SUMMARY

The use of routing services has become wide-spread and are used by both companies and private consumers. Routing is a key feature of popular services such as Google Maps and Bing Maps.

Vehicle routing algorithms rely on weighted graph representations of road networks, where an edge represents a road segment. In such graphs the weight of an edge is the cost, e.g. travel time or fuel consumption, associated with traversing the edge. The quality of a route suggested by a vehicle routing algorithm therefore depends on the accuracy of the edge weights.

GPS devices are cheap and collecting large quantities of vehicle trajectory data has therefore become inexpensive. As a result, several models have been devised to assign time-varying weights to edges or paths in road networks based on GPS trajectories.

Models for assigning time-varying weights in road networks can broadly be categorized as aggregation models, which provide estimations of cost distributions for edges or routes based on aggregates of GPS trajectories, and parametric models which fit a function to the feature representations of road segments or routes.

Aggregation models require a sufficient amount of data to provide an accurate estimation and therefore discard GPS trajectories if the number of GPS trajectories is below a threshold, i.e. there are too few GPS trajectories to provide an accurate estimate.

Parametric models are based on machine learning algorithms and rely on abstract feature representations of road segments or routes. Their estimations are therefore not limited to only road segments or routes with a sufficient number of GPS observations. The drawback is that

GPS observations are not used in the estimate beyond the training phase, even if an aggregate of the observations could provide a better estimate.

In general, much of the effort in deploying machine learning algorithms is spent on labor-intensive feature engineering: the design of preprocessing and data transformation pipelines. As a result, feature learning has gained traction within the machine intelligence community and has demonstrated empirical success in both industry and academia. Learned features may replace or supplement engineered features of existing parametric models, potentially increasing both accuracy and ease of deployment.

In this paper, we make two contributions that complement existing weight estimation models for road networks.

First, we devise a general framework for cost estimation in road networks. The framework updates a prior weight estimate based on available observations and thus bridges the gap between aggregation models and parametric models. The update procedure is based on Bayesian statistics and can be performed in constant time by storing just the number of observations and their mean.

We evaluate the cost estimation framework on a travel time estimation task where a prior estimate of an edge-interval pair is provided by a linear cost estimation model. The cost estimation framework decrease the MAPE by 7.43% (1.37 percentage points) using 100% of the training set. Much of the improvement is gained with just 20% of the training data, however, which decreases MAPE by 6.78% (1.38 percentage points). Although we have only considered updates of cost estimates for edge-interval pairs in this paper, it can in principle be extended to cost estimation of paths in general, and continuous time by sampling observations within some interval surrounding a point in time.

Second, we adapt feature learning techniques from the domain of natural language processing to learn features of edges in weighted graph representations of road networks. Our feature learning framework, road2vec, overcomes the problem of geodesically distant edges never co-occurring by capturing an abstract representation of an edge's surroundings based on its simple graph label. In addition to our general cost estimation framework, we present road2vec, an adaption feature learning techniques from the domain of language modelling, which learn representations of edges and edge descriptors (a representation of an edge's labels in our experiments).

We replace the engineered features of linear cost estimation baseline model with feature representations learned using road2vec and evaluate them both on a travel time estimation task. Using the learned feature representations reduces the MAPE by 7.54% (1.39 percentage points). In addition, the learned feature representations are very robust; reducing the training data from 100% to 20% increases the MAPE by just 1.0% (0.17 percentage points). Using the learned feature representations with a linear model in conjunction with the general cost estimation framework yields the most accurate model with a MAPE of 16.41%: a 9.39% (1.70 percentage points) lower MAPE than a baseline aggregation model and a 10.99% (2.03 percentage points) lower MAPE than a baseline parametric model. After parameter tuning, the learned feature representations for the model are learned in under 10 minutes using a road network of 115977 road segments, without any need for domain knowledge.

