Assessing the Similarity of Traffic Scenarios Using Graph Neural Networks

Murat Can Üste
Technische Universität München
Email: can.ueste@tum.de

***Abstract*—The development of automated driving functions require extensive testing to demonstrate automated vehicles have better performance than humans. As the real-world testing is expensive and time-consuming, increasing efforts are being made to develop cost-effective virtual testing environments. It is required to increase our understanding of the traffic scenarios in terms of their varieties and their dynamics, to handle increasing amounts of datasets consisting of real-world or synthesized traffic scenarios. This paper proposes a graph representation for traffic scenarios that can encode the low level information about the entities such as the motion of the vehicles and lanelet structure, as well as the higher level information such as pairwise relationships and the behavior of the entities for the duration of the scenario. The proposed graph representations are then used together with an unsupervised method for learning similarities between graph representations called Deep Divergence Graph Kernels (DDGK) to construct an embedding space with pair-wise divergence scores. The learned embeddings then can be used for tasks such as clustering, classification, or additional analysis on the scenarios, which would help us with our understanding of them in terms of their varieties and dynamics.**

I. Introduction

The development of fully automated vehicles pose number of challenges to the manufacturers, regulators, and city planners. The challenges are ranging from the development and testing of the software and hardware of the vehicles, to defining a clear legal framework addressing all possible aspects related to automated vehicles. One of the most important problems in the interest of all parties are the safety assurance of automated vehicles, which is hard to solve as billions of kilometers of testing is required to satisfy the minimum requirements [1] [2] [3]. As a cost-effective alternative to kilometer based testing of automated vehicles, scenario based testing approach is seen as a promising way to test and validate automated vehicle functions [4]. As a result of this approach, there are increasing efforts to record and create datasets of real-world traffic scenarios, which can then be used to generate scenario-based virtual testing environments [5] [6] [7] [8]. However, despite the efforts, the quantity, and variety of scenarios is still not up to the desired level due to the stochastic nature of real-world traffic. Since these recordings are taken from real-world, it is hard to find some varieties of scenarios frequently, which often time are the most interesting one in terms of testing automated driving functions. In recent works, virtual scenario generation based on specifications, or real-world datasets are seen as a way to address the lack of variety and amount of test scenarios, especially for the rare cases. Although these methods are useful, they either require prior knowledge about the specification of different scenarios that can occur in real-world traffic, or set of real-world traffic scenarios that have similar properties, to base the scenario generation on. Even though rare scenarios can be detected using methods like importance sampling or similar methods as in [9] and [10] to detect rarely occurring behavior from the given datasets, these methods don't give a meaningful classification or clustering information, which is essential for the mentioned scenario generation methods.

A. Clustering and Classification on Traffic Scenarios

The clustering and classification of the traffic scenarios are commonly handled manually, or by expert systems. Recent works have been focusing on unsupervised learning models that cluster and classify different traffic scenarios based on available datasets to automate these processes. In [11], Kruber et. al. have used Random Forest based clustering method to categorize similar traffic scenarios. The method produces a similarity matrix which can then be further processes with hierarchical clustering to reorder and prepare graphically interpretable representations. Similarly, in their following work [12], a modified Random Forest technique called extended Unsupervised Random Forest (xMURF) has been used to determine a data-adaptive similarity measure for the scenarios for clustering. Differently from their first work, they have used the clustering results for training a classification model to assign cluster labels to new scenarios. Even though these methods automate the process, they still require expert knowledge for extracting features that can describe the characteristics of a traffic scenario. In these works, feature extraction is based on the vehicles that are in the immediate surrounding of the ego vehicle. This approach can't be extended to the scenarios where there are different types of traffic participants. It also can't encode the effects of other participants that are not in the immediate vicinity of the ego vehicle, which might be important in scenarios with urban settings. As a result, this approach is not easily scalable to the increasing quantity and variety of scenario datasets.

Using expert knowledge for extracting features, and using them together with learning models have been the standard approach to solve various problems. In the past decade, however, including the feature extraction process to the overall learning model instead of hand-coding them has shown performance boosts on various learning tasks. For example, the computer

vision field have seen major performance gains in supervised and unsupervised problems by extending the models to learn the features of the images with Deep Convolutional Neural Networks [13]. The main benefit of such an approach is to process the data as a whole, and optimize the feature descriptions according to data, without performing dimensionality reduction or hand-coded feature extraction which leads to some degree of information loss. The scenario clustering or classification methods can also benefit from this approach. Unfortunately, it is hard to represent traffic scenarios in a similar convenient matrix format as in images without processing the available information, which again leads to some degree of information loss. However, in recent years, there has been works focusing on implementing a similar approach on graph structured data, which is more flexible and can describe a wide variety of problems [14] [15] [16]. Due to the complex nature of traffic scenarios, using graph structure to represent them, and leveraging the recently developed methods for end-to-end graph similarity analysis can be a promising approach to clustering and classification tasks.

B. Graph Representation and Graph Similarity

Recent works implemented a learning model, called Graph Neural Networks, which can process graph structured data. These networks are mainly used for supervised tasks such as classification and regression, and less suited for unsupervised tasks. However, for assessing the similarity between scenarios, we need a similarity metric between graphs to perform clustering or classification tasks. Even though there are algorithms like *Weisfeiler-Lehman* graph kernels [17] that can be used to test isomorphism between two graphs, there is no method for the similarity between two given graphs that will consider the node and edge structure, as well as the node and edge attributes. There have been a recent surge of interest in applying neural network models to graph structured data [18] [15] [19] [20] [21] [22], influenced by the success of deep learning methods on the domains where the structure of the data is known a priori such as natural language processing and computer vision. However, due to the flexibility of the graph structured data, a neural network model must learn the desired task at hand while learning the representation of the node, edge, attribute, and community structure itself. Initial approaches like DeepWalk [19] focused on generic representations of graph primitives such as graph's nodes and edges, while the present approaches ignore learning general graph and node representations, and focus on using the graph structure during learning to classify graph's nodes, edges, or attributes. There has also been works focused on unsupervised learning of node representations [23] [24] [19] [25], edge representations [22], or latent community structure [26] [27] [28], however, there is relatively little work that focused on learning the representations of entire graphs [29] [30].

When it comes to learning the similarity between graphs, generally there are two different approaches: end-to-end supervised graph classification or graph representation learning. In supervised graph classification approaches, the aim is to solve an end-to-end whole-graph classification problem. These approaches [31] [32] [33] [34], learn an intermediate representation of a graph as a precondition to solve the classification task. These learned representations can be used for comparing similarities of graphs, however, they are heavily biased towards maximizing performance on the classification task. Considering the lack of labeled datasets for traffic scenarios, this approach can't be implemented to solve the classification of traffic scenarios, and can't be used for assessing the similarity between traffic scenarios. On the other hand, there has been great progress in graph representation learning approaches. However, these approaches come with some limitations. The first limitation is the reliance on feature engineering using methods like graph coefficient, motif distribution, or spectral decomposition to represent the graphs [35] [36] [37]. These methods of feature engineering are limited as they can only consider known graph signals. The second limitation is the reliance on algorithmic heuristics from the isomorphism literature to encode the graphs [32] [34]. One another limitation is about the assumption of identical nodes in pairs of graphs that are being compared. Although this can be useful for calculating a similarity score, it is not so useful considering the flexibility of the graph structure when it comes to traffic scenarios.

In [38], Al-Rfou et. al. proposes *Deep Divergence Graph Kernels* for learning representations over graphs that can encode a relaxed notion of graph isomorphism. Unlike the previous work, *DDGK* does not rely on supervision, domain specific knowledge, or known alignments, and learns the node representations, graph representations, as well as attention-based alignment between graphs jointly. The learned embeddings of the graphs can be further used for graph to graph supervised or unsupervised tasks such as classification, clustering, or ranking.

In the following sections, we will first introduce our proposed graph representation for traffic scenarios in section II. Afterward, in section III, we will explain the *Deep Divergence Graph Kernels* framework [38] and its modifications we introduced to apply it to our traffic scenario representations. In section IV, we will first talk about the datasets and tools used for creating traffic scenario representations, and then we will show our experiment results on applying *DDGK* framework to our traffic scenario representations. Finally, in section V, we conclude this paper by summarizing the results and future work.

II. Graph Representation of Traffic Scenarios

In this section, we will introduce the proposed graph representation of traffic scenarios. A traffic scenario that can be used for validating the automated vehicle functions must express the dynamics of a traffic scene including the automated vehicle and other traffic participants and traffic elements such as lanelet structure of the scene, the static and dynamic obstacles in the scene, etc. There are multiple traffic scenario datasets that are based on video recordings such as [6], [5], and [8]. However, even though these datasets provide

sufficient data for defining a traffic scene, it is necessary to have a common format that would make working with these traffic scenarios easy. In this paper, we use the *CommonRoad* framework [39] for defining the traffic scenarios and related tools of *CommonRoad* framework for applying further analytic operations on the scenarios, as well as converting the video recording datasets such as [8] into *CommonRoad Scenarios*.

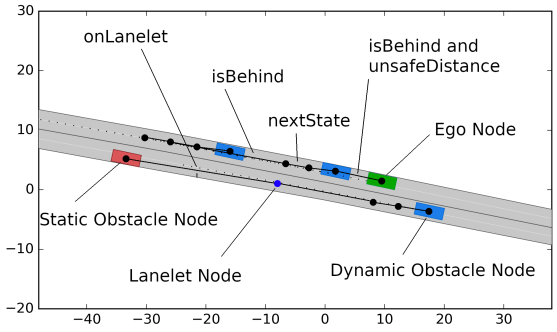

Fig. 1: Graph representation of a *CommonRoad* scenario. Note that most of the edges and nodes are not shown in order to prevent cluttering of the visualization.

