: Cross validated P@k results on the Sequence Prediction task in the knowledge transfer *one-to-many* setting. In each row a different model was trained on A_i alone and then evaluated on a different $A_{\forall j \neq i}$ per column. Each model is able to give valuable predictions on assemblies smaller than the ones it was trained on.

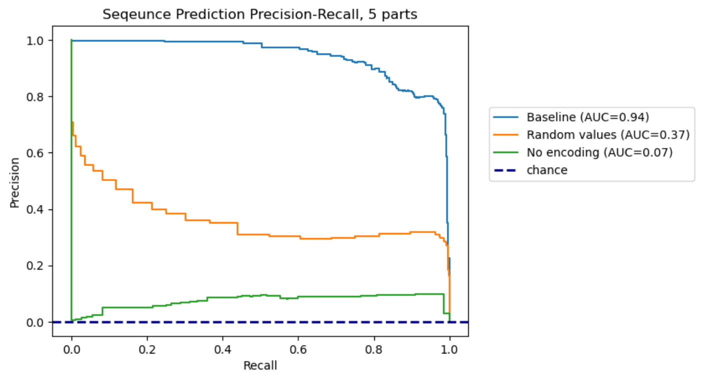
Training Set	Step-by-Step AUC (↑)					Complete Sequence AUC (↑)				
	A_3	A_4	A_5	A_6	A_7	A_3	A_4	A_5	A_6	A_7
A_4	0.92 ± 0.11		0.48 ± 0.12	0.41 ± 0.11	0.43 ± 0.12	0.93 ± 0.09		0.28 ± 0.12	0.25 ± 0.15	0.25 ± 0.15
A_5	0.93 ± 0.06	0.89 ± 0.07		0.78 ± 0.14	0.59 ± 0.16	0.83 ± 0.07	0.70 ± 0.09		0.36 ± 0.12	0.24 ± 0.07
A_6	0.90 ± 0.10	0.89 ± 0.11	0.93 ± 0.04		0.59 ± 0.16	0.73 ± 0.16	0.68 ± 0.24	0.71 ± 0.13		0.24 ± 0.07

Table 5.1.: Feasibility classifiers AUC scores in Knowledge Transfer *one-to-many* setting. In each row, a model trained on A_i is evaluated on different test sets $A_{\forall j \neq i}$.

high *P@k* results for smaller assemblies[5]. For instance, A_5 (third row) obtains high results for A_3 and A_4, which makes sense, as the constraints guiding the assembly of smaller structures are *contained* in larger structures[6]. The same pattern repeats with AUC in Table 5.1.

[5]We do not perform this experiment on A_7, as there are relatively small amount of assemblies with 7 parts in the dataset.

[6]We provide detailed P@k results in Table A.1.

(a)


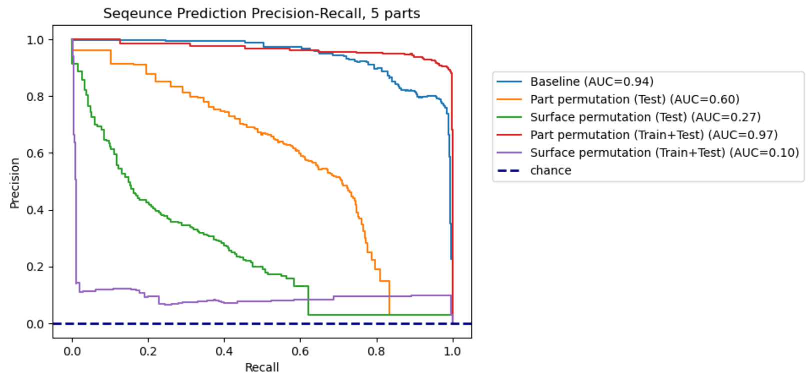
(b)


Figure 5.6.: Ablation study of part and surface positional encodings. In (a), we replace the encodings with random values or remove them entirely. In (b), we permute the order of parts and surfaces in test and training time, revealing their respective importance for the model's ability to grasp geometrical bias.

5.3.3. Ablation Studies

Positional Encoding

In our assembly graph (Section 4.2), both the part and surface node embeddings contain an id field which we represent as a $16d$ sinusoidal positional encoding (Vaswani et al., 2017). We conducted a series of experiments on A_5 to look into the contribution of these to our model and confirm they do not cause an over-fit to the training data.

First, we experimented with removing the positional encoding field altogether or replacing it with an identically lengthed random value. In Fig. 5.6a, positional encoding dramatically increases the performance of our model. We hypothesize that these positions introduce geometrical bias, i.e., understanding of the part and surface interactions, which is helpful in our task.

We number assembly parts and surfaces in a constant order. Parts are counted beginning from the one closest to the environment origin. Surfaces, on the hand, are always numbered clockwise, starting from the respective part top. As we see in Fig. 5.6b, permuting these in test time cause severe degradation in performance, indicating constant numbering harm the model's ability to generalize. However, introducing permutations into the training process produces surprising findings. While part order permutations make the model more robust in test time, allowing it to outperform the baseline, surface order permutations cause it to fail. We believe this demonstrates the importance of these features for the model's understanding of the parts' geometrical structure.

5.4. Feasibility Prediction Task

In the second experiment, given an assembly we are interested in predicting its feasibility class, i.e. *feasible* or *infeasible*.

5.4.1. Classifier Comparison

Normalizing Flows

We proposed using NF to model the distribution of feasible assemblies $P_{feasible}$. The feasibility class is obtained by comparing the model's predicted log-likelihood score to a predefined threshold. As elaborated in Section 4.5.1, our model uses a single-class training setting.

Baseline Classifier: Sequence Set Size

Our baseline feasibility classifier (Section 4.5.2) uses the size of the sequence set, predicted by the Sequence Predictor, as an indication of the assembly feasibility.

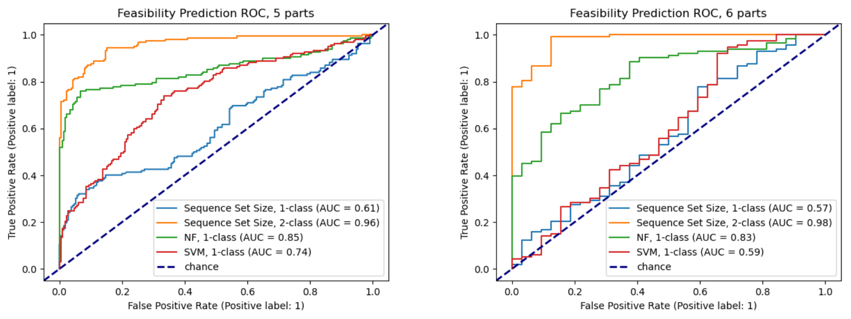
We trained this base model in two settings. First, the model was trained in a single-class setting on feasible assemblies alone. Second, a binary classification setting was used, by including both feasible and infeasible assemblies in the training set. For infeasible samples, we created a dummy trajectory, in which no part is chosen by the expert demonstrator in the assembly's initial state. We justify this design in a series of experiments in Section 5.4.4.

One-class SVM

As an additional reference to our method, we report the results of a One-class Support Vector Machine (OC-SVM) (Schölkopf et al., 1999) model[7], as it was used by previous graph-level AD works (H. T. Nguyen et al., 2020; L. Zhao & Akoglu, 2021). This model was trained in a similar setting to the NF, i.e., on the pooled node embeddings created by our GNN for feasible assemblies alone.

Classifier	AUC (↑)	
	A_5	A_6
Sequence Set Size, binary-class	0.96	0.98
NF, single-class	0.85	0.83
OC-SVM	0.74	0.59
Sequence Set Size, single-class	0.61	0.57

Table 5.2.: Feasibility classifiers AUC score on balanced test sets.



(a) (b)

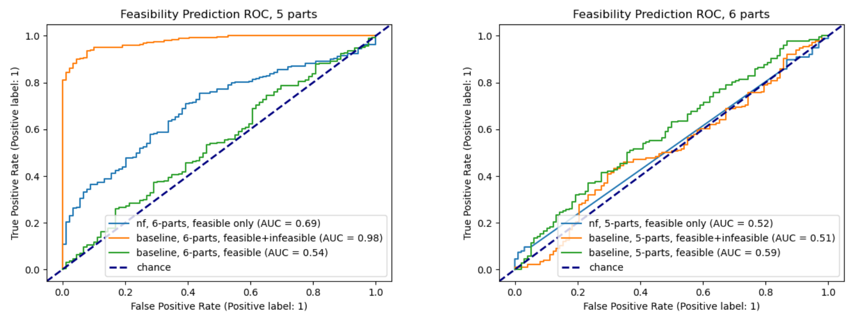
Figure 5.7.: Comparison of the different methods for feasibility classification for A_5 (a) and A_6 (b). The baseline model, based on Sequence Set Size, achieves the best results, but it is trained in a binary setting.