We also investigate the learned feature representations capability to capture structural similarity in road networks by using K-means clustering to find meaningful clusters on the road network of Northern Jutland, Denmark. The quality of the clusters are measured using Calinski-Harabasz Index and produce both coherent and well-separated clusters. Visualizing the clusters on the road network of Northern Jutland revealed several clusters containing structurally similar road segments, such as a cluster for important main roads in large towns, a cluster containing motorway segments, and a cluster containing roads through the town center of smaller towns.

The general cost estimation framework and the feature learning framework, road2vec, may be combined. With linear model MAPE is reduced by 9.39% (1.70 percentage points) and 10.99% (2.03 percentage points) compared to two baseline models.

Improving Cost Estimation Models with Estimation Updates and road2vec: a Feature Learning Framework for Road Networks

Martin Fruensgaard & Tobias Skovgaard Jepsen
{*mfruen12, tjeps12*}*@student.aau.dk*
Aalborg University

Abstract—The use of routing services has become wide-spread and are used by both companies and private consumers. Vehicle routing algorithms rely on weighted graph representations of road networks, where an edge represents a road segment and the weight of an edge is the cost associated with traversing the edge, e.g. a travel time. The quality of a route suggested by a vehicle routing algorithm therefore depends on the accuracy of the edge weights. As a result, several models have been devised to assign time-varying weights to edges or paths in road networks based on Global Positioning System (GPS) trajectories.

We make two distinct contributions in this paper. First, we describe a general cost estimation framework which updates a prior cost estimate of an cost estimation model based on GPS observations. Second, we propose a feature learning framework, road2vec, which adapt feature learning techniques from language modelling to learn representations of edges which can be used to not only train cost estimation models, but also capture structural similarity of road segments in road networks.

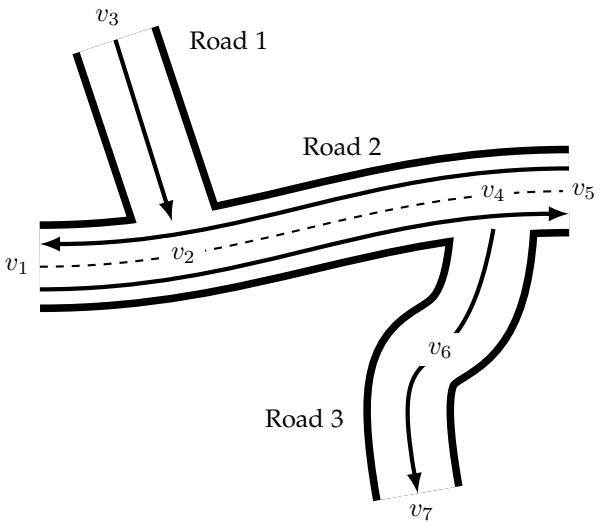
The general cost estimation framework and road2vec used separately reduces the estimation error of a baseline model by 7.43% and 7.54%, respectively, on a travel time estimation task. When used in conjunction, the estimation error is reduced by 9.39%.

1 Introduction

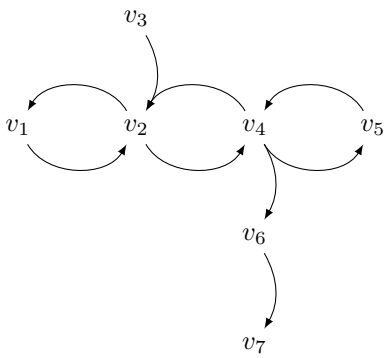
The use of routing services has become wide-spread and are used by both companies and private consumers. Routing is a key feature of popular services such as Google Maps [1] and Bing Maps [2].

Vehicle routing algorithms rely on weighted graph representations of road networks, where an edge represents a road segment [3], as illustrated by Fig. 1. In such graphs the weight of an edge is the cost, e.g. travel time or fuel consumption, associated with traversing the edge. The quality of a route suggested by a vehicle routing algorithm therefore depends on the accuracy of the edge weights.

GPS devices are cheap and collecting large quantities of vehicle trajectory data has therefore become inexpensive [4, 3]. As a result, several models have been devised to assign time-varying weights to edges or paths in road networks based on GPS trajectories [5, 6, 7, 8, 3, 9].