A graph is defined as tuple $G = (V, E)$, where V is the set of nodes and $E \subseteq (V \times V)$ is the set of edges. Each node of a graph G can have an attribute vector Y, where the attributes of node v_i are denoted as y_i, and the attributes of an edge (v_i, v_j) are denoted as y_{ij}. In addition to the node and edge attributes, each graph G can have additional attributes defining auxiliary meta-data such as it's label where applicable. In our case, the graph would represent a traffic scenario, where nodes represent the traffic participants, and the edges represent the pairwise relations between nodes. An example traffic scenario from *CommonRoad*, and its graph representation can be seen in Figure 1.

A. Graph Nodes and Node Attributes

In our proposed graph representation of traffic scenarios, the graph nodes represent the lanelet structure and traffic participants such as ego vehicle, dynamic obstacles, as well as static obstacles. The types of nodes used are:

- *Lanelet Node*: The node type used for representing the lanelets of a traffic scenario. They include *Object Type* and *Object ID* attributes. These nodes can be used for investigating the lanelet structures in scenarios, and their relations with or effects on other participants in the scenario.
- *Static Obstacle Node*: The node type used for representing the static obstacles in the scenario. Similar to *lanelet* nodes, they include *Object Type* and *Object ID* attributes. Additionally, depending on the task hand *position* and *orientation* of the static obstacle can be added as attributes. When investigating the behavior or the similarities of dynamic traffic participants of the scenario, we have to take the static obstacles into account as well.
- *Dynamic Obstacle Node*: The node type used for representing the dynamic obstacles in the scenario based on their trajectories throughout the scenario duration. Every dynamic obstacle node contains *Object Type* and *Object ID* attributes, as well as a *Time Step* attribute where the node represents the state of the dynamic obstacle at the specified value of it. Additionally, depending on the task at hand, attributes such as *Position*, *Orientation*, *Velocity*, and *Steering Angle* can be added.
- *Ego Vehicle Node*: The node type used for representing the ego vehicle in the scenario based on its trajectory throughout the scenario duration. Its attributes are the same with *Dynamic Obstacle Node*, the only difference is its type.

The attributes of each node can vary depending on the node's type, however, they can be summarized as:

- *Object Type*: Defines the type of the node in the scenario. Can get one of the values defined above.
- *Object ID*: A unique identifier of an object in a *CommonRoad* scenario.
- *Time Step*: Time step of the node in the case of dynamic objects or ego vehicle of the scenario.
- *Additional Attributes*: Set of additional attributes that can be defined depending on the node's type.

The object types used for representing a scenario can vary depending on the task at hand, where one or many types of the types can be excluded from the graph representation. For example, if we are only interested in similarities of lanelet structures between the scenarios, we can exclude all types except the lanelet nodes. The *Object ID* is used during the edge creation process to reference the original object from the *CommonRoad* scenario and perform predicate analysis on the nodes. The *Time Step* attribute is only used for the dynamic elements of a *CommonRoad* scenario to represent an object's state at the given *Time Step*. The *Position* attributes are used during the visualization of a graph, and they can be used together with the *Additional Attributes* for the task at hand, which we will mention in section V.

These nodes are necessary for describing the similarities between scenarios in terms of the dynamic participants of a scenario. Depending on the task at hand, other dynamic elements of a scenario can be added to the scenario representation such as traffic lights, or other elements. Further details of *CommonRoad* scenarios and the definition of the scenario elements can be found in [39].

B. Graph Edges and Edge Attributes

In our proposed graph representation of traffic scenarios, the graph edges represent the pair-wise relations of the traffic participants. All edges contain a single attribute *Label*, which is a K dimensional vector $Y \in \{0, 1\}^K$ that contains the labels of the relation between pair of nodes. Similar to the node attributes, the number of relations that can be defined between nodes can vary depending on the task at hand.

In our proposed graph representation of traffic scenarios, we used the predicates *onLanelet* and *isBehind* from [40], and predicate *unsafeDistance* which indicates the *safe distance to preceding vehicle* [41] rule violation for defining the relations between dynamic elements and the lanelet structure of a scenario. Additionally, in order to define the relation between dynamic obstacle or ego vehicle trajectory nodes mentioned in II-A, the *isNextState* predicate was used. For defining the lanelet structure of a scenario, the lanelet nodes were also connected using the *isAdjacent* predicate, which is based on the lanelet adjacency. This set of predicates can be extended with new predicates or reduced depending on the task at hand. If the task requires other attributes such as numerical values, the edge labels can be updated accordingly. Below, we explain the predicates in more detail:

1) isAdjacent: In order to define the structure of the lanelet network of a *CommonRoad* scenario, the *isAdjacent* predicate was used. The edges with this label are defined only between lanelet nodes that are adjacent to each other with having one of the successor, predecessor, left of, or right of relations. Formal definition of these relations can be found in [41]. Depending on the task at hand, the lanelet relationships can be defined using *successor*, *predecessor*, *left of*, or *right of* labels, we used only *isAdjecent* label in our experiments in order to simplify the implementation.

2) isNextState: The first predicate used between dynamic obstacle and ego vehicle nodes is the *isNextState* predicate. This edge is defined between the dynamic obstacle or ego vehicle nodes with the same *Object ID* where $timestep_{v_i} < timestep_{v_j}$. These edges can describe the trajectories of the dynamic elements of a traffic scenario in a graph representation.

3) isBehind: The second predicate used between dynamic obstacle and ego vehicle nodes is the *isBehind* predicate. This label is defined on the edges between the dynamic obstacle or ego vehicle nodes that are on the same lanelet or on merging, diverging, or succeeding lanelets, with different *Object ID* but same *timestep* where $dist(v_i, v_j) < r$. The details of this predicate can be found in [40].

4) unsafeDistance: The third predicate used between dynamic obstacle and ego vehicle nodes is the *unsafeDistance* predicate. Similar to the *isBehind* predicate, this label is defined on the edges between dynamic obstacle or ego vehicle nodes with different *Object ID* but the same *timestep* where $dist(v_i, v_j) \leq d < r$. Essentially, this predicate is used between nodes that violate the *safe distance to preceding vehicle* rule defined in [41].

The vehicles following another vehicle within the same lane must maintain a safe distance to ensure collision-free trajectories even if one or several vehicles suddenly stop. In the case of another vehicle causing a safe distance violation due to a cut-in maneuver, the following vehicle must recover the safe distance within a predefined time after the start of the cut-in. This rule can be used to detect critical situations in a scenario, since violation of safe distance rule might lead to crashes.

This predicate is highly related to the *isBehind* predicate, and requires the vehicle to follow one another before an *unsafeDistance* predicate can be defined. This predicate can be used between dynamic elements of a scenario to be able to represent the critical behaviors in the scenario.

5) onLanelet: In order to define the effects of the lanelet network on traffic participants, *onLanelet* predicate was used. This label is defined on the edges between static obstacle, dynamic obstacle, or ego vehicle nodes and the lanelet nodes. The position attributes of the nodes were used to determine whether the object represented by the node is located on the specified lanelet. Details of this predicate can also be found in [40].

We now have defined the graph representations of traffic scenarios. In the following section, we will explain the *DDGK* framework that will be used in order to process this graph representations to assess the similarity between scenarios, and perform clustering or classification tasks.

III. Learning Graph Representations and Deep Divergence Graph Kernels

The aim of this paper is to learn a continuous representation $\Psi(G) \in \mathbb{R}^N$ for each graph in the given family of traffic scenario graphs $G_0, G_1, \ldots, G_N$ that can encode the node and edge structure, as well as their attributes. However, these encodings can't be useful for comparing graphs if the method of graph encoding produces one of many equally good representations each time. For example, by permuting the dimensions we can get two different but equal representations which are not comparable since they exist in two different spaces. To avoid this problem, an equivalence class across all possible encodings of a graph is needed. This type of equivalence can be defined if two encodings of a graph lead to the same pair-wise similarity scores when they are compared to other graphs in the dataset. In [38], Al-Rfou et. al. proposes to learn a graph representation by comparing it to a population of graphs. In order to compare the similarity of a pair of graphs, they suggest dividing the dataset into the *source* and *target* sets, and rely on deep neural networks to measure the *divergence* between their structure and attributes. These divergence scores than can form the *"feature vector"* of the *target* graph. In other words, the aim is to learn an embedding based kernel function $k()$ as a similarity metric for graph pairs, defined as the following:

$$k(G_a, G_b) = ||\Psi(G_a) - \Psi(G_b)||^2 \tag{1}$$

For a given set $\mathcal{S}$ of *source* graphs, and a set $\mathcal{T}$ of *target* graphs, the i^{th} dimension of the representation $\Psi(G \in \mathcal{T}) \in \mathbb{R}$ of any member of the set $\mathcal{T}$ can be defined as:

$$\Psi(G)_i = \sum_{v_j \in V_T} f_{g_i}(v_j) \tag{2}$$

where $g_i \in \mathcal{S}$, and $f_{g_i}()$ is a predictor of some structural property of the graph G, but parameterized by the graph g_i.

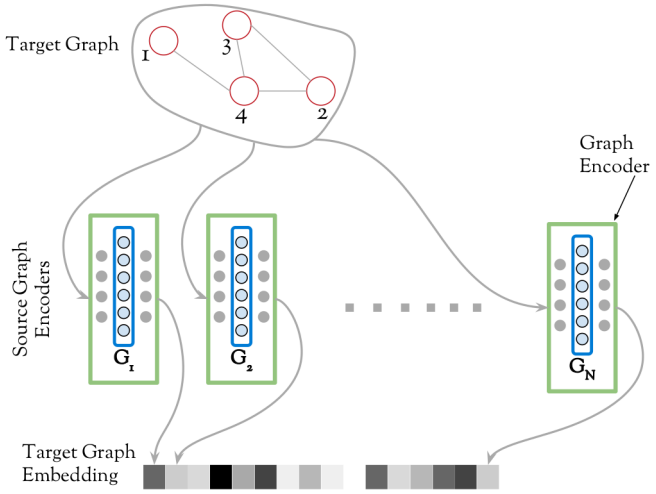

Fig. 2: Learning graph representations by measuring the divergence of a target graph across a population of source graph encoders. First, a graph encoder is trained for each graph in the source graph population. Second, for each of these encoders, the divergence score of the target graph is measured. Finally, these divergence scores are used to compare the vector representation of the target graph [38].