Evaluation

In Fig. 5.7 and Table 5.2, we compare the results of the three models described above in the feasibility perdition task for A_5 and A_6. The baseline classifier, trained in a binary

[7] An overview of OC-SVM is provided in Appendix B.

setting on both feasible and infeasible assemblies is able to best separate between the classes. Next in line is the NF model trained in a single-class setting. The OC-SVM works on the GNN embeddings directly and thus has only limited success. Finally, the baseline model, which also uses this single-class setting, is not much better than chance.



(a) (b)

Figure 5.8.: Ability of the feasibility classifiers to transfer knowledge. The classifiers in (a) were trained on A_6 and tested on A_5, whereas in (b) they were trained on A_5 and tested on A_6.

5.4.2. Knowledge Transfer

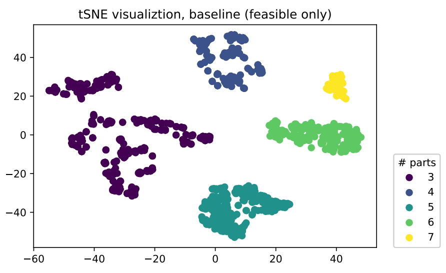
Similar to the experiments we ran in Section 5.3.2, we are interested in measuring the ability of our classification methods to transfer knowledge about feasibility between differently sized assemblies.

In Fig. 5.8, we train the classifiers on A_5 and test them on A_6 (b) and vice-versa (b). In line with the results in Section 5.3.2, we see that transfer of feasibility knowledge is possible only from larger assemblies to smaller ones.

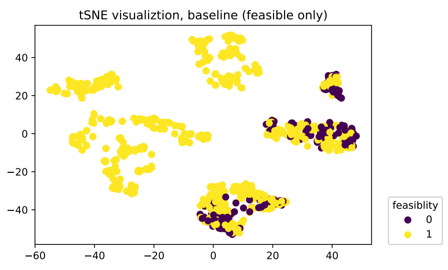
5.4.3. Latent Space

Though our NF model uses many parameters (Table 4.2), it still lags behind the binary-class setting. To better understand its limitations, we examine the graph's latent representation, i.e., the pooled part node embeddings. First, we visualize them in a $2d$ space with t-distributed Stochastic Neighbor Embedding (t-SNE) (Van der Maaten & Hinton, 2008).

Figure 5.9 presents the t-SNE embeddings of feasible and infeasible assemblies of variable sizes using latents originating from our baseline model, trained on feasible ones alone. First, in Fig. 5.9a, we see that these latents' rich semantics encode information about the assembly size. However, at least for the case of this baseline model, it can not distinguish between feasible and infeasible samples at all (Fig. 5.9b); the corresponding clusters present a mixture of both feasible and infeasible instances, without a clear decision boundary.

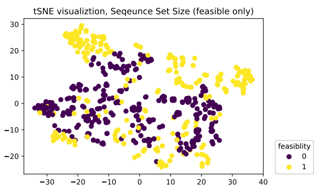


(a)

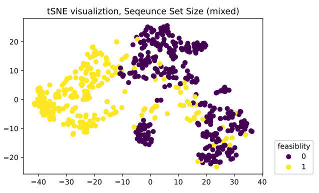


(b)

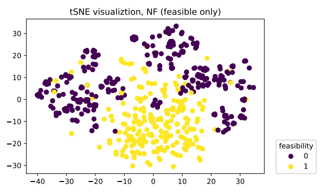
Figure 5.9.: t-SNE visualization of latents created by the baseline model, trained on feasible assemblies alone. Assembly latents are color labeled by their size (a) and ground truth feasibility class (b). Best viewed in color.



(a)



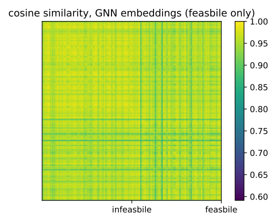
(b)



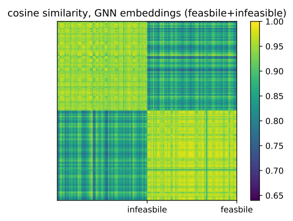
(c)

Figure 5.10.: t-SNE visualization of A_5 latents created by baseline model in a single-class setting (a), a binary-class setting (b) and the baseline latents transformed by the NF model into a Gaussian base distribution (c).

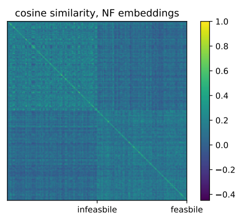
In Fig. 5.10, we look closer into the cluster of A_5 assemblies. Figure 5.10a presents the latents of the baseline model. In Fig. 5.10b, we see how the model trained using binary supervision, i.e., on both feasible and infeasible assemblies, is able to cluster infeasible



(a)



(b)



(c)

Figure 5.11.: Cosine Similarity between A_5 latents. In (a) the latents are produced by the baseline model, trained on feasible assemblies alone. In (b), by the model trained on both feasible and infeasible assemblies. Finally, in (c), latents were produced by the NF, trained on feasible assemblies only. Best viewed in color.

latents together[8]. Finally, in Fig. 5.10c, we show the baseline latents transformed by our NF model into a Gaussian base distribution. Here again, a separation between the classes is evident.

Qiu et al. (2022) described a common problem in deep one-class classification in which all the data embeddings collapse and become similar. To better understand this phenomenon, we compute the Cosine Similarity[9] between test set embeddings of both feasible and infeasible assemblies and plot the resulting tables in Fig. 5.11.

Indeed, the embeddings of the model trained only on feasible assemblies are almost identical (Fig. 5.11a). We overcome this problem by transforming these latents into another representation space with NF. The model can push the embeddings of the two classes apart (Fig. 5.11c), as infeasible assemblies embeddings are allocated with lower density. However, our method can not to retrieve the near-optimal embeddings generated by the model trained in a binary-class setting (Fig. 5.11b).

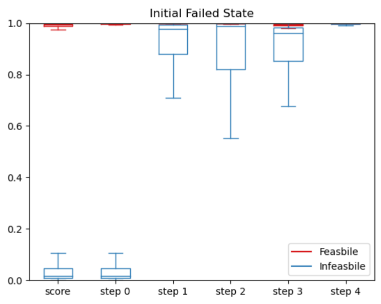
[8]In the baseline model, feasibility classification is based on the output probability of individual node embeddings and not on the pooled representation. However, we believe this analysis still provides insights into the model's working.

[9]The *Cosine Similarity* between two vectors x, y is defined as $S_C(x,y) = (x \cdot y^T)/(|x||y|)$. $S_C \in [-1,1]$ such that when $S_C = 1$ the two vectors are identical, when $S_C = 0$ they are orthogonal and finally when $S_C = -1$ they are exactly the opposite.

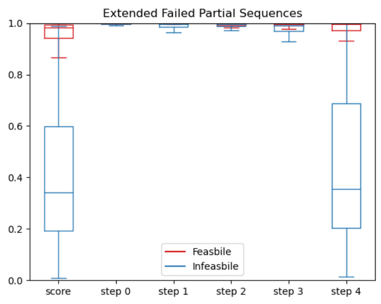
5.4.4. Ablation Studies

Infeasible assemblies trajectory

The baseline model loss term compares its output to the expert sequence trajectories (Section 4.3). We experimented with two possible approaches for assigning infeasible assemblies with a target trajectory.



(a)

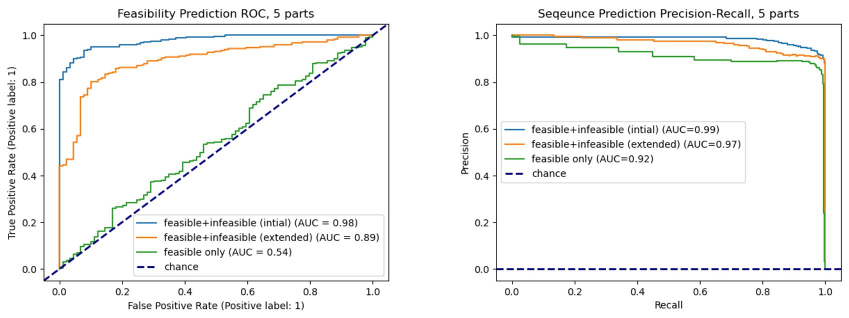


(b)

Figure 5.12.: Two approaches for incorporating infeasible assemblies in the training set, as reflected in the mean step probability and overall sequence score, an example for A_5. In (a), the assembly initial state is assigned with a dummy trajectory. In (b), failed sequences attempted by the simulation are extended with the dummy state. Approach (a) is more effective in lowering infeasible assemblies score.