(a) Road Network.



(b) Graph Representation.

Fig. 1: Example of a road network (a) and its graph representation (b).

1.1 Weight Assignment Models

Models for assigning time-varying weights in road networks can broadly be categorized as aggregation models [8, 10, 9, 5], which provide estimations of cost distributions for edges or routes based on aggregates of GPS trajectories, and parametric models [6, 11, 7, 12] which fit a function to the feature representations of road segments or routes.

The estimated cost distributions of aggregation models approaches the population distribution as the number of GPS observations recorded from a road segment or route goes towards infinity. Given sufficient data, the aggregation models can therefore provide the most accurate estimates. Aggregation models are inherently data reliant and their cost distribution estimations are limited to road segments or routes for which there are sufficient data which is not always available. Thus they are severely affected by data sparsity. In addition, aggregation models discard data if the number of observations is below a certain threshold [8, 10, 5, 9]. Existing aggregation models rely on various strategies to overcome the data sparsity problem, such as

- simple, but naive, cost distribution estimations based on speed limit and length, and computationally expensive convolution of cost distributions [8, 10],
- using observations from temporally close time intervals and similar road segments [5], and
- estimating cost distributions of origin-destination pairs [9] rather than road segments or routes. This reduces the data sparsity problem, since there are $O(|2^E|)$ possible routes, but just $O(|V|^2)$ possible origin-destination pairs where V is the number of vertices and $E \subseteq V \times V$ is the set of edges in a graph representation of a road network.

Parametric models are based on machine learning algorithms [6, 11, 7, 12] and rely on abstract feature representations of road segments or routes. Their estimations are therefore not limited to only road segments or routes with a sufficient number of GPS observations. The drawback is that GPS observations are not used in the estimate beyond the training phase, even if an aggregate of the observations could provide a better estimate. Additionally, the performance of parametric models is strongly connected to the feature representation of road segments and routes.

In general, much of the effort in deploying machine learning algorithms is spent on labor-intensive feature engineering: the design of preprocessing and data transformation pipelines [13, 14]. As a result, feature learning has gained traction within the machine intelligence community and has demonstrated empirical success in both industry and academia [13, 15]. Learned features may replace or supplement engineered features of existing parametric models, potentially increasing both accuracy and ease of deployment.

1.2 Feature Learning in Graphs

To the best of our knowledge, there is no published work on feature learning of road segments or routes in road networks specifically, but several feature learning methods for graphs have recently emerged driven by advancements in feature learning in the domain of language modelling [14, 16, 17, 18]. The basic premise of these methods is that words which often co-occur with the same words in sentences, within a window of size c, are similar and will be embedded closely in a vector space which serves as the data representation of that word [16]. The notion of words is very flexible and may refer to both nodes and edges in graphs. The notion of sentence is strongly connected to the notion of similarity and sentence construction is therefore task-specific [14].

DeepWalk [16] was first to generalize feature learning of words to feature learning of nodes. They use nodes as words and generate sentences from random walks in graphs.

Like DeepWalk, node2vec [14] uses nodes as words, but generates sentences using a biased random walk which also takes into account the weights of edges; the higher the weight, the more likely the walk will travel along that edge. The biased random walk is parametrized, s.t. it can be adjusted to behave more like a breadth-first search, a depth-first search, or something inbetween.

Content-Enhanced Network Embedding (CENE) [17] expands upon DeepWalk and node2vec in the social network setting where some nodes have associated text content. This allows them to optimize a joint objective function based on both structural similarity and content similarity. Consequently, nodes that are similar both in terms of co-occurrence in sentences and in terms of their content will have similar data representations.

The above approaches claim to be applicable to graphs in general, but assume that similar nodes can be reached by traversing few edges and, in the case of node2vec, that these edges will have large weights. This assumption may be reasonable in social networks, but is inadequate in road networks for a number of reasons.