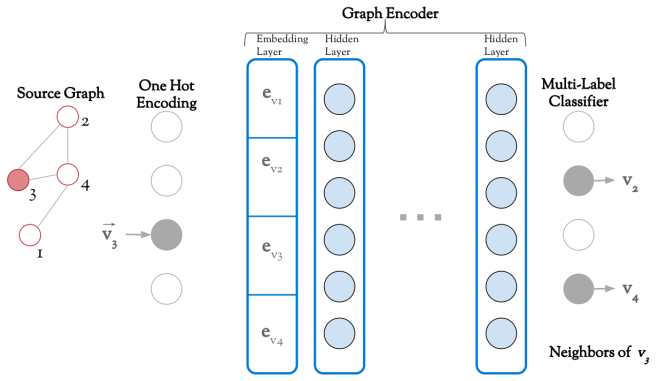

Fig. 3: A Node-To-Edges Encoder has to predict the neighbors of a vertex given in the form of a one-hot encoded vector as input. In this example, the input graph contains 4 vertices, and the encoder has to predict the neighbors of vertex v_3. First, the one-hot encoding representation $\vec{v_3}$ of vertex v_3 is multiplied by a linear embedding layer. Then, this embedding e_{v_3} is passed to a DNN which produces scores for each vertex in V. Finally, these scores are normalized using the *sigmoid* function to produce the final predictions, in this case $\{v_2, v_4\}$ [38].

It should be noted that the sets $(\mathcal{S}, \mathcal{T})$ could be disjoint, overlapping, or equal.

This is accomplished by first learning the structure of a source graph by passing it through a graph encoder. Afterwards, to measure the divergence score of a target graph, the source encoder is used to predict the structure of the target graph. If the pair of source and target graphs are similar, the source encoder should be able to predict the target graph's structure correctly, therefore having a lower divergence score, and higher in the other case. This operation can be performed for each source encoder to form the vector representation of the target graph (Figure 2).

A. Graph Encoding

To learn the structure of a graph, an encoder needs to be learned that is capable of reconstructing such structure with the given partial or distorted information. In [38], Al-Rfou et. al. proposes to use *Node-To-Edges* encoder (Figure 3) for its simplicity.

Node-To-Edges Encoder aims to predict the neighbors of a given single vertex. This can be modeled as a multilabel classification task since the predictions are not mutually exclusive, and the objective function $J(\theta)$ can be defined as:

$$J(\theta) = \sum_i \sum_j \log Pr(v_j | v_i, \theta) \tag{3}$$

The vertices in the graph v_i is represented by one-hot encoding vector $\vec{v_i}$. The embedding of the vertex can be calculated by multiplying the encoding vector $\vec{v_i}$ with a linear layer $\mathbf{E} \in \mathbb{R}^{|V| \times d}$, where $|V|$ is the number of vertices in the graph, and d is the size of the embedding space. This embedding vector represents the input feature set given to the encoder tasked with predicting the adjacency of all vertices. The encoder H, is implemented as a fully connected deep neural network with an output layer of size $|V|$ and trained as a multilabel classifier.

B. Cross-Graph Attention

We now have an encoder for encoding individual graphs. However, considering that the graphs in the dataset not necessarily will have the same size in terms of node and edge structure, we need a method that can compare different sized graphs. To address this issue, Al-Rfou et. al. propose a method of learning the alignment between graphs, that can operate in the absence of a direct mapping between nodes. The method they proposed is an attention mechanism, called *isomorphism attention*, that can align the nodes of a target graph against those of a source graph.

The *isomorphism attention* is a model that allows bi-directional mapping between the given source $\mathcal{S}$ and target $\mathcal{T}$ graph. This model consists of two separate attention networks, where the first network allows the nodes in the target graph to *attend* to the nodes in the source graph, and the second network allows neighborhood representations in the source graph to attend to neighborhoods in the target graph.

The first *attention network* is denoted as $\mathcal{M}_{\mathcal{T} \to \mathcal{S}}$, assigns every node in the target graph $(u_i \in \mathcal{T})$ a probability distribution over the nodes of the source graph $(v_j \in \mathcal{S})$ by using a *multiclass* classifier:

$$Pr(v_j | u_i) = \frac{e^{\mathcal{M}_{\mathcal{T} \to \mathcal{S}}(v_j, u_i)}}{\sum_{v_k \in V_{\mathcal{S}}} e^{\mathcal{M}_{\mathcal{T} \to \mathcal{S}}(v_k, u_i)}} \tag{4}$$

Using this attention network, the nodes of a target graph can be passed to the source graph encoder as an input regardless of their differences in node structures, since the alignment operation maps the target graph nodes to the source graph nodes.

The second network is a *reverse attention network*, denoted as $(\mathcal{M}_{\mathcal{S}\to\mathcal{T}})$, which maps the neighborhood in the source graph to the neighborhood in the target graphs by using a *multilabel* classifier:

$$Pr(u_j|\mathcal{N}(v_i)) = \frac{1}{1+e^{-\mathcal{M}_{\mathcal{S}\to\mathcal{T}}(u_j,\mathcal{N}(v_i))}} \tag{5}$$

By using this reverse attention network, the neighborhood prediction of the source graph can be aligned back to the neighborhood of the target graph, allowing us to use different sized graphs for comparison. In other words, the addition of these two networks to the source graph encoder allow us to construct a target graph encoder that can predict the neighbors of the target graph by utilizing the structure of the source graph (Figure 4).

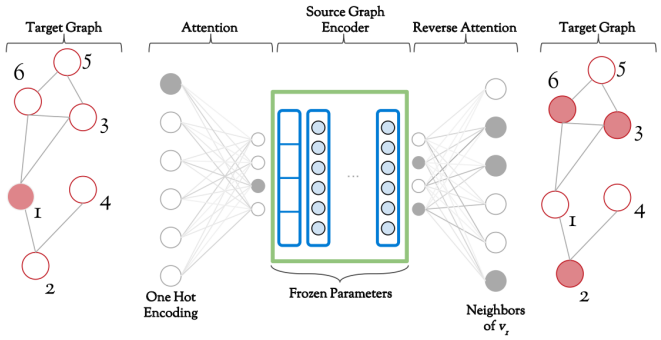

Fig. 4: Attention layers mapping the target graph nodes onto the source graph. The augmented encoder has to predict the neighbors of node 1 in the target graph. First, node 1 is passed through the attention layer which assigns it to the nodes of the source graph, which is 3 in this case. Second, the source graph encoder predicts the neighbors of node 3, which are $\{2,4\}$ in this case. Finally, the reverse attention network maps nodes $\{2,4\}$ of the source graph to nodes $\{2,3,6\}$ of the target graph which are the predicted neighbors of node 1 [38].

C. Attribute Consistency

So far we have defined the graph encoder and the attention networks for encoding and predicting the node and edge structures of the graphs. However, labeled graphs are not only defined by their structures, but also by the attributes of their nodes and edges. The attention network can be used for mapping the nodes of the target graphs to the source graph nodes, but there might be several equally good mappings with different node and edge attributes. To learn the alignment of the node and edge attributes, Al-Rfou et. al. proposes adding regularizing losses to the attention and reverse attention networks.

Given that the attention network $\mathcal{M}_{\mathcal{T}\to\mathcal{S}}$ learns the distribution $Pr(u_k|v_j)$ for the nodes v of source graph and nodes u of target graph, the probability distribution over the attributes inferred by the attention process can be defined as:

$$Q_n(y_i|u_j) = \sum_k \mathcal{M}_{\mathcal{T}\to\mathcal{S}}(y_i|v_k)Pr(v_k|u_j) \tag{6}$$

Following this, the attention regularizing loss over the node attributes can be defined as the average cross-entropy loss between the observed distribution of attributes and the inferred ones by:

$$L = \frac{1}{|V_{\mathcal{T}}|}\sum_j^{|V_{\mathcal{T}}|}\sum_i Pr(y_i|u_j)\log(Q_n(y_i|u_j)) \tag{7}$$

where $|V_{\mathcal{T}}|$ is the number of nodes in the target graph, and $y_i \in \mathcal{Y}$ where $\mathcal{Y}$ is the set of node attributes.

For preserving the edge attributes over nodes, the regularizing loss can be calculated similarly by replacing Q_n with Q_e in Equations 6 and 7, where $Q_e(z_i|u) = Pr(z_i|u)$ is the normalized attribute counts over all edges connected to the node u, and $y_i \in \mathcal{Z}$ where $\mathcal{Z}$ is the set of edge attributes.

In addition to the node and label regularization losses for the attention network, regularization losses for the reverse attention network is also introduced which is based on the neighborhood mapping of the reverse attention network. The distribution of attributes over a node's neighborhood can be calculated as the frequency of occurrence of each attribute in the neighborhood normalized by the number of attributes appearing in the neighborhood. Similarly, the distribution of edge attributes over a node's neighborhood can be calculated by normalizing their frequencies over the total number of attributes of edges connected to the neighborhood appearing at 2-hops distance from the node.

D. Deep Divergence Graph Kernels

Now that we defined the methods for graph representation learning and graph alignment, we need a graph kernel based on divergence scores that can utilize these methods. In [38], Al-Rfou et. al. proposes to use the ability of the proposed augmented encoder's prediction as a measure of graph similarity.