In the first, the assembly's initial state is assigned with a dummy trajectory, in which no part should be chosen. In the second, for each infeasible assembly, we include all failed sequences provided by the simulation environment (Section 5.1) and extend with a dummy final state. Meaning, multiple failed trajectories are included in the training set for the same infeasible assembly.

We trained our model with the two approaches and compared its step-by-step probabilities on feasible and infeasible assemblies. When examining Fig. 5.12a, we notice that in the first approach, maximal reduction in infeasible sequences probability is achieved in the initial step, though a decline is also noticed in the following steps. On the other hand, for the second (Fig. 5.12b), the reduction is concentrated almost entirely in the last step. Indeed, there is an additional benefit for the first approach in terms of running time, as infeasible assemblies could be identified early, with no need to traverse through the remaining steps. Overall we witness a greater reduction in infeasible sequence probabilities with the first approach, and therefore better separation



(a) (b)

Figure 5.13.: Comparison of the two methods for incorporating infeasible assemblies into the model's training set to the baseline model, trained on feasible assemblies alone. Both methods achieve similar results in sequence prediction (b), therefore adding infeasible assemblies does not harm the training procedure. However, using failed initial state score higher in the classification task (a).

in the downstream classification task, as reflected in an almost perfect AUC score (Fig. 5.13a).

In a second experiment, we verified that adding infeasible assemblies to the training set does not affect the model performance in the previous sequence prediction task. We compared in this setting the models trained above with the baseline model, trained on feasible assemblies alone (Fig. 5.13b).

Sequence score

We examined several aggregation functions as candidates for the Sequence Score (Section 4.4.1). The first requirement for this score is to represent the overall confidence of the model in a predicted sequence, considering the downstream task of feasibility classification. The second is to be independent of the length of the sequence, allowing its use regardless of the number of parts in the assembly. We considered several candidate aggregation functions: *Minimum*, *Maximum*, *Mean*, *Variance* and *Product*. We trained the model on a mixture of feasible and infeasible assemblies from A_5 and A_6 and then compared the candidate function values on model output sequences.

Examining Fig. 5.14, we see that both the minimum and the product functions enable similar separation between the feasible and infeasible assemblies. However, the product value is highly coupled to the length of the sequence, as reflected in the difference in its value between A_5 and A_6 for feasible assemblies.

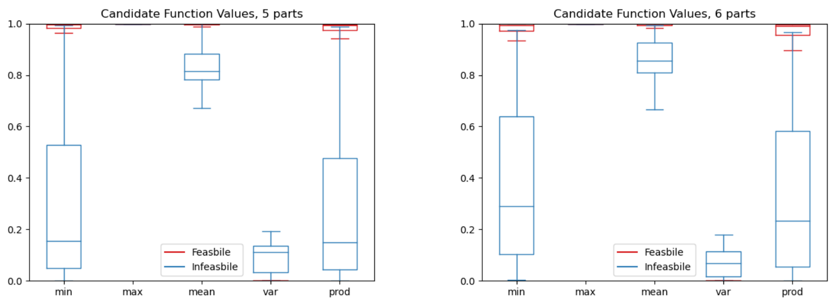


Figure 5.14.: Values of candidate aggregation functions on the probabilities of predicted sequences of both infeasible and infeasible assemblies. The minimum function was chosen as sequence score.

Normalizing Flows Likelihood

The NF log-likelihood equation (Eq. 3.15) is a sum of two terms: the density of the transformed point in the base distribution and the determinants of the Jacobian of the flows transformation matrices.

To better understand the contribution of each of these terms to the model's ability to separate between the classes, we plot their value separately for an A_5 test set in Fig. 5.15. For better visibility, we use Kernel Density Estimation (KDE) (Parzen, 1962) to plot the smoothed distributions. As we see, the determinants are the main contributing factor to class separation, whereas the values produced by the base distribution collide.

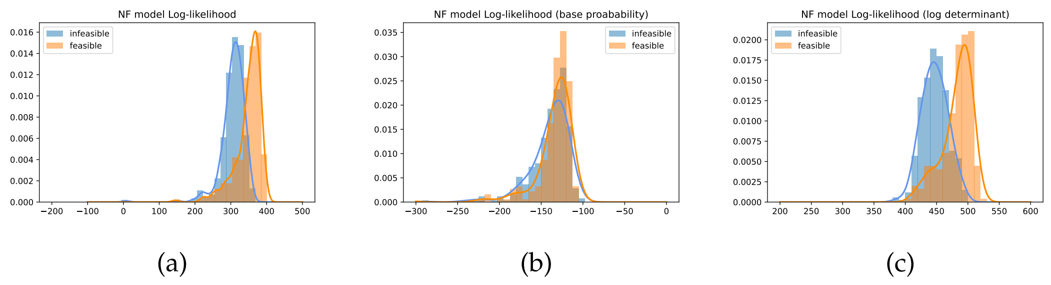


Figure 5.15.: NF predicted log-likelihood values for feasible and infeasible assemblies from A_5. The prediction of the model (a) is a sum of the base probability (b) and the transformation matrices log determinant (c).

6. Conclusion

6.1. Problem Definition

In this thesis, we addressed the Assembly Sequence Planning (ASP) problem, which is aimed at deriving an order of operations, according to which a target product can be assembled step-by-step by a robot system. We studied two sub-tasks: *Feasibility Prediction* and *Sequence Prediction*. In the first, given a geometric description of a final product, we predict if a target robot system would be able to assemble it without collision. In the second, we infer a series of feasible placement actions, leading to the complete product.

We work on a propriety dataset comprised of specifications of structures made of two types of metal profiles, connected with angle brackets. With these three distinct atomic parts, multiple structures with varying levels of assembly complexity are defined. A target robotic system simulation environment was tasked with assembling these structures, exhaustively attempting all possible part ordering. Successful sequences were used as ground truth trajectories for our method.

6.2. Our Approach

We represent each assembly as a graph with its parts surfaces as nodes, encoding their corresponding geometrical distances and orientations as graph edges. This graph also contains information about the current state of the assembly, i.e., which parts are already in place, using part nodes. This heterogeneous graph representation provides flexibility in depicting different structures and is agnostic to rotations and mirroring of the assembly.

We exploited the graph's geometrical biases and designed a GNN pipeline, creating latent representations for each assembly part. These latents are first utilized to infer a probability per part, indicating if the part in question could be placed in its target position in the current state of the assembly. To this end, since the placement is only dependent on the current state of the assembly, we trained the network in a step-by-step supervised setting, using the dataset sequences as ground-truth demonstrations. We traversed the assembly state tree with DFS, obtaining complete assembly sequences.

The part node latents are also pooled to create a single graph representation, which is plugged into a downstream feasibility predictor. We employed Normalizing Flows model for feasibility prediction, training it in a single-class setting on latents originating from feasible assemblies. The NF model predicts the probability density of a given structure under the feasible assemblies' distribution.

6.3. Contributions

Our approach has several key contributions:

1. Our Assembly Graphs are flexible, allowing us to represent different $2d$ structures while being agnostic to rotation and mirroring. Using a GNN pipeline allows us to exploit geometrical biases and maintain high memory efficiency; our GNN backbone requires $\approx 52k$ trainable parameters. Since we train our network using Imitation Learning, our setting is highly efficient in training time, requiring just 33 epochs of training on a few thousand examples. As we use a step-by-step setting, our model is independent of the number and length of predicted sequences. Finally, we require only a few seconds, the time required to traverse the assembly state-tree, to infer assembly sequences.

2. Instead of using a separate network, we exploit the graph latents produced by our GNN pipeline for feasibility prediction. Contrary to previous approaches, our NF-based model is trained in a single-class setting, modeling the distribution of feasible assemblies. It is capable of providing a density estimation for a queried assembly and not only an anomaly score. We are the first to apply NF for Out-of-Distribution detection in a graph-level AD setting.

3. We conducted extensive experiments to evaluate the capability of our model to transfer knowledge between different assembly tasks, as previous methods lacked this capacity. Our method can generalize knowledge gained on larger assemblies and then apply it to smaller ones. We believe this is the case since the relevant constraints guiding the assembly of the smaller structures are contained in larger assemblies.

7. Outlook

7.1. Limitations

Our work has several limitations:

1. We conducted our experiments on a limited dataset of assemblies, made of up to 7 parts from 3 atomic types. We assumed all of them to be 2*d* structures, i.e., lying on a flat surface. Our evaluation was executed based on simulated ground-truth trajectories, without an actual robot-in-the-loop.
2. Though our method predicts the feasibility of a given assembly structure, it does not provide explanations guiding its decision. In addition, given an input structure, our method predicts feasible assembly sequences, but it does not consider the question of sequence optimally (in terms of assembly time, etc.).
3. Our NF-based feasibility classification model beats the OC-SVM classifier, directly working on the assembly graph's latents. However, though it includes many trainable parameters, it still lags behind the binary-class baseline model. We believe this is the case since the latents generated by the GNN collapse to near identical representations.