First, although road networks display spatial autocorrelation of the traversal cost across a road segment [11, 7], this tends to be conditional on characteristics of the road segment and its structural role [6]. For instance, we would not expect the traversal cost per unit of length of a motorway exit to resemble that of its connected motorway.

Second, geodesically distant[1] road segments may be similar, but DeepWalk, CENE, and node2vec generate sentences based on topology. As a result, road segments with a larger geodesic distance than the context size cannot co-occur, even if they are similar, and will therefore not be given similar feature representations.

1.3 Contributions

In this work, we make two contributions that complement existing weight estimation models for road networks.

First, we devise a general framework for cost estimation in road networks. The framework updates a prior weight estimate based on available observations and thus bridges the gap between aggregation models and parametric models. The update procedure is based on Bayesian statistics and can be performed in constant time by storing just the number of observations and their mean.

Second, we adapt feature learning techniques from the domain of natural language processing to learn features of edges in weighted graph representations of road networks.

1. The geodesic distance between two nodes is the length of the path between them which contains the fewest edges.

Our feature learning framework, road2vec, overcomes the problem of geodesically distant edges never co-occurring by capturing an abstract representation of an edge's surroundings based on its simple graph label. Training a simple linear model with the learned features display superior estimation accuracy and greater robustness to data sparsity than the baseline methods on the task of travel time estimation of trips. In addition, they are capable of finding structural similarity on road networks.

The general cost estimation framework and road2vec used separately reduces the estimation error of a linear baseline model by 7.43% and 7.54%, respectively, on a travel time estimation task described in Section 6. When used in conjunction, the estimation error is reduced by 9.39%.

The rest of the paper is arranged as follows. In Section 3 we describe how we model road networks and trips. In Section 4 we describe the general framework for cost estimation. In Section 5 we present our framework road2vec, and how it adapts feature learning methods from the domain of language modelling to the domain of road networks. In Section 6 we evaluate the general cost estimation framework and the feature representations learned using road2vec on a travel time estimation task. In Section 7 we cluster based on the learned feature representations of edges to find structural similarity in road networks. Finally, we close with our conclusion in Section 8 and future work in Section 9.

2 Related Work

To the best of our knowledge, there is no published work which uses GPS observations to update cost distribution estimates. The most closely related work to our cost estimation framework is therefore the general approach to Bayesian estimation of the mean on which our framework is based [19]. Our cost estimation framework is complementary to the existing weight assignment models discussed in Section 1.1.

The cost estimation framework complements aggregation models when the amount of GPS observations is below the threshold. In that case, such models default to simple estimates based on the speed limit and length of a segment and discard any available GPS observations. Our framework allows this simple estimate to be updated based on the available observations and thus enables aggregation models to take advantage of the available data, rather than discarding it.

Parametric models fit a function which is unlikely to be representative of any single road segment or route, but best describes all the road segments or routes in the training set as a whole. Our general framework complements parametric models by adjusting their estimate of a road segment or route based on its recorded observations. Effectively, this allows parametric models to inherit a property of aggregation models: estimated cost distributions approach the population distribution as the number of GPS observations recorded from a road segment or route goes towards infinity.

Existing feature learning techniques [16, 14, 17] for graphs have been discussed in Section 1.2. They all share the limitation that nodes can only co-occur in sentences if their geodesic distance is smaller or equal to the context size. Our feature learning approach adresses this issue by constructing sentences for each edge in the network in which the words are descriptors: abstract representations of the edge and nearby edges that are independent of the road network topology. During feature learning a neighborhood (analogue to a sentence) is sampled for each edge and during training on the neighborhood the edge is given as an extra context word. This produces an identical feature representation of edges if the same neighborhoods are sampled for them.

Another departure from existing feature learning techniques for graphs is our choice of architecture. We use the Distributed Memory Model of Paragraph Vectors (PVDM) architecture, an extension of the Continuous Bag of Words (CBOW) architecture [20], which learns feature representations of words and paragraphs jointly. In the case of our feature learning framework, feature representations are learned for both the edge descriptors (words) and the edges (paragraphs). PVDM predicts a word given its surrounding words and the sentence, paragraph, or document it occurs in [20]. Conversely, existing feature learning techniques are based on the SkipGram architecture [16, 14, 17], which predict the surrounding words given the middle word. In both cases, the probability of a word is computed using the softmax function [21, 20].