We aim to learn a metric that measures the divergence score between a pair of graphs $\{\mathcal{S},\mathcal{T}\}$, where we expect the divergence score to be low if the two graphs are similar, and vice versa. The divergence score given to the graph $\mathcal{T}$ can then be defined as:

$$D'(\mathcal{S}\|\mathcal{T}) = \sum_{v_i\in V_{\mathcal{T}}}\sum_j -\log Pr(v_j|v_i, H_{\mathcal{S}}) \tag{8}$$

where $H_{\mathcal{S}}$ is the encoder trained on a graph $\mathcal{S}$. However, since $H_{\mathcal{S}}$ is not a perfect predictor of the graph $\mathcal{S}$ structure, we can assume that $D'(\mathcal{S}\|\mathcal{S}) \neq 0$, and to set it to zero, we can define:

$$D(\mathcal{S}\|\mathcal{S}) = D'(\mathcal{S}\|\mathcal{T}) - D'(\mathcal{S}\|\mathcal{S}) \tag{9}$$

However, it should be noted that this definition is not symmetric as $D(\mathcal{T}\|\mathcal{S})$ might not necessarily equal to $D(\mathcal{S}\|\mathcal{T})$. If required, symmetry can be defined by $D(\mathcal{S},\mathcal{T}) = D(\mathcal{S}\|\mathcal{T}) + D(\mathcal{T}\|\mathcal{S})$.

Using these learned metrics, the target graphs can be represented as points in vector space of source graphs, where the value of the i_{th} dimension for a given target graph $\mathcal{T}_j$ is $D(\mathcal{T}_j\|\mathcal{S}_i)$. Formally, for a given set of N source graphs, the target graph representation can be defined as:

$$\Psi(G_{\mathcal{T}}) = [D(\mathcal{T}\|\mathcal{S}_0), D(\mathcal{T}\|\mathcal{S}_1), \ldots, D(\mathcal{T}\|\mathcal{S}_N)] \qquad (10)$$

E. Training

After defining all parts of the graph representation learning and the deep divergence graph kernels, we can now explain all steps of this framework. The details of the algorithm and the training steps can be referenced from [38] in detail, however in summary:

1) First train graph encoders for each graph in the set of source graphs $\mathcal{S}$
2) Construct an augmented encoder from each of the source graph encoders in accordance with the selected target graph structure for each of the target graphs in $\mathcal{T}$
   a) First, freeze the parameters of the source graph encoders.
   b) Second, add the attention and reverse attention networks for mapping between the target graph nodes to the source graph nodes, and vice versa.
   c) Third, add the regularizing losses for preserving the node and edge attributes if available
3) Train the augmented encoders on the input, which is the selected target graph, and iterate over all target graphs in the set $\mathcal{T}$.
4) Finally, after the training is done, we use the augmented encoder prediction scores to compute the divergence between the graph pair.

As stated in [38], not all $M \times N$ (where M and N is the number of graphs in the sets $\mathcal{T}$ and $\mathcal{S}$, respectively) comparisons are necessary to achieve high performance. It is shown that empirically less than 20% of source graphs are required, which significantly reduces the computational complexity (see [38] for details).

IV. Experiments

In the previous sections, we have defined the proposed graph representation, and the *DDGK* framework for learning graph representations and embedding them in a kernel space that can be used for tasks such as clustering and classification. In this section, we will show our experiment results.

During the development of the *DDGK* framework, we first reproduced the results found in [38] using the MUTAG [42] dataset. Afterwards, we used developed framework for investigating the similarities of traffic scenarios based on their lanelet structures using the *CommonRoad* scenarios, which are available on their website (IV-A). Following this experiment, in order to investigate the similarity of traffic scenarios based on their dynamic elements, we used *HighD* [8] dataset by creating scenario graphs using the dynamic obstacles and the ego vehicle together with lanelet structure in a scenario (IV-B). For both of these experiments, we investigated the usage of the *DDGK* embeddings for clustering and classification tasks.

A. Lanelet Structure Similarity

In order to investigate the similarity of traffic scenarios based on their lanelet structures, we used the *CommonRoad* framework, and the existing scenarios in their website[1]. The scenarios on *CommonRoad* website can be filtered based on their sources, and their tags such as *Critical* and *Comfort*, which we used to investigate whether the learned embeddings and the divergence scores can be used for scenario classification tasks.

First, we separated the scenarios that have *Critical* or *Comfort* tags defined from the rest of the scenarios. Scenarios with *Critical* tag are the scenarios that can be used for validating automated vehicle functions under critical conditions such as unsafe distance to the preceding vehicle, or illegal cut-in to the front of the ego vehicle. The scenarios with *Comfort* tag are the scenarios where there is no instance of critical situations throughout the scenario duration. This separation is not necessary for the clustering task, however, the learned embeddings and the divergence scores can also be used for the classification task, so we separate them before the clustering task. Afterward, 80 of the scenario graphs were selected randomly as the target graphs, with ratios 30% to 70% for *Comfort* and *Critical*, respectively. Among these 80 scenario graphs, 8 of them (10%) are selected to be source graphs which are used to train graph encoders. We used 0.01 as the learning rate, trained the graph encoders for 600 epochs. The *Node-to-Edge* losses of the selected source scenario graphs can be seen in Figure 5.

After the encoders were trained, all 80 target graphs were used for training attention and reverse attention networks together with the source encoders for calculating the divergence scores. Similar to graph encoder training, we used 0.01 learning rate and trained the models for 600 epochs. The multilabel classification losses of the target graphs can be seen in Figure 11. In this experiment, node and edge label regularizations were not used since there is only one type for the nodes and the edges.

In Figure 12, the correlation between the dimensions of the embedding vectors of randomly selected target graphs are illustrated. The dimensions of the subplots indicate the source graph pairs from the combinations of all source graphs used for encoding. On the figures, we can see the distribution of all target graphs based on their scores with the corresponding source encoders. As it can be seen, the distributions of the target graph scores according to the source encoders have different patterns, which helps with tasks such as clustering

[1]The CommonRoad Website can be used for viewing, filtering and downloading traffic scenarios: https://commonroad.in.tum.de/

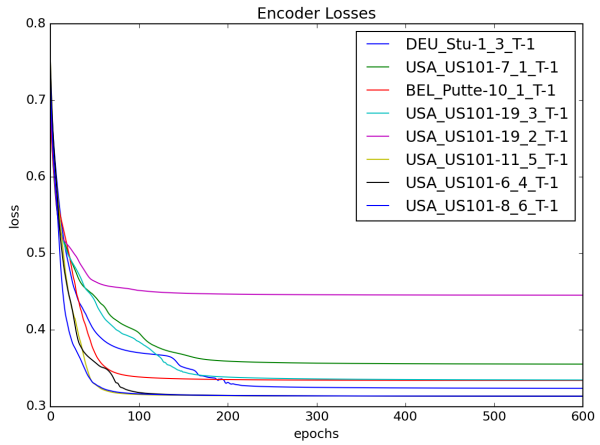

Fig. 5: The scenario graphs selected as source graphs were used for training the graph encoders for the lanelet structure similarity analysis. Their multilabel classification losses after 600 epochs with 0.01 learning rate is illustrated above.

and classification. Additionally, on the figures, the *Comfort* scenarios are represented by green samples, and the *Critical* scenarios are represented by red samples.

1) Hierarchical Clustering: In order to assess the similarity of traffic scenarios based on their lanelet structures, we have used the divergence scores of the target graphs in hierarchical clustering. The results of the clustering algorithm can be seen in Figure 6. In order to better understand the similarities between clusters, some of the scenarios were selected and visualized in Figure 7 together with their graph representations.

As it can be seen from the Figure 7, the clustering algorithm was able to group scenarios into 2 main groups based on their lanelet structures. Cluster group 1 contains scenario graphs with lower number of nodes and edges. Even though there are outliers like *ARG_Carcarana-12_1-T-1* in group 1, specifically under its leaf numbered as group 4, they usually have their edges uniformly distributed across their nodes, unlike the scenario graphs in group 8.

Even though only node and edge structures were used for similarity analysis of scenarios, the hierarchical clustering algorithm was able to group similar scenarios together. This application can be extended by labeling different lanelet nodes based on their types such as *exit ramp*, or by adding edge attributes indicating whether the lanelets intersect, in order to perform advanced analysis on similarities.

2) Classification: After performing the clustering on traffic scenario graphs, we have also performed classification based on their graph labels *Comfort* and *Critical* using the learned embeddings. We have performed grid search using the scikit-learn SVM [43] as our classifier. 10-fold cross validation was applied over the selected target graphs. The kernel choices were linear, rbf, poly, sigmoid and the regularization coefficient C was varying between 10 and 10^9. The classifier was able to predict the graph label correctly based on its graph structure 95% of the time. As it was mentioned in the clustering task, the graph representations can be extended in order to perform different classification tasks.

B. Combined Lanelet Structure and Vehicle Behavior

After our analysis on lanelet structure similarity, we used the *HighD Dataset* [8], which consist of 147 driven hours of real traffic recordings over 6 location in Germany, in order generate highway scenarios, and use them for dynamic obstacle and ego vehicle behavior analysis. Since these recordings can be converted to *CommonRoad Scenarios* using the *CommonRoad Dataset Converter* tool [2]. The scenarios generated from the *HighD Dataset* can be used for investigating the interactions between vehicles in highway situations. Furthermore, the data provided with the *HighD* dataset can be used for labeling the individual scenarios since it contains information such as *distance headway*, *time headway*, *time to collision*, and *lane change*.

In order to be able to generate scenarios from the *HighD* recordings, we first preprocessed the available data for irregularities and measurement errors. Afterwards, we selected vehicles with *minimum time headway* metric lower than or equal to 1.75. The instance where the vehicle has *minimum time headway* was configured to be the middle time step of the scenario that is going to be generated. As lower *time headway* indicates critical situations since it violates the *safe distance to preceding vehicle* rule, these scenarios were assigned the *Critical* label. The vehicles with higher *time headway* metric were selected in order to generate *Comfort* scenarios, where the middle time step of the scenario corresponds to the instance where vehicles *minimum time headway* was minimum.