7.2. Future Research

Based on the above, we suggest the following directions for future research:

1. **Generalization** Generalize our graph representation to 3*d* structures. This could be done by adding two more surfaces to each part and considering parallel and orthogonal relations in the 3*d* space. We suggest conducting experiments on larger assemblies with various atomic parts, also combining an actual robot in the evaluation.
2. **Optimality** Consider the optimality of the predicted assembly sequence, for instance, by including only the optimal trajectories in the ground truth or adding a prediction head to the pipeline, e.g., *assembly-elapsed-time*.

3. **Explainability** Since NF models are differentiable, we can take the gradient of the model's predicted density with respect to its input. This may allow us to provide an explanation for the model output (e.g. distance between two specific parts) and solve problems that prevent the structure assembly. Another approach could be exploring the decision boundary between feasible and infeasible assemblies' latents. Moving latents to the other side of the boundary may allow the creation of counter-factual explanations, i.e., feasible assemblies as similar as possible to the infeasible ones (Atad et al., 2022), using generative models such as Graph Auto-regressive models (You et al., 2018).

4. **Improve model performance** Few directions are suggested to improve the NF model performance. These include training the GNN and NF end-to-end, changing the NF training objective to be fully supervised (Kirichenko et al., 2020), or encouraging the GNN latents to be more expressive (Qiu et al., 2022). Another issue could be that the Gaussian base distribution, used by our NF model, lacks the sufficient capability to model the complex target distribution. This problem could be circumvented by using other complex base distributions (Stimper et al., 2022).

A. Appendix: Detailed Results

A.1. Sequence Prediction Task

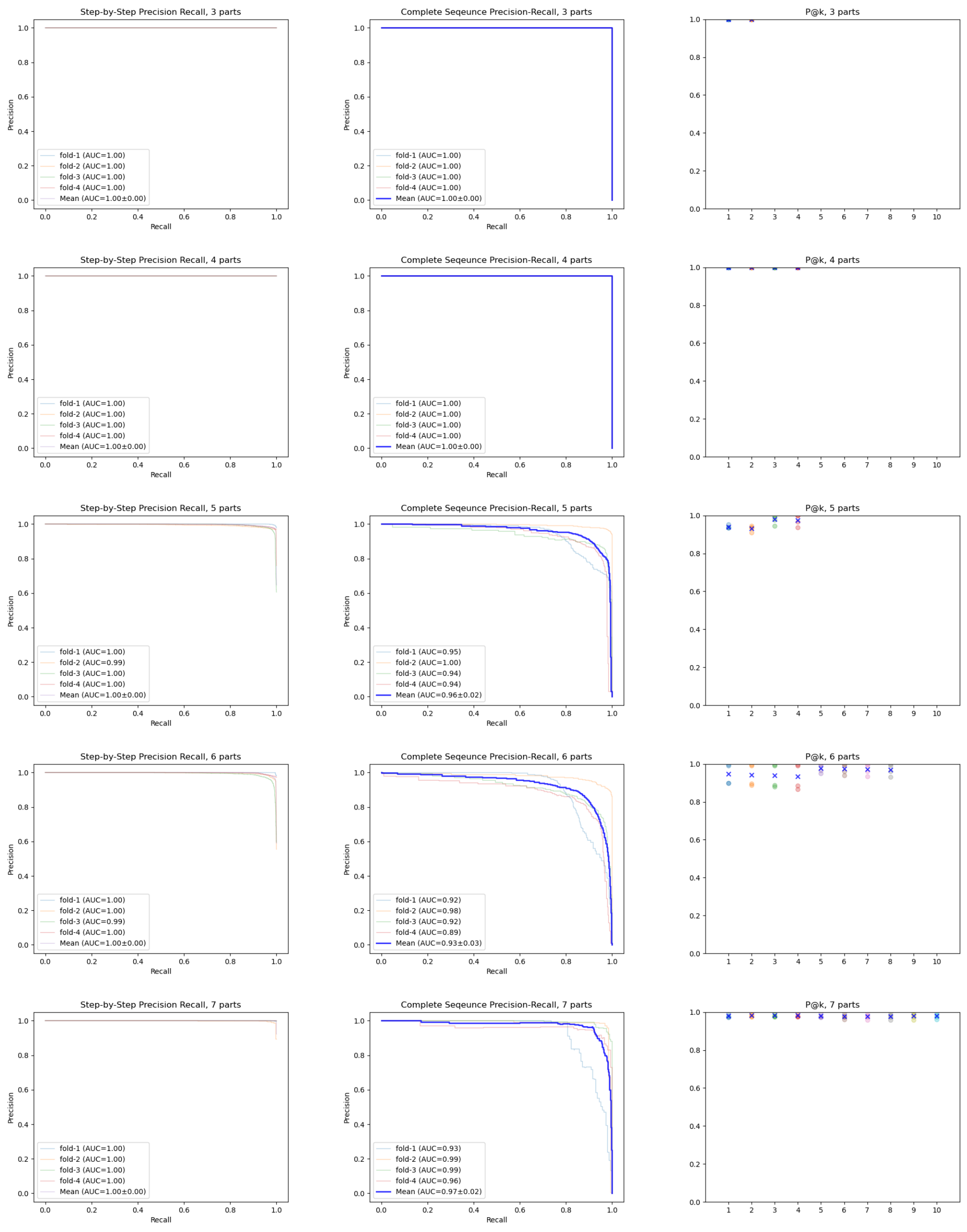


Figure A.1.: Cross-validated sequence-prediction results for the baseline model.

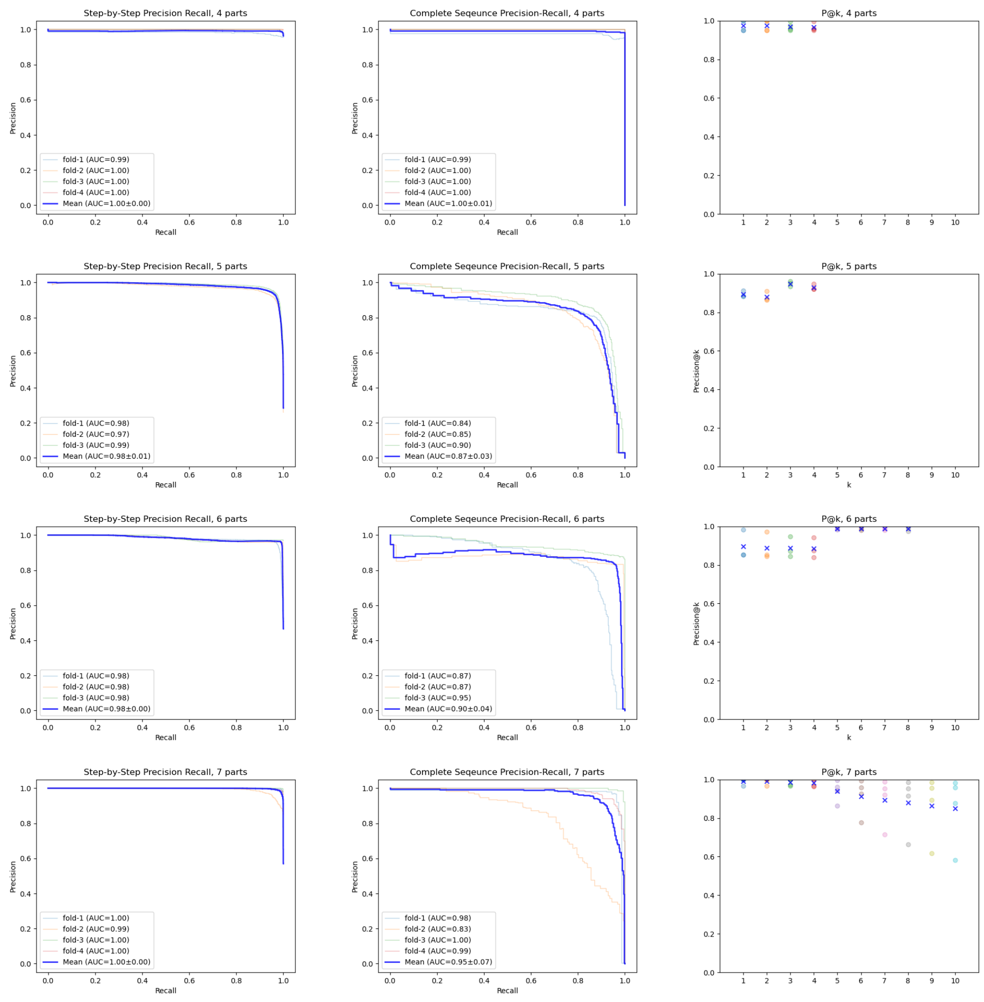


Figure A.2.: Cross-validated sequence-prediction results in the *many-to-one* setting. In each row a model was trained on all but A_i and is evaluated on A_i.