The reason for the difference in architecture is that PVDM displays superior performance to its SkipGram based counterpart for learning representations of paragraphs [20]; edges in our case. The PVDM architecture has the advantage of being much faster to train than the SkipGram architecture due to its smaller output layer [21], but in general the superior feature learning architecture in terms of accuracy depends on the task [21].

The faster training speed in combined with the small number of distinct descriptors used as words, makes our feature learning framework faster than existing feature learning techniques for graphs given the same corpus size. In addition, existing feature learning techniques approximate the softmax function when computing the probability of a word [16, 14, 17], but the small number of distinct descriptors used as words in our feature learning framework makes exact computation of the softmax feasible and improves the quality of the learned feature representations.

The most closely related feature learning technique for graphs, is CENE [17]. CENE incorporates textual content associated with nodes in social networks into their feature representation learning [17]. Similarly, the labels of edges in road networks can be incorporated into our feature representation learning approach as edge descriptors. Our approach to incorporating information associated with nodes or edges differs from CENE's, however. CENE [17] construct a graph containing both regular nodes and content nodes resulting in an augmented network consisting of node-node edges and node-content edges. They optimize a joint objective function based on both structural similarity and content similarity. This is done by including both regular nodes and content nodes in the corpus and representing content nodes as a combination, e.g. an average, of separately learned word representations representing words in the text content [17].

Our approach is similar to CENE in the sense that content in the form of edge descriptors may be included in the corpus as words, however, edges are not. Instead, the edge a neighborhood of descriptors is sampled from is

included in the context. In addition, we do not train feature representations of the descriptors separately, but jointly, and thus learn feature representations of both edge descriptors and edges.

3 PRELIMINARIES

3.1 Modelling Spatio-Temporal Road Networks

In this section, we model how spatio-temporal road networks as a graph.

Definition 3.1. *A road network is a weighted directed graph* $G = (V, E, T, L, l, w)$*, where* V *i.e. a set of* vertices, $E \subseteq V \times V$ *is a set of* edges, T *is a continuous time domain containing all times of day,* L *is a set of labelling functions that maps labels to edges,* $l \colon E \to \mathbb{R}^+$ *is a function mapping the length of each edge, and* $w \colon E \times T \to \mathbb{R}^+$ *assigns* time-varying weights *to all edges.*

Roads in a road network are divided into segments. A vertex $v_i \in V$ represents an intersection between segments or the end of a road.

An edge $(v_1, v_2) \in E$ represents a directed segment which allows travel from v_1 to v_2, as shown by Fig. 1. It is important to model segments as directed edges to represent driven directions and since otherwise one-directional segments, such as the segment going from v_3 to v_2 in Fig. 1, could not be represented.

The set of label functions L, depends on the map being used. In this project we use the map from OpenStreetMap (OSM) [22] and the city zones from *PlansystemDK* [23] to derive the label functions $category$, $limit$, $city$, $regulated_s$, and $regulated_t$.

$category(e)$ returns the road category of edge e which is either "motorway", "motorway link", "expressway", "highway", "main road", or "connecting road". See Table 9 for the mapping of OSM categories to these categories. $limit(e)$ returns the speed limit of edge e. $city(e)$ returns whether an edge e is within a city zone or not. $regulated_s(e)$ and $regulated_t(e)$ returns whether the source intersection v_s or target intersection v_t, respectively, is regulated by traffic lights for an edge $e = (v_s, v_t)$.

In road networks there is a natural flow of traffic from one edge to another, in which case we say they are *connected*.