After generating the scenarios from the recordings, their graph representations were generated using all available object types as well predicate attributes. 80 of the generated scenario graphs were selected randomly as the target graphs, with the ratios 30% to 70% for *Comfort* and *Critical*, respectively. Among these 80 scenario graphs, 8 of them (10%) are selected to be source graphs which are then used to train graph encoders. Similar to the previous experiment, we used 0.01 as the learning rate, and trained the graph encoders for 600 epochs. The *Node-to-Edge* losses of the selected source scenario graphs can be seen in Figure 8.

After the encoders were trained, all 80 target graphs were used for training attention and reverse attention networks together with the source encoders for calculating the divergence scores. Similar to graph encoder training, we used 0.01 learning rate and trained the models for 600 epochs. The multilabel classification losses of the target graphs can be seen on Figure 13.

In Figure 14, the correlation between the dimensions of the embedding vectors of randomly selected target graphs are illustrated. The dimensions of the subplots indicate the source graph pairs from the combinations of all source graphs used for encoding. On the figures, we can see the distribution of

[2]The *CommonRoad Dataset Converter* tool: https://gitlab.lrz.de/tum-cps/dataset-converters

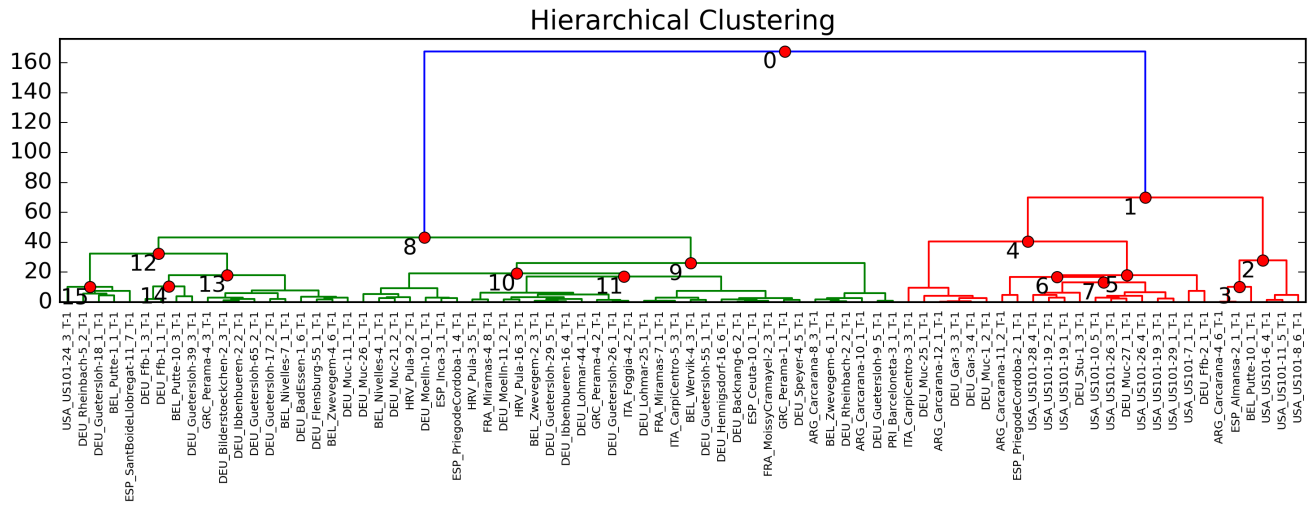

Fig. 6: The DDGK embeddings of the lanelet structure similarity analysis were clustered using hierarchical clustering, and leaves with highest number of samples are enumerated. The details of the clustering results with examples are illustrated in Figure 7.

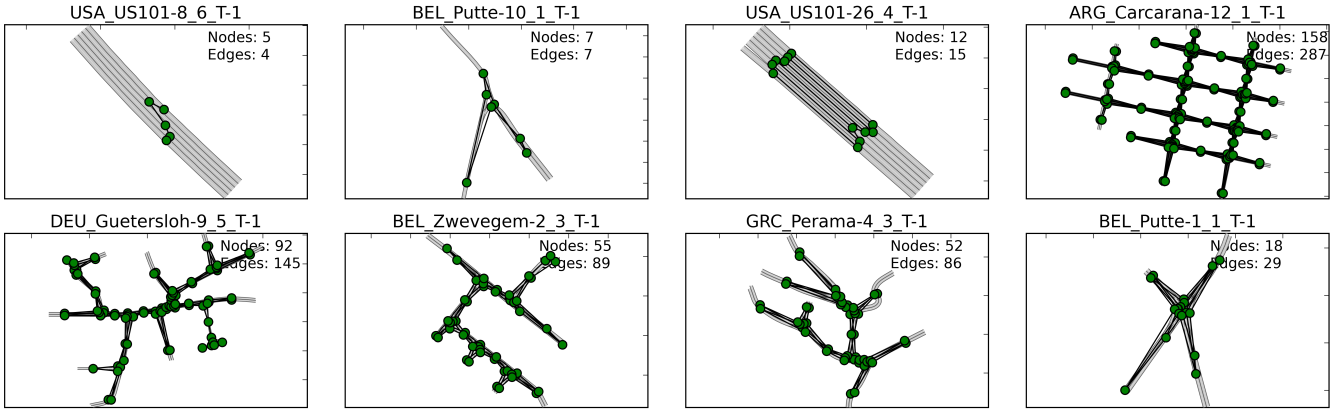

Fig. 7: The visualization of the scenarios together with their graph representations for the lanelet structure analysis. The scenarios on the first row is from the cluster group 1, and the scenarios on the second row is from the group 8 of the hierarchical clustering illustrated in Figure 6. The two scenarios on the left of first row belong to group 2, and the two scenarios belong to group 4. On the second row, the two scenarios on the left belong to group 9, and the two scenarios on the right belong to group 12. As it can be seen from the figures, group 1 mostly contains scenario graphs with lower number nodes and edges. Although there are outliers like *ARG_Carcarana-12_1-T-1* in group 1, specifically under its leaf numbered as group 4, they usually have their edges uniformly distributed across their nodes, unlike the scenario graphs in group 8. The scenarios under group 8 have a ”spiky” structure compared to the scenarios under group 1, even if their number of nodes and edges is low (although not as low as the group 9).

all target graphs based on their scores with the corresponding source encoders. As it can be seen, the distributions of the target graph scores according to the source encoders have different patterns, however they are not as obvious as the case with the previous experiment. Additionally, as it was done on the previous experiment, on the figures, the *Comfort* scenarios are represented by green samples, and the *Critical* scenarios are represented by red samples.

1) Hierarchical Clustering: In order to assess the similarity of traffic scenarios based on their lanelet structures combined with vehicle behavior, we have used the divergence scores of the target graphs in hierarchical clustering similar to the previous experiment. The results of the clustering algorithm can be seen in Figure 9. To better understand the similarities between clusters, some of the scenarios were selected and visualized in Figure 10 together with their graph representations.

As it can be seen from the Figure 10, the clustering algo-

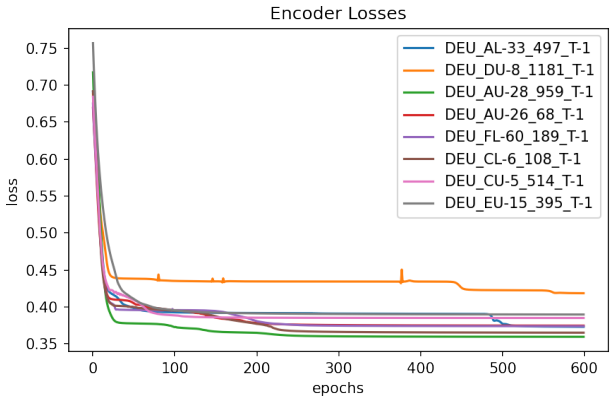

Fig. 8: The scenario graphs selected as source graphs were used for training the graph encoders for the combined lanelet structure and vehicle behavior analysis. Their multilabel classification losses after 600 epochs with 0.01 learning rate is illustrated above.

rithm was able to group scenarios into 2 main groups based on their vehicle behaviors. Cluster group 1 contains scenario graphs with lower number of nodes and edges. Cluster group 3, on the other hand, contains scenario graphs with higher number of nodes and edges. Additionally, group 1 contains edges labeled as *isBehind* and *unsafeDistance* predicates. Even though group 3 also has *isBehind* edges, the *unsafeDistance* edges are not found.

The hierarchical clustering algorithm was able to group similar scenarios, however the results don't show obvious patterns when it comes to vehicle behaviors. In order to improve the performance, additional attributes for nodes such as *position*, *orientation*, and *velocity* might be needed. However, including these attributes would require additional modification on the DDGK model as it is currently implemented for label based classification tasks.

2) Classification: Similar to the previous experiment, after performing the clustering on traffic scenario graphs, we have also performed classification based on their graph labels *Comfort* and *Critical* using the learned embeddings. Same as before, we have performed grid search using the scikit-learn SVM [43] as our classifier. 10-fold cross validation was applied over the selected target graphs. The kernel choices were linear, rbf, poly, sigmoid and the regularization coefficient C was varying between 10 and 10^9. The classifier was able to predict the graph label correctly based on its graph structure 75% of the time. By extending and optimizing the graph representations as well as the DDGK model, this result might be improved.

V. Conclusion

Throughout this paper, we first reviewed the literature on graph similarity analysis, introduced our graph representations for traffic scenarios, explained and implemented *DDGK* framework for using it for similarity analysis of traffic scenarios, and showed our experiment results.