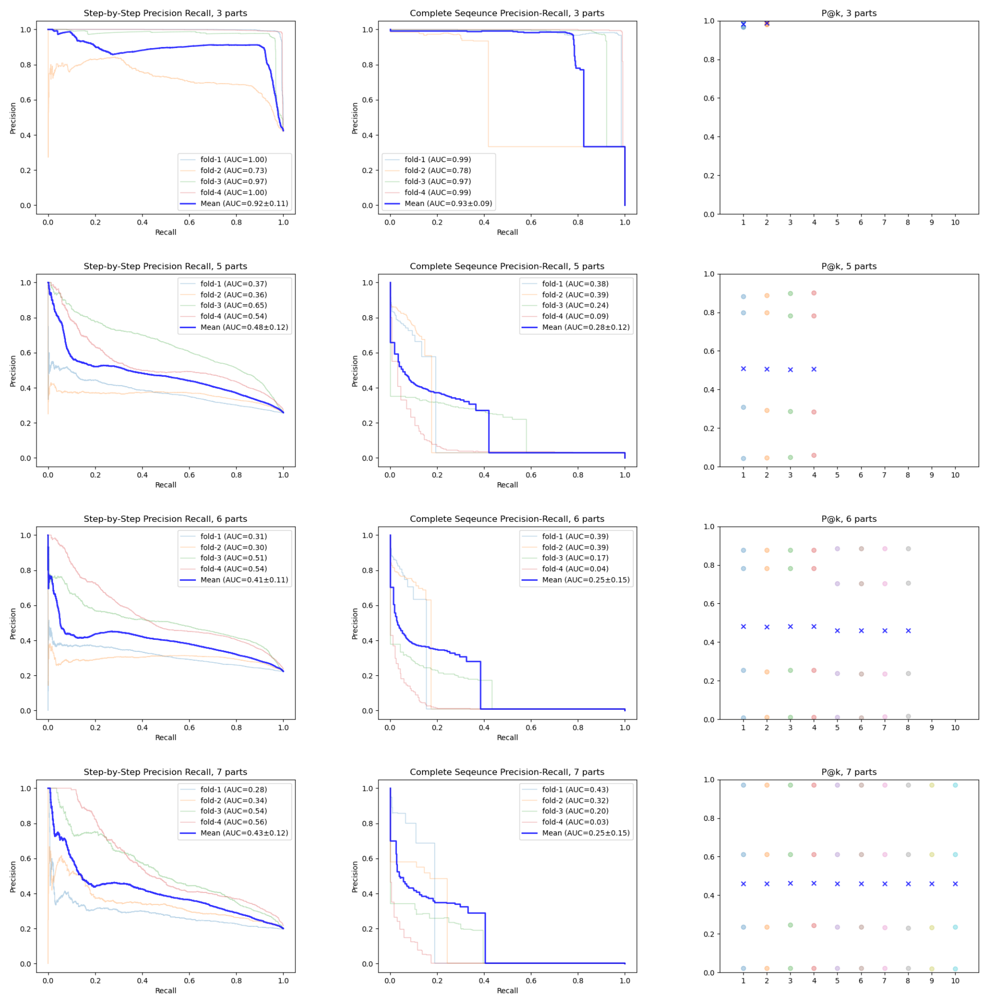


Figure A.3.: Cross-validated sequence-prediction results in the *4-to-many* setting. The model was trained on A_4 and in each row its results on another $A_{i \neq 4}$ are given.

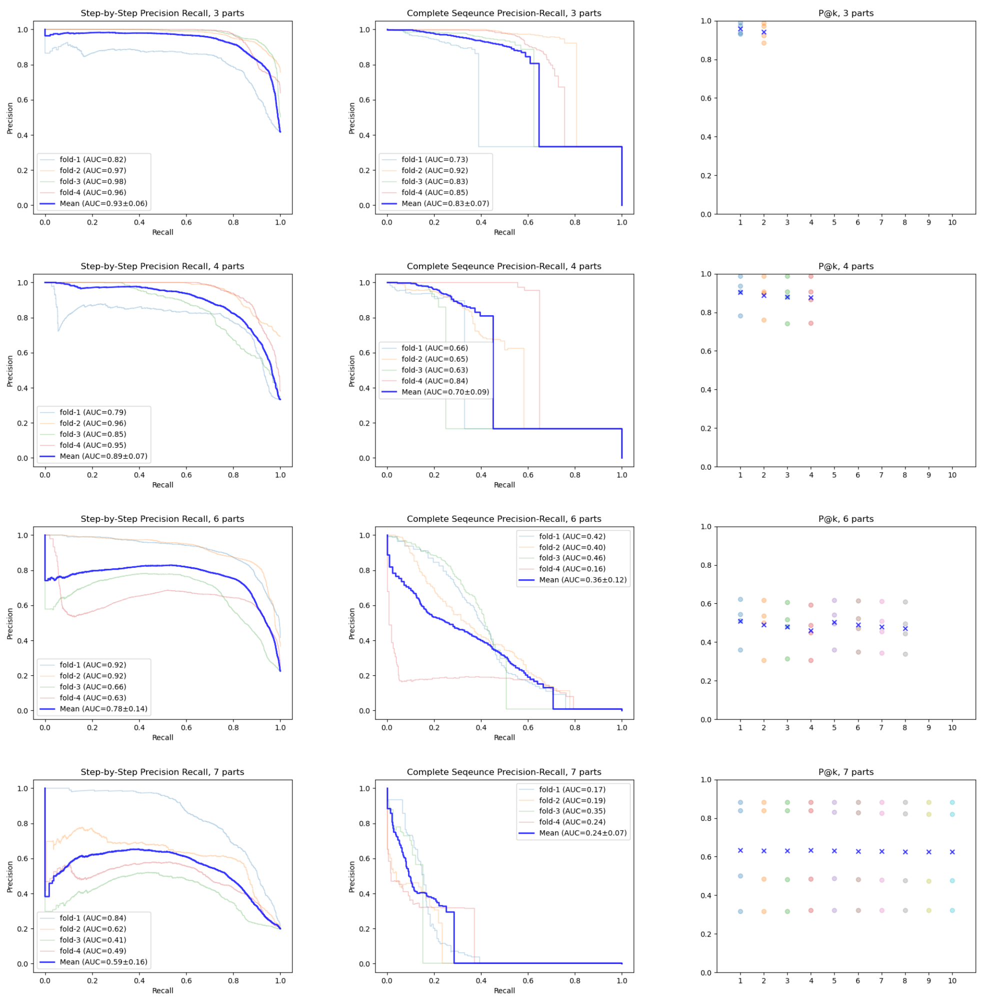


Figure A.4.: Cross-validated sequence-prediction results in the *5-to-many* setting. The model was trained on A_5 and in each row its results on another $A_{i \neq 5}$ are given.