Definition 3.2. *An edge* (v_1, v_2) *is connected to another edge* (v_2, v_3) *if* $v_1 \neq v_3$*, i.e. they have an intersection or end of segment in common and are not different directions of the same bidirectional segment.*

For example, edge (v_1, v_2) in Fig. 1b is connected to (v_2, v_3), but (v_1, v_2) is not connected to (v_2, v_1).

It is often useful to work on road networks with a discretized time domain where T is divided into intervals. We refer to this as a discretized road network. Let an interval from time $a \in T$ up to, but excluding time $b \in T$, be denoted as

$$[a; b) = \{t | t \in T \wedge a \leq t < b\}$$

Definition 3.3. *Given a road network* $G = (V, E, T, L, l, w)$*, its discretized road network is a weighted directed graph* $G = (V, E, T, L, l, w)15$*, where* Tg *is a set of intervals starting from midnight of size* g *and the granularity* $g \in \mathbb{R}^+$ *is in minutes. I.e.* $T_{15} = \{$*[0:00; 0:15)*$, \cdots,$ *[23:45; 0:00)*$\}$

3.2 Trips and Edge Traversal Costs

Trips are derived from GPS observations. A GPS trajectory $gpsTr = (gps_1, \ldots gps_n)$ is a sequence of GPS observations where a GPS observation $gps_i = (loc, t, cost')$ specifies the location loc and the normalized cost $cost'$, e.g. a recorded speed or fuel consumption per kilometer, of a vehicle at time $t \in T$. The GPS observations of a GPS trajectory are map-matched and pre-processed s.t. a $gps_i = (loc, t, cost')$ in a GPS trajectory is mapped to an edge record $edge_i = (e, t, cost')$. This yields edge trajectories of the form

$$edgeTr = ((e_1, t_1, cost'_1), \ldots, (e_n, t_n, cost'_n))$$

Finally, we map edge trajectories to trips. Depending on the frequency of the GPS observations, an edge record may occur several times consecutively in the edge records of an edge trajectory. Given a sequence of k edge records $(e, t_i, cost'_i), \ldots, (e, t_k, cost'_k)$ for the same edge, e, in an edge trajectory

$$\begin{aligned} edgeTr = ((e_1, t_1, cost'_1) \ldots, (e_i, t_i, cost'_i), \ldots, \\ (e_i, t_k, cost'_k), \ldots, (e_n, t_n, cost'_n)) \end{aligned} \tag{1}$$

We map such sequences in edge trajectories to 3-tuples $(e_i, t_i, cost_i)$ where $cost$ is the arithmetic mean of the costs $cost'_j$ for $i \leq j \leq k$. The 3-tuples are inserted into a trip s.t. the trip preserves the order of the edges in the edge trajectory. I.e., the edge trajectory in Eq. (1) is turned into a trip

$$trip = ((e_1, t_1, cost_1) \ldots, (e, t_i, cost_i), \ldots, (e_n, t_n, cost_n)) \tag{2}$$

Definition 3.4. *Given a road network* $G = (V, E, T, L, l, w)$*, a trip observed in* G *is a sequence of triples* $(e, t, cost)$*, where* $e = (v_1, v_2) \in E$*,* $t \in T$ *is the time of arrival on edge* e *and* $cost \in \mathbb{R}^+$ *is the normalized cost of traversing the edge computed as the mean normalized cost of the GPS observations matched to edge* e*.*

We denote the set of all observed trips in a road network $G = (V, E, T, L, l, w)$ as O_G. It is often useful to refer to only trips crossing a particular edge $e \in E$ during an interval $I \subset T$. For this purpose we use

$$O_G(e, I) = \{(e, t, cost) | (e, t, cost) \in O_G \wedge t \in T\} \tag{3}$$

or simply $O_G(e)$ if $I = T$, i.e. all trips that cross edge e.

4 GENERAL FRAMEWORK FOR COST ESTIMATION IN ROAD NETWORKS

In this section, we present a general framework for cost estimation in road networks. The framework updates a cost estimate of an edge-interval pair using observations from the edge-interval pair. The framework works with any cost estimation model, both aggregation models and parametric models. Aggregation models already base their estimates on observations, however. We therefore expect the framework to be most valuable for parametric models, since the framework allows parametric cost estimation models to inherit a desirable property of aggregation models: the updated cost estimate of a parametric model approaches the population mean as the number of observations goes toward infinity.