Our proposed graph representation is flexible, and extensible, however it requires fine-tuning of the attributes based on the task at hand.

DDGK framework was successfully cluster and classify traffic scenarios based on their lanelet structure, however, the similarities and patterns were not obvious in the case of vehicle behavior analysis. Although the similarities were hard to interpret, the results show that with further optimization on the graph representation as well as the *DDGK* framework, promising results can be achieved.

References

[1] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a Formal Model of Safe and Scalable Self-driving Cars," *arXiv:1708.06374 [cs, stat]*, Oct. 2018.

[2] D. Zhao and H. Peng, "From the Lab to the Street: Solving the Challenge of Accelerating Automated Vehicle Testing," *arXiv:1707.04792 [cs]*, Jul. 2017.

[3] M. Maurer, J. Christian Gerdes, B. Lenz, and H. Winner, *Autonomous Driving: Technical, Legal and Social Aspects*. Springer Nature, 2016.

[4] "Home - pegasus-EN," https://www.pegasusprojekt.de/en/.

[5] J. Halkias and J. Colyar, "NGSIM interstate 80 freeway dataset," *US Federal Highway Administration, FHWA-HRT-06-137, Washington, DC, USA*, 2006.

[6] J. Colyar and J. Halkias, "US highway 101 dataset," *Federal Highway Administration (FHWA), Tech. Rep. FHWA-HRT-07-030*, pp. 27–69, 2007.

[7] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3213–3223.

[8] R. Krajewski, J. Bock, L. Kloeker, and L. Eckstein, "The highD Dataset: A Drone Dataset of Naturalistic Vehicle Trajectories on German Highways for Validation of Highly Automated Driving Systems," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov. 2018, pp. 2118–2125.

[9] D. Zhao, H. Lam, H. Peng, S. Bao, D. J. LeBlanc, K. Nobukawa, and C. S. Pan, "Accelerated Evaluation of Automated Vehicles Safety in Lane-Change Scenarios Based on Importance Sampling Techniques," *Trans. Intell. Transport. Sys.*, vol. 18, no. 3, pp. 595–607, Mar. 2017.

[10] M. O'Kelly, A. Sinha, H. Namkoong, J. Duchi, and R. Tedrake, "Scalable end-to-end autonomous vehicle testing via rare-event simulation," *arXiv preprint arXiv:1811.00145*, 2018.

[11] F. Kruber, J. Wurst, and M. Botsch, "An Unsupervised Random Forest Clustering Technique for Automatic Traffic Scenario Categorization," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. Maui, HI: IEEE, Nov. 2018, pp. 2811–2818.

[12] F. Kruber, J. Wurst, E. S. Morales, S. Chakraborty, and M. Botsch, "Unsupervised and Supervised Learning with the Random Forest Algorithm for Traffic Scenario Clustering and Classification," in *2019 IEEE Intelligent Vehicles Symposium (IV)*. Paris, France: IEEE, Jun. 2019, pp. 2463–2470.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[14] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[15] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," *arXiv:1609.02907 [cs, stat]*, Feb. 2017.

[16] V. Garcia and J. Bruna, "Few-shot learning with graph neural networks," *arXiv preprint arXiv:1711.04043*, 2017.

[17] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels." *Journal of Machine Learning Research*, vol. 12, no. 9, 2011.

[18] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," *arXiv:1706.02216 [cs, stat]*, Sep. 2018.

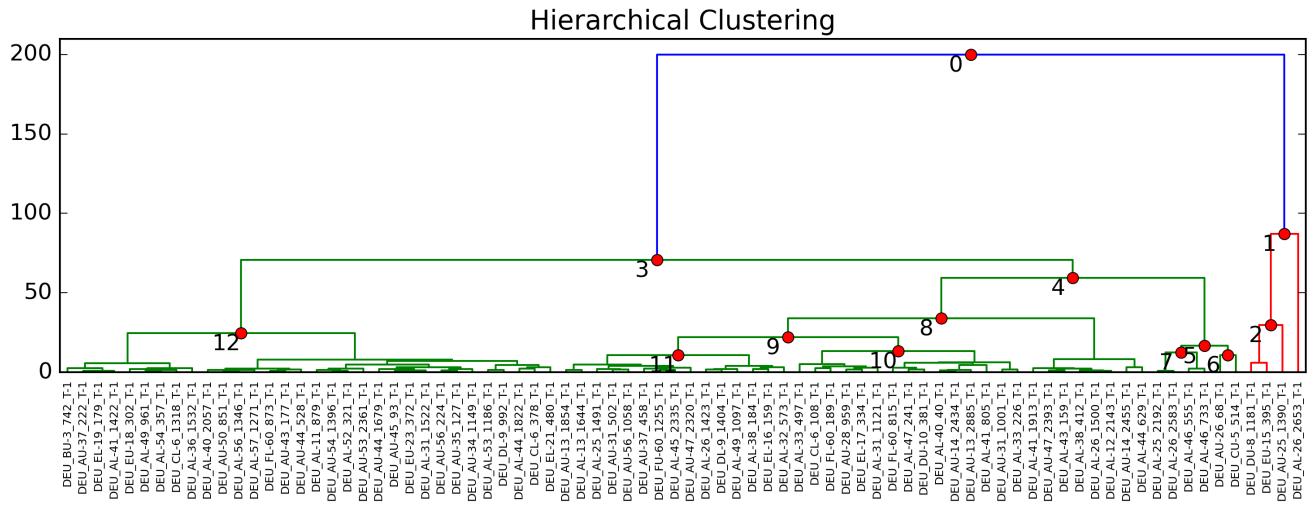

Fig. 9: The DDGK embeddings of the lanelet structure combined with vehicle behavior similarity analysis were clustered using hierarchical clustering, and leaves with highest number of samples are enumerated. The details of the clustering results with examples are illustrated in Figure 10.

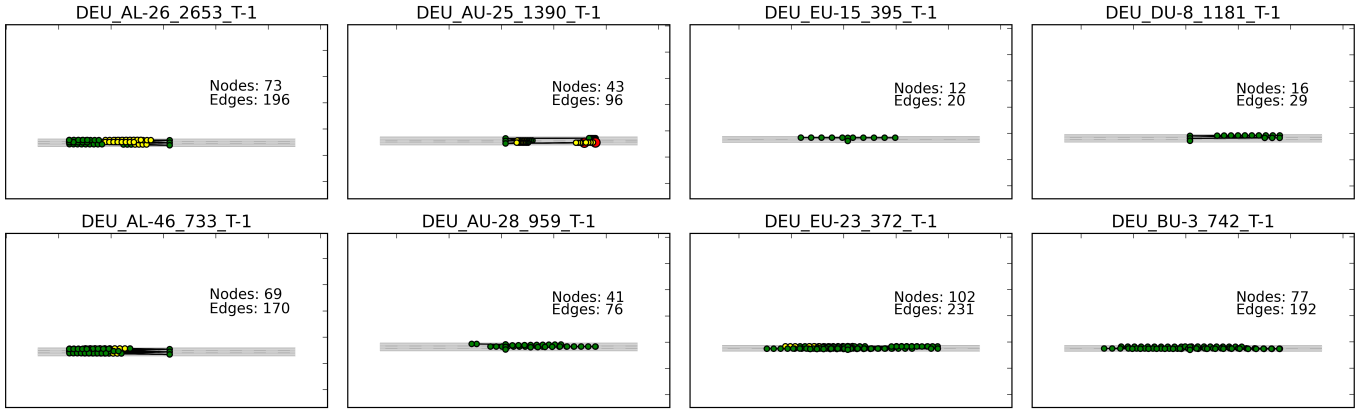

Fig. 10: The visualization of the scenarios together with their graph representations for the lanelet structure combined with vehicle behavior similarity analysis. The scenarios on the first row is from the cluster group 1, and the scenarios on the second row is from the group 3 of the hierarchical clustering illustrated in Figure 9. Additionally, nodes with *isBehind* edges are indicated by yellow color, and the nodes with *unsafeDistance* edges are indicated by red color. The scenarios on first row shows all scenarios of group 1 from right to left. On the second row, the two scenarios on the left belong to group 4, and the two scenarios on the right belong to group 12. As it can be seen from the figures, group 1 contains scenario graphs with lower number nodes and edges. Additionally, they contain *isBehind* or *unsafeDistance* edges. In cluster group 3, the number of nodes and edges is higher than group 1, however, they contain less *isBehind* edges, and no *unsafeDistance* edges at all.

[19] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014, pp. 701–710.

[20] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," *arXiv:1710.10903 [cs, stat]*, Feb. 2018.

[21] D. Zhu, P. Cui, D. Wang, and W. Zhu, "Deep variational network embedding in wasserstein space," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2827–2836.

[22] S. Abu-El-Haija, B. Perozzi, and R. Al-Rfou, "Learning edge representations via low-rank asymmetric projections," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 1787–1796.

[23] S. Cao, W. Lu, and Q. Xu, "Deep Neural Networks for Learning Graph Representations," *AAAI*, vol. 30, no. 1, Feb. 2016.

[24] A. Grover and J. Leskovec, "Node2vec: Scalable Feature Learning for Networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, Aug. 2016, pp. 855–864.