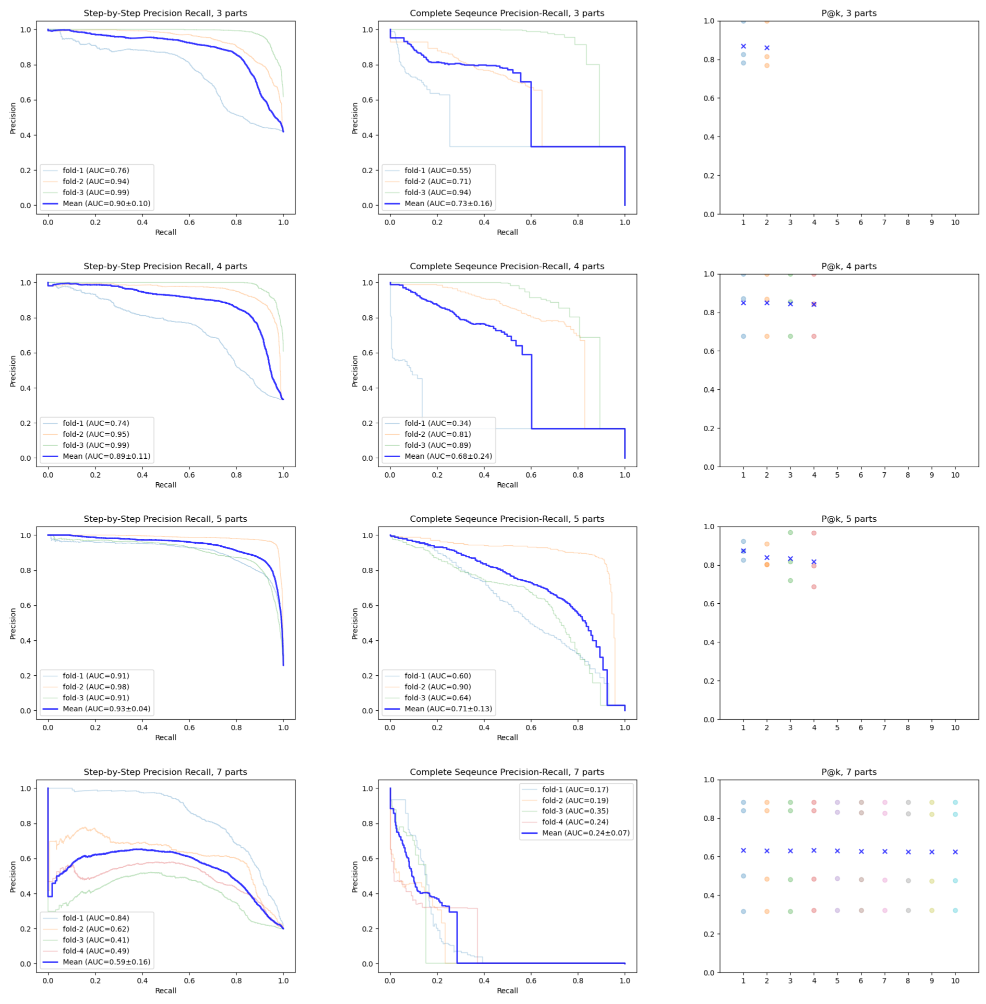


Figure A.5.: Cross-validated sequence-prediction results in the *6-to-many* setting. The model was trained on A_6 and in each row its results on another $A_{i \neq 6}$ are given.

Training/ Testing	Precision@k (↑)									
	A_3		A_4				A_5			
	$k = 1$	$k = 2$	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 1$	$k = 2$	$k = 3$	$k = 4$
A_4	0.99 ± 0.02	0.97 ± 0.04	▨	▨	▨	▨	0.24 ± 0.40	0.25 ± 0.40	0.25 ± 0.39	0.25 ± 0.39
A_5	0.97 ± 0.02	0.96 ± 0.02	0.80 ± 0.21	0.80 ± 0.22	0.79 ± 0.23	0.79 ± 0.23	▨	▨	▨	▨
A_6	0.91 ± 0.06	0.92 ± 0.05	0.90 ± 0.09	0.91 ± 0.08	0.91 ± 0.08	0.91 ± 0.08	0.86 ± 0.09	0.84 ± 0.11	0.89 ± 0.12	0.87 ± 0.13

Training/ Testing	Precision@k (↑)							
	A_6							
	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$	$k = 8$
A_4	0.27 ± 0.39	0.27 ± 0.39	0.28 ± 0.39	0.28 ± 0.39	0.28 ± 0.39	0.28 ± 0.39	0.28 ± 0.39	0.29 ± 0.39
A_5	0.59 ± 0.22	0.58 ± 0.21	0.57 ± 0.20	0.56 ± 0.19	0.60 ± 0.23	0.58 ± 0.21	0.57 ± 0.20	0.57 ± 0.20
A_6	▨	▨	▨	▨	▨	▨	▨	▨

Training/ Testing	Precision@k (↑)									
	A_7									
	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$	$k = 8$	$k = 9$	$k = 10$
A_4	0.37 ± 0.39	0.37 ± 0.39	0.37 ± 0.39	0.37 ± 0.39	0.37 ± 0.39	0.37 ± 0.39	0.37 ± 0.39	0.37 ± 0.39	0.37 ± 0.39	0.37 ± 0.39
A_5	0.70 ± 0.15	0.68 ± 0.13	0.68 ± 0.12	0.67 ± 0.12	0.66 ± 0.11	0.64 ± 0.10	0.63 ± 0.10	0.62 ± 0.10	0.62 ± 0.10	0.62 ± 0.10
A_6	0.73 ± 0.32	0.73 ± 0.32	0.72 ± 0.31	0.73 ± 0.31	0.66 ± 0.22	0.62 ± 0.16	0.61 ± 0.11	0.61 ± 0.08	0.59 ± 0.07	0.58 ± 0.07

Table A.1.: Cross-validated Sequence Prediction P@k scores in *one-to-many* setting. In each row, the model was trained on A_i and then evaluated on a different $A_{j \neq i}$ per column.

A.2. Feasibility Prediction Task

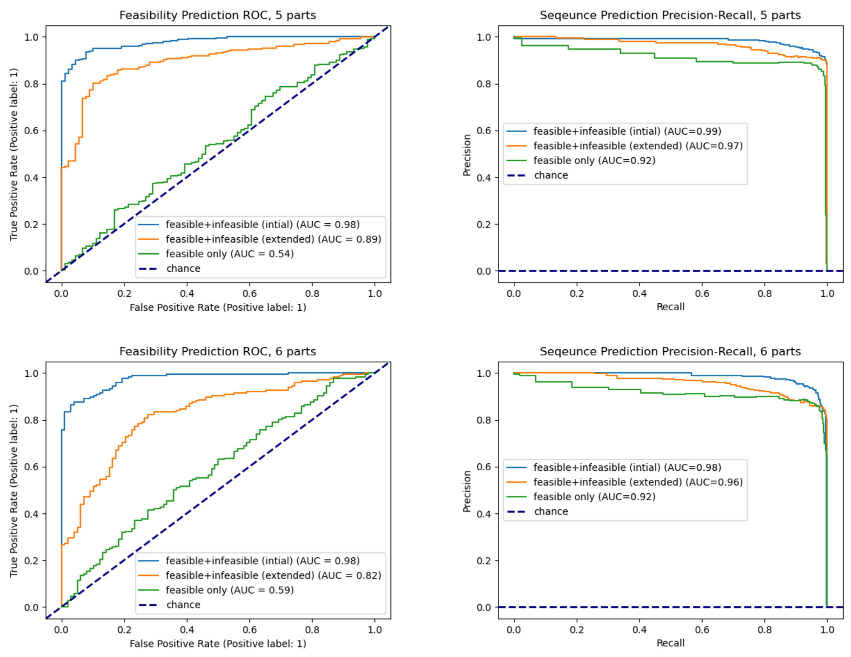


Figure A.6.: Comparison of the two methods for incorporating infeasible assemblies into the model's training set to the baseline model, trained on feasible assemblies alone. Evaluation in both feasibility and sequence prediction tasks for A_5 and A_6.

B. Appendix: One-class SVM

One-Class Support Vector Machine (OC-SVM) (Schölkopf et al., 1999) is a model that learns a decision function for novelty detection: classifying new data as similar or different to the training set.

Let $\{x_i\}_{i=1}^{n}$ be training samples in some space $\mathbb{R}^D$. Let $\Phi : \mathbb{R}^D \to F$ a feature map, mapping into a dot product space, such that the dot product can be computed by evaluating some simple kernel $k(x, y) = \Phi(x) \cdot \Phi(y)$. In our setting we used the Radial Basis Function (RBF) kernel (Williams & Rasmussen, 2006):

$$k(x,y) = \exp\left(-\frac{||x-y||^2}{2 \cdot l^2}\right) \tag{B.1}$$

Where $l > 0$ is a length scale parameter.

The underline idea behind OC-SVM is to "map the data into the feature space corresponding to the kernel, and to separate them from the origin with maximum margin" (Schölkopf et al., 1999). This is achieved with a binary function which returns $+1$ in a region which captures the training samples and -1 elsewhere (Fig. B.1). To find this decision function, we solve the quadratic program (*Primal Problem*):

$$\min_{\omega \in F,\ \xi \in \mathbb{R}^D,\ \rho \in \mathbb{R}} \frac{1}{2}||\omega||^2 + \frac{1}{\nu n}\sum_{i=1}^{n} \xi_i - \rho \tag{B.2}$$
$$\text{s.t } \forall i = 1, \ldots, n : (\omega \cdot \Phi(x_i)) \geq \rho - \xi_i$$
$$\forall i = 1, \ldots, n : \xi_i \geq 0$$

Where $n \in \mathbb{N}$ is the number of training samples, ξ_i are the slack variables from the soft-margin SVM formulation (Cortes & Vapnik, 1995) and $\nu \in (0, 1)$ is a hyper-parameter.

Using Lagrange multipliers and the kernel function for the dot-product calculation (*Kernel Trick*), we obtain the *Dual Problem*:

$$\min_{\alpha} \frac{1}{2}\sum_{i,\ j=1}^{n} \alpha_i \alpha_j k(x_i, x_j) \tag{B.3}$$

$$\text{s.t } \forall i = 1, \ldots, n : 0 \leq \alpha_i \leq \frac{1}{\nu n},\ \sum_{i=1}^{n} \alpha_i = 1$$

Once the optimal solution α^* is obtained, the offset ρ can be computed with $\rho = \omega \cdot \Phi(x_i) = \sum_{j=1}^{n} \alpha_j^* k(x_j, x_i)$, where x_i is some sample whose corresponding α_i^* is not at the upper or lower bound, i.e. $\alpha_i^* \in (0, 1/\nu n)$.

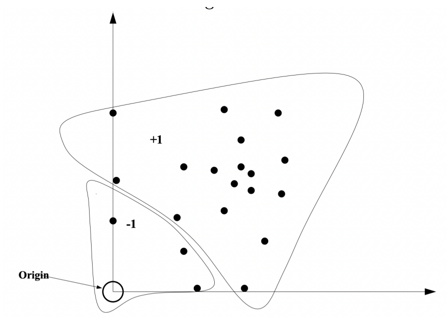


Figure B.1.: One-Class SVM classifier. The origin is the only initial member in the -1 class (Manevitz & Yousef, 2001).

Finally, the OC-SVM decision function is:

$$f(x) = sgn\left(\sum_{i=1}^{n} \alpha_i k(x_i, x) - \rho\right) \tag{B.4}$$

If $f(x) \geq 0$ then x is in the target class, otherwise, it is marked as novelty. The samples for which $\alpha_i > 0$ are called Support Vectors (SV).

In the primal equation (Eq. B.3), the parameter ν controls the trade-off between the model complexity and the training error (Xiao et al., 2014). It sets an upper-bound on the fraction of outliers (training examples regarded out-of-class) and a lower-bound on the number of training examples used as SV.

List of Figures

List of Tables

List of Acronyms

ACO Ant Colony Optimization.

AD Anomaly Detection.

ASP Assembly Sequence Planning.

AUC Area Under Curve.

CAD Computer-Aided Design.

DFS Depth First Search.

DoF Degrees of Freedom.

EA Evolutionary Algorithms.

GA Genetic Algorithms.

GAN Generative Adversarial Network.

GAT Graph Attention Network.

GCN Graph Convolution Network.

GIN Graph Isomorphism Network.

GNN Graph Neural Network.

IR Information Retrieval.

KDE Kernel Density Estimation.

MDP Markov Decision Process.

NF Normalizing Flows.

NN	Neural Networks.
OC-SVM	One-class SVM.
OoD	Out-of-Distribution.
P@k	Precision@k.
R@k	Recall@k.
RBF	Radial Basis Function.
RL	Reinforcement Learning.
ROC	Receiver Operating Characteristic.
ROC AUC	Area Under the Receiver Operating Characteristic Curve.
SV	Support Vectors.
SVDD	Deep Support Vector Data Description.
SVM	Support Vactor Machines.
t-SNE	t-distributed Stochastic Neighbor Embedding.
TAMP	Task and Motion Planning.

Bibliography

Ab Rashid, M. F. F. (2017). A hybrid ant-wolf algorithm to optimize assembly sequence planning problem. *Assembly Automation*.

Atad, M., Dmytrenko, V., Li, Y., Zhang, X., Keicher, M., Kirschke, J., Wiestler, B., Khakzar, A., & Navab, N. (2022). Chexplaining in style: Counterfactual explanations for chest x-rays using stylegan. *arXiv preprint arXiv:2207.07553*.

Bapst, V., Sanchez-Gonzalez, A., Doersch, C., Stachenfeld, K., Kohli, P., Battaglia, P., & Hamrick, J. (2019). Structured agents for physical construction. *International conference on machine learning*, 464–474.

Battaglia, P., Pascanu, R., Lai, M., Jimenez Rezende, D., et al. (2016). Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, *29*.

Bellman, R. (1957). A markovian decision process. *Journal of mathematics and mechanics*, 679–684.

Bogachev, V. I., Kolesnikov, A. V., & Medvedev, K. V. (2005). Triangular transformations of measures. *Sbornik: Mathematics*, *196*(3), 309–335.

Bouhsain, S. A., Alami, R., & Simeon, T. (2022). Learning to predict action feasibility for task and motion planning in 3d environments.

Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. (2000). Lof: Identifying density-based local outliers. *ACM sigmod record*, *29*(2), 93–104.

Brody, S., Alon, U., & Yahav, E. (2021). How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*.

Chen, W.-C., Tai, P.-H., Deng, W.-J., & Hsieh, L.-F. (2008). A three-stage integrated approach for assembly sequence planning using neural networks. *Expert Systems with Applications*, *34*(3), 1777–1786.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*(3), 273–297.

Croft, W. B., Metzler, D., & Strohman, T. (2010). *Search engines: Information retrieval in practice* (Vol. 520). Addison-Wesley Reading.

Cui, Q., Wu, S., Huang, Y., & Wang, L. (2019). A hierarchical contextual attention-based network for sequential recommendation. *Neurocomputing*, *358*, 141–149.

De Fazio, T., & Whitney, D. (1987). Simplified generation of all mechanical assembly sequences. *IEEE Journal on Robotics and Automation*, *3*(6), 640–658.

De Mello, L. H., & Sanderson, A. C. (1990). And/or graph representation of assembly plans. *IEEE Transactions on robotics and automation*, *6*(2), 188–199.

Dinh, L., Krueger, D., & Bengio, Y. (2014). Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*.

Dinh, L., Sohl-Dickstein, J., & Bengio, S. (2016). Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.

Driess, D., Oguz, O., Ha, J.-S., & Toussaint, M. (2020). Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 9563–9569.

Edelkamp, S., & Korf, R. E. (1998). The branching factor of regular search spaces. *AAAI/IAAI*, 299–304.

Eswaran, D., Faloutsos, C., Guha, S., & Mishra, N. (2018). Spotlight: Detecting anomalies in streaming graphs. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1378–1386.

Falcon, W., & The PyTorch Lightning team. (2019). *PyTorch Lightning* (Version 1.4).

Fey, M., & Lenssen, J. E. (2019). *Fast Graph Representation Learning with PyTorch Geometric*.

Funk, N., Chalvatzaki, G., Belousov, B., & Peters, J. (2022). Learn2assemble with structured representations and search for robotic architectural construction. *Conference on Robot Learning*, 1401–1411.

Garrett, C. R., Chitnis, R., Holladay, R., Kim, B., Silver, T., Kaelbling, L. P., & Lozano-Pérez, T. (2021). Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, *4*, 265–293.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural message passing for quantum chemistry. *International conference on machine learning*, 1263–1272.

Gomes-Selman, J., & Demir, N. (2019). Graph level anomaly detection.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial networks. *Communications of the ACM*, *63*(11), 139–144.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, *22*(1), 5–53.

Iwankowicz, R. R. (2016). An efficient evolutionary method of assembly sequence planning for shipbuilding industry. *Assembly Automation*.

Jiménez, P. (2013). Survey on assembly sequencing: A combinatorial and geometrical perspective. *Journal of Intelligent Manufacturing*, *24*(2), 235–250.

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Kirichenko, P., Izmailov, P., & Wilson, A. G. (2020). Why normalizing flows fail to detect out-of-distribution data. *Advances in neural information processing systems*, *33*, 20578–20589.

Kobyzev, I., Prince, S. J., & Brubaker, M. A. (2020). Normalizing flows: An introduction and review of current methods. *IEEE transactions on pattern analysis and machine intelligence*, *43*(11), 3964–3979.

Kwon, D., Kim, H., Kim, J., Suh, S. C., Kim, I., & Kim, K. J. (2019). A survey of deep learning-based network anomaly detection. *Cluster Computing*, *22*(1), 949–961.

Lagraa, S., Amrouche, K., Seba, H., et al. (2021). A simple graph embedding for anomaly detection in a stream of heterogeneous labeled graphs. *Pattern Recognition*, *112*, 107746.

Li, B., Wu, Y., Sun, H., Cheng, Z., & Liu, J. (2022). Unity 3d-based simulation data driven robotic assembly sequence planning using genetic algorithm. *2022 14th International Conference on Computer and Automation Engineering (ICCAE)*, 1–7.

Li, R., Jabri, A., Darrell, T., & Agrawal, P. (2020). Towards practical multi-object manipulation using relational reinforcement learning. *2020 ieee international conference on robotics and automation (icra)*, 4051–4058.

Lim, J., Ryu, S., Park, K., Choe, Y. J., Ham, J., & Kim, W. Y. (2019). Predicting drug–target interaction using a novel graph neural network with 3d structure-embedded graph representation. *Journal of chemical information and modeling*, *59*(9), 3981–3988.