[25] A. Tsitsulin, D. Mottin, P. Karras, and E. Müller, "VERSE: Versatile Graph Embeddings from Similarity Measures," in *Proceedings of the 2018 World Wide Web Conference*, ser. WWW '18. Republic and Canton of Geneva, CHE: International World Wide Web Conferences

Steering Committee, Apr. 2018, pp. 539–548.
[26] S. Cavallari, V. W. Zheng, H. Cai, K. C.-C. Chang, and E. Cambria, "Learning Community Embedding with Community Detection and Node Embedding on Graphs," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, ser. CIKM '17. New York, NY, USA: Association for Computing Machinery, Nov. 2017, pp. 377–386.
[27] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community Preserving Network Embedding," *AAAI*, vol. 31, no. 1, Feb. 2017.
[28] V. W. Zheng, S. Cavallari, H. Cai, K. C.-C. Chang, and E. Cambria, "From Node Embedding To Community Embedding," *arXiv:1610.09950 [cs]*, Sep. 2017.
[29] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, and R. Faulkner, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.
[30] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural Message Passing for Quantum Chemistry," in *International Conference on Machine Learning*. PMLR, Jul. 2017, pp. 1263–1272.
[31] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks," *AAAI*, vol. 33, no. 01, pp. 4602–4609, Jul. 2019.
[32] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning Convolutional Neural Networks for Graphs," in *International Conference on Machine Learning*. PMLR, Jun. 2016, pp. 2014–2023.
[33] A. J.-P. Tixier, G. Nikolentzos, P. Meladianos, and M. Vazirgiannis, "Graph classification with 2d convolutional neural networks," in *International Conference on Artificial Neural Networks*. Springer, 2019, pp. 578–593.
[34] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An End-to-End Deep Learning Architecture for Graph Classification," *AAAI*, vol. 32, no. 1, Apr. 2018.
[35] M. Berlingerio, D. Koutra, T. Eliassi-Rad, and C. Faloutsos, "NetSimile: A Scalable Approach to Size-Independent Network Similarity," *arXiv:1209.2684 [physics, stat]*, Sep. 2012.
[36] A. Tsitsulin, D. Mottin, P. Karras, A. Bronstein, and E. Müller, "Netlsd: Hearing the shape of a graph," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2347–2356.
[37] P. Yanardag and S. Vishwanathan, "Deep Graph Kernels," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '15. New York, NY, USA: Association for Computing Machinery, Aug. 2015, pp. 1365–1374.
[38] R. Al-Rfou, D. Zelle, and B. Perozzi, "DDGK: Learning Graph Representations for Deep Divergence Graph Kernels," *The World Wide Web Conference on - WWW '19*, pp. 37–48, 2019.
[39] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2017, pp. 719–726.
[40] M. Klischat and M. Althoff, "Synthesizing Traffic Scenarios from Formal Specifications for Testing Automated Vehicles," p. 8.
[41] S. Maierhofer, A.-K. Rettinger, E. C. Mayer, and M. Althoff, "Formalization of Interstate Traffic Rules in Temporal Logic," p. 8.
[42] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity," *Journal of medicinal chemistry*, vol. 34, no. 2, pp. 786–797, 1991.
[43] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, and J. Vanderplas, ""Scikit-learn: Machine learning in Python." Journal of machine learning research 12 (Oct), 2825–2830," *URL: http://jmlr. org/papers/v12/pedregosa11a. html*, 2011.

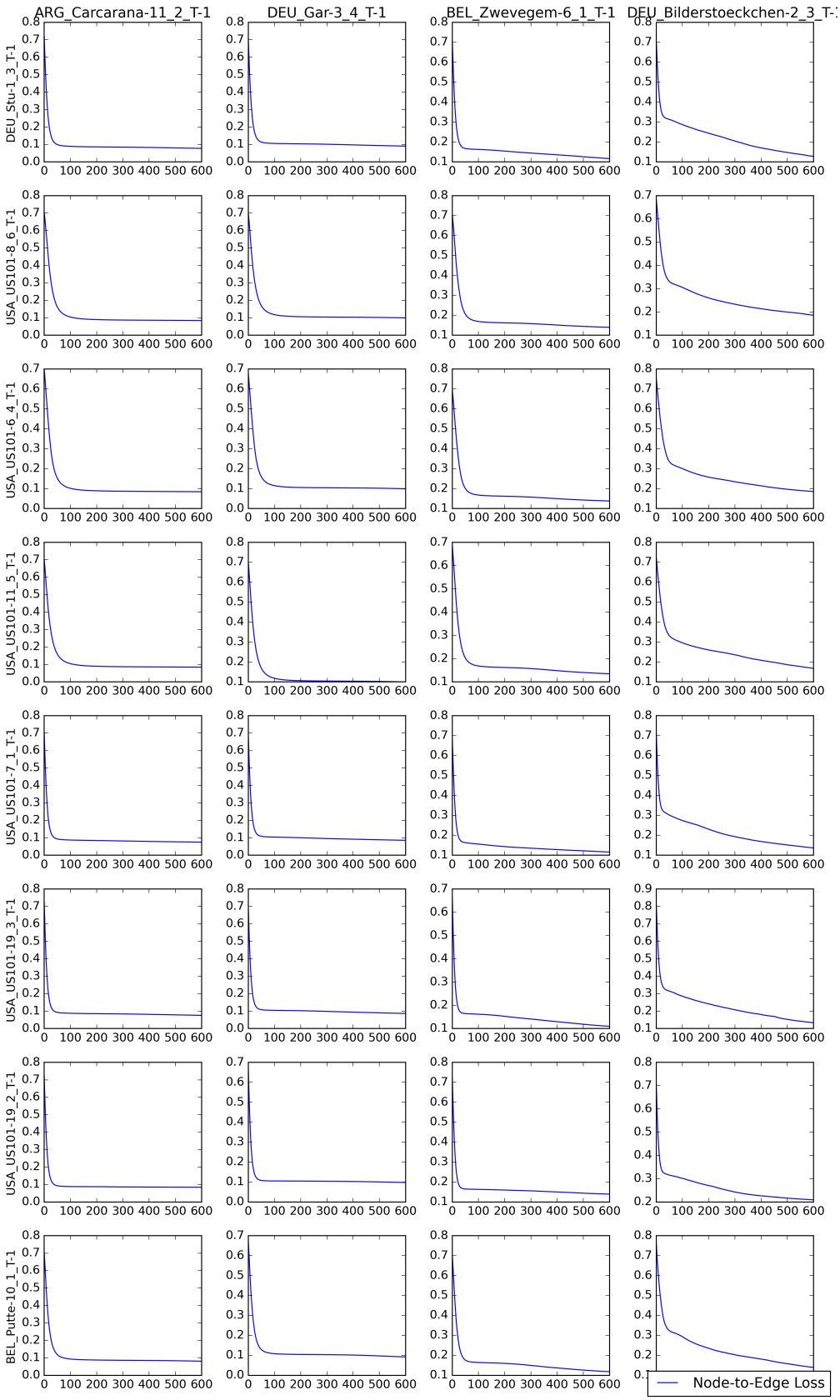

Fig. 11: The scenario graphs selected as the target graphs were used to train attention and reverse attention networks for the lanelet structure combined with vehicle behavior similarity analysis. On the figures, the multilabel classification losses of 4 randomly selected graphs (columns) using the graph encoders (rows) can be seen. The models were trained without using node and edge label regularizations since we only have *Lanelet* nodes and *isAdjacent* edges in these representations, therefore only the multilabel classifier loss values based on target graph adjacency predictions are shown.

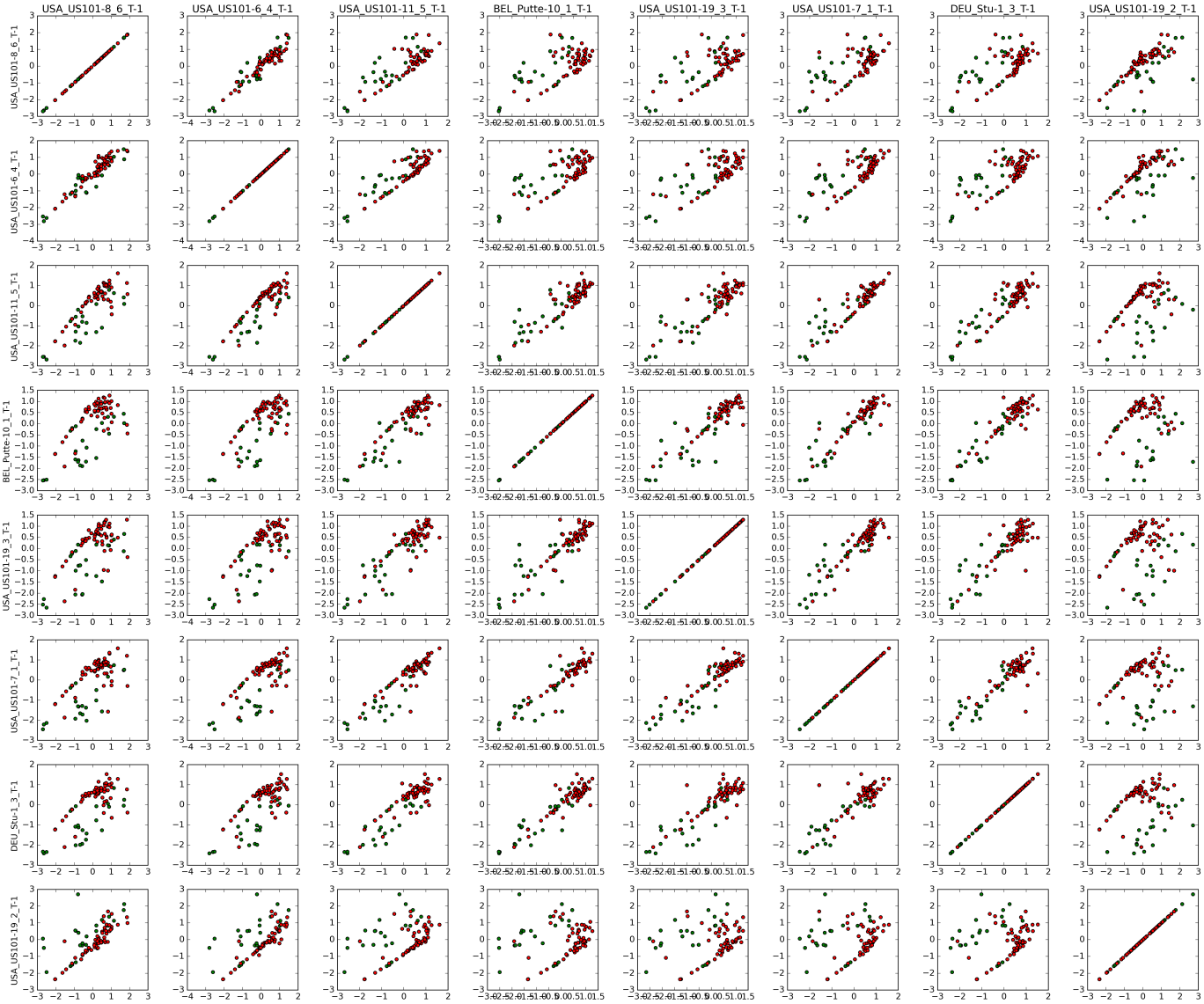

Fig. 12: The distribution of scenario graphs based on their divergence scores for the lane structure analysis are shown in the figures. Combinations of source graphs are placed on the x and y axes of figures, and the target graphs are plotted based on their corresponding divergence scores in order to analyze the correlation between the dimensions of the embedding vectors of target graphs. Additionally, scenarios with *Comfort* label are shown as green samples, and the scenarios with *Critical* label are shown as red samples.