Lin, Y., Wang, A. S., Undersander, E., & Rai, A. (2022). Efficient and interpretable robot manipulation with graph neural networks. *IEEE Robotics and Automation Letters*, *7*(2), 2740–2747.

Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). Isolation forest. *2008 Eighth IEEE International Conference on Data Mining*, 413–422.

Ma, R., Pang, G., Chen, L., & van den Hengel, A. (2022). Deep graph-level anomaly detection by glocal knowledge distillation. *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, 704–714.

Ma, X., Wu, J., Xue, S., Yang, J., Zhou, C., Sheng, Q. Z., Xiong, H., & Akoglu, L. (2021). A comprehensive survey on graph anomaly detection with deep learning. *IEEE Transactions on Knowledge and Data Engineering*.

Maas, A. L., Hannun, A. Y., Ng, A. Y., et al. (2013). Rectifier nonlinearities improve neural network acoustic models. *Proc. icml*, *30*(1), 3.

Manevitz, L. M., & Yousef, M. (2001). One-class svms for document classification. *Journal of machine Learning research*, *2*(Dec), 139–154.

Nachman, B., & Shih, D. (2020). Anomaly detection with density estimation. *Physical Review D*, *101*(7), 075042.

Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., & Jaiswal, S. (2017). Graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*.

Nguyen, H. T., Liang, P. J., & Akoglu, L. (2020). Anomaly detection in large labeled multi-graph databases. *arXiv preprint arXiv:2010.03600*.

Nguyen, S., Oguz, O. S., Hartmann, V. N., & Toussaint, M. (2020). Self-supervised learning of scene-graph representations for robotic sequential manipulation planning. *CoRL*, 2104–2119.

Noseworthy, M., Moses, C., Brand, I., Castro, S., Kaelbling, L., Lozano-Perez, T., & Roy, N. (2021). Active learning of abstract plan feasibility. *arXiv preprint arXiv:2107.00683*.

Nottensteiner, K., Bodenmueller, T., Kassecker, M., Roa, M. A., Stemmer, A., Stouraitis, T., Seidel, D., & Thomas, U. (2016). A complete automated chain for flexible assembly using recognition, planning and sensor-based execution. *Proceedings of ISR 2016: 47st International Symposium on Robotics*, 1–8.

Parzen, E. (1962). On estimation of a probability density function and mode. *The annals of mathematical statistics*, *33*(3), 1065–1076.

Qiu, C., Kloft, M., Mandt, S., & Rudolph, M. (2022). Raising the bar in graph-level anomaly detection. *arXiv preprint arXiv:2205.13845*.

Rani, B. J. B. et al. (2020). Survey on applying gan for anomaly detection. *2020 International Conference on Computer Communication and Informatics (ICCCI)*, 1–5.

Rashid, M. F. F., Hutabarat, W., & Tiwari, A. (2012). A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches. *The International Journal of Advanced Manufacturing Technology*, *59*(1), 335–349.

Rodriguez, I., Nottensteiner, K., Leidner, D., Durner, M., Stulp, F., & Albu-Schaffer, A. (2020). Pattern recognition for knowledge transfer in robotic assembly sequence planning. *IEEE Robotics and Automation Letters*, *5*(2), 3666–3673.

Rodrıguez, I., Nottensteiner, K., Leidner, D., Kaßecker, M., Stulp, F., & Albu-Schäffer, A. (2019). Iteratively refined feasibility checks in robotic assembly sequence planning. *IEEE Robotics and Automation Letters*, *4*(2), 1416–1423.

Rudolph, M., Wandt, B., & Rosenhahn, B. (2021). Same same but differnet: Semi-supervised defect detection with normalizing flows. *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, 1907–1916.

Ruff, L., Vandermeulen, R., Goernitz, N., Deecke, L., Siddiqui, S. A., Binder, A., Müller, E., & Kloft, M. (2018). Deep one-class classification. *International conference on machine learning*, 4393–4402.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. *IEEE transactions on neural networks*, *20*(1), 61–80.

Schölkopf, B., Williamson, R. C., Smola, A., Shawe-Taylor, J., & Platt, J. (1999). Support vector method for novelty detection. *Advances in neural information processing systems*, *12*.

Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., & Borgwardt, K. M. (2011). Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, *12*(9).

Shih, W. C. (2020). Global supply chains in a post-pandemic world. *Harvard Business Review*, *98*(5), 82–89.

Sinanoğlu, C., & Börklü, H. R. (2005). An assembly sequence-planning system for mechanical parts using neural network. *Assembly Automation*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929–1958.

Stimper, V., Schölkopf, B., & Hernández-Lobato, J. M. (2022). Resampling base distributions of normalizing flows. *International Conference on Artificial Intelligence and Statistics*, 4915–4936.

Suárez-Ruiz, F., Zhou, X., & Pham, Q.-C. (2018). Can robots assemble an ikea chair? *Science Robotics*, *3*(17), eaat6385.

Sun, Y., & Han, J. (2013). Mining heterogeneous information networks: A structural analysis approach. *Acm Sigkdd Explorations Newsletter*, *14*(2), 20–28.

Thomas, U., Barrenscheen, M., & Wahl, F. M. (2003). Efficient assembly sequence planning using stereographical projections of c-space obstacles. *Proceedings of the IEEE International Symposium onAssembly and Task Planning, 2003.*, 96–102.

Thomas, U., Stouraitis, T., & Roa, M. A. (2015). Flexible assembly through integrated assembly sequence planning and grasp planning. *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, 586–592.

Ulyanov, D., Vedaldi, A., & Lempitsky, V. (2016). Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*.

Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, *9*(11).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Wang, X., Ji, H., Shi, C., Wang, B., Ye, Y., Cui, P., & Yu, P. S. (2019). Heterogeneous graph attention network. *The world wide web conference*, 2022–2032.

Watanabe, K., & Inada, S. (2020). Search algorithm of the assembly sequence of products by using past learning results. *International Journal of Production Economics*, *226*, 107615.

Wellhausen, L., Ranftl, R., & Hutter, M. (2020). Safe robot navigation via multi-modal anomaly detection. *IEEE Robotics and Automation Letters*, *5*(2), 1326–1333.

Wells, A. M., Dantam, N. T., Shrivastava, A., & Kavraki, L. E. (2019). Learning feasibility for task and motion planning in tabletop environments. *IEEE robotics and automation letters*, *4*(2), 1255–1262.

Weng, L. (2018). Flow-based deep generative models. *lilianweng.github.io*.

Williams, C. K., & Rasmussen, C. E. (2006). *Gaussian processes for machine learning* (Vol. 2). MIT press Cambridge, MA.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, *32*(1), 4–24.

Xiao, Y., Wang, H., & Xu, W. (2014). Parameter selection of gaussian kernel for one-class svm. *IEEE transactions on cybernetics*, *45*(5), 941–953.

Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2018). How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Xu, L., Ren, T., Chalvatzaki, G., & Peters, J. (2022). Accelerating integrated task and motion planning with neural feasibility checking. *arXiv preprint arXiv:2203.10568*.

Yang, J., Xu, R., Qi, Z., & Shi, Y. (2021). Visual anomaly detection for images: A survey. *arXiv preprint arXiv:2109.13157*.

Yang, J., Zhou, K., Li, Y., & Liu, Z. (2021). Generalized out-of-distribution detection: A survey. *arXiv preprint arXiv:2110.11334*.

Ye, Y., Gandhi, D., Gupta, A., & Tulsiani, S. (2020). Object-centric forward modeling for model predictive control. *Conference on Robot Learning*, 100–109.

You, J., Ying, R., Ren, X., Hamilton, W., & Leskovec, J. (2018). Graphrnn: Generating realistic graphs with deep auto-regressive models. *International conference on machine learning*, 5708–5717.

Zhang, C., Song, D., Huang, C., Swami, A., & Chawla, N. V. (2019). Heterogeneous graph neural network. *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 793–803.

Zhang, K., Lucet, E., Sandretto, J. A. D., Kchir, S., & Filliat, D. (2022). Task and motion planning methods: Applications and limitations. *19th International Conference on Informatics in Control, Automation and Robotics ICINCO 2022)*, 476–483.

Zhao, L., & Akoglu, L. (2021). On using classification datasets to evaluate graph outlier detection: Peculiar observations and new insights. *Big Data*.

Zhao, M., Guo, X., Zhang, X., Fang, Y., & Ou, Y. (2019). Aspw-drl: Assembly sequence planning for workpieces via a deep reinforcement learning approach. *Assembly Automation*.

Zhu, M. (2004). Recall, precision and average precision. *Department of Statistics and Actuarial Science, University of Waterloo, Waterloo*, *2*(30), 6.

Zhu, Y., Tremblay, J., Birchfield, S., & Zhu, Y. (2021). Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 6541–6548.