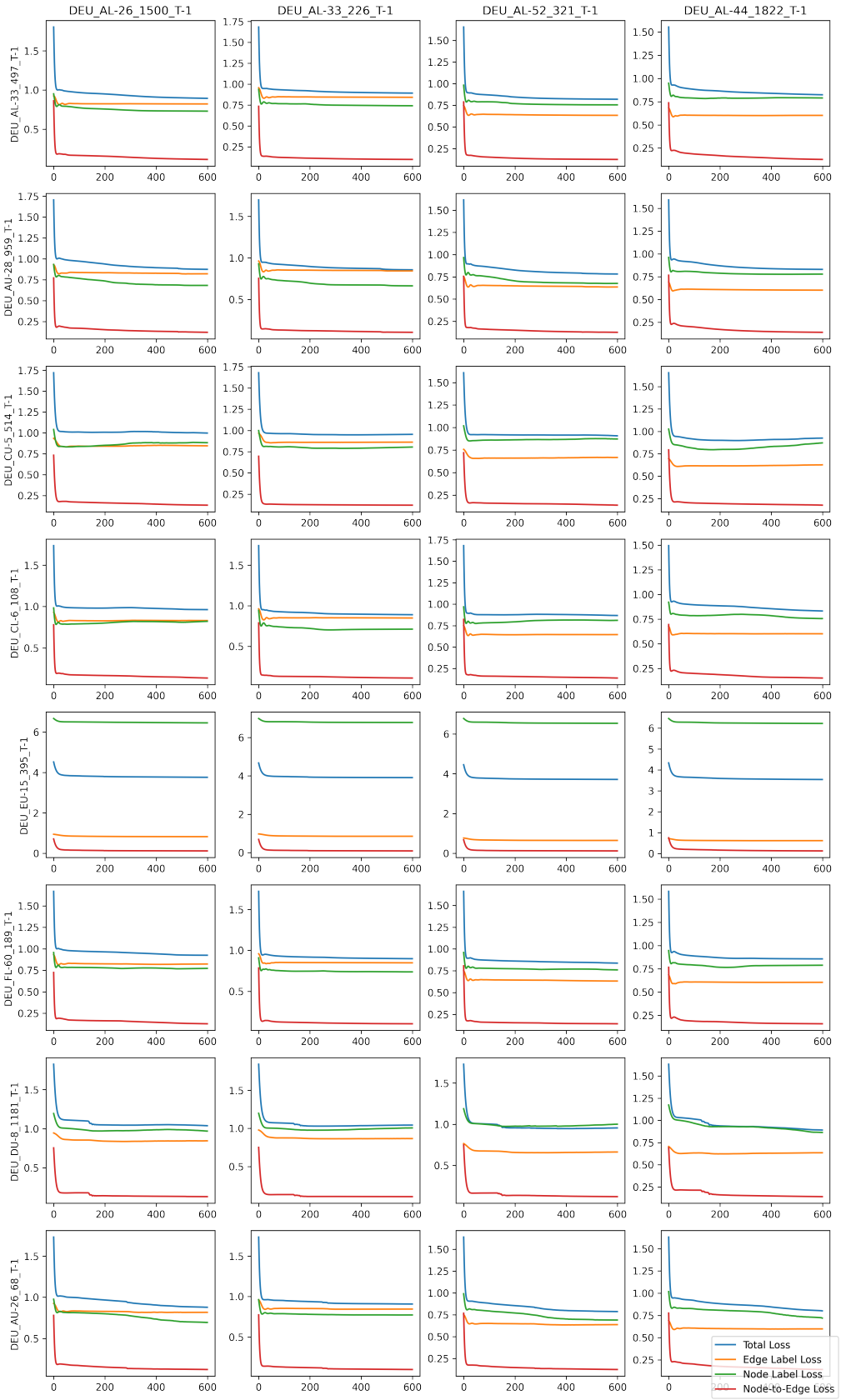

Fig. 13: The scenario graphs selected as the target graphs were used to train attention and reverse attention networks for the lanelet structure combined with vehicle behavior similarity analysis. On the figures, the multilabel classification losses of 4 randomly selected graphs (columns) using the graph encoders (rows) can be seen. The models were trained with using node and edge label regularizations, where their values over the duration of training can be seen in the figures.

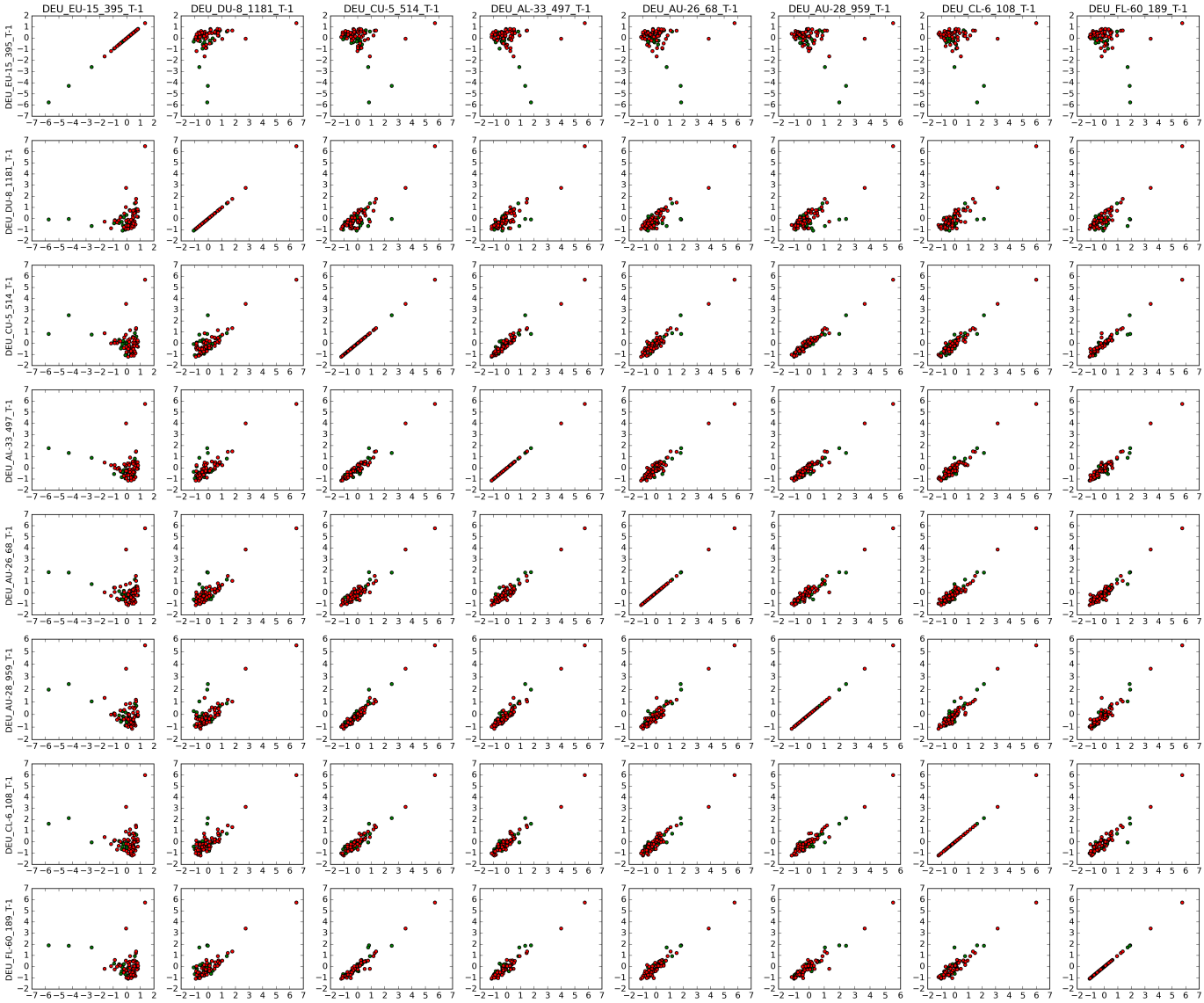

Fig. 14: The distribution of scenario graphs based on their divergence scores for the lane structure combined with vehicle behavior analysis are shown in the figures. Combinations of source graphs are placed on the x and y axes of figures, and the target graphs are plotted based on their corresponding divergence scores in order to analyze the correlation between the dimensions of the embedding vectors of target graphs. Additionally, scenarios with *Comfort* label are shown as green samples, and the scenarios with *Critical* label are shown as red samples.