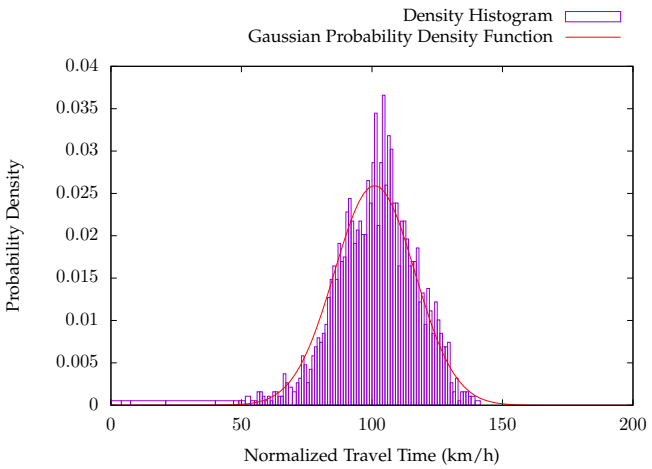


Fig. 2: The distribution of observed travel times normalized segment length of 1886 trips across a motorway segment represented as a density histogram of the relative frequency the travel times and the probability density function of its Gaussian approximation with mean 101.04 and variance 15.38^2.

A simple way of normalizing travel time to compare edge-intervals is to compute an expected speed. Given a road network $G_g = (V, E, T_g, L, l, w)$, the speed across an edge $e \in E$ during interval $I \in Tg$ is

$$speed(e, I) = \frac{l(e)}{cost(e, I)} \tag{4}$$

where $cost(e, I)$ is an expected travel time across edge e during interval I.

Travel time distributions are in general very complex and does not follow standard distributions, such as the Gaussian distribution [8]. However, when normalized according to Eq. (4), the distribution of travel times across an edge e during a 15 minute interval I does not deviate significantly ($p \leq 0.05$) from a Gaussian distribution (see Fig. 2) for 70% of edge-intervals in the Danish road network [6] using the Shapiro–Wilk normality test [24] on each edge-interval. We therefore assume that the underlying distribution of the normalized cost to traverse any edge-interval pair is Gaussian, where the mean cost of the distribution is interpreted as the normalized cost estimate for the edge-interval, and base our framework on Bayesian estimation of the mean [19].

To update the estimate of an edge-interval pair (e, I), the framework requires the following parameters

- A prior (normalized) cost estimate $cost_0$ which serves as an initial mean of the cost distribution of (e, I),
- an expected variance σ_0^2 of the mean which reflects the uncertainty of $cost_0$, and
- an expected variance σ^2 of the cost distribution of the edge-interval which reflects the observation noise.

All the parameters may differ for each edge-interval and may be found analytically, considered a (hyper)parameter or estimated by estimation models.

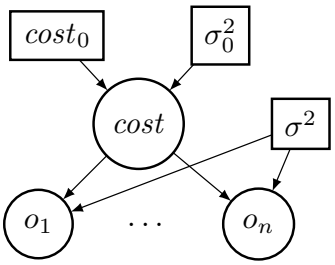


Fig. 3: Illustration of the conditional dependencies of the framework for cost estimation.

Fig. 3 gives an overview of the framework. The likelihood of the observations o_i for $1 \leq i \leq n$ is dependent on the mean of the distribution, $cost$, from which they are sampled and the variance of that distribution, σ^2. The mean, $cost$, in turn depends on the prior cost estimate, $cost_0$, and the expected variance of the mean, σ_0^2.

4.1 Bayesian Estimation of the Cost

We now formalize the cost estimation framework illustrated in Fig. 3.

The goal is to compute the posterior distribution of the mean cost of an edge-interval pair (e, I) given the observed trips $O_G(e, I)$. Using Bayes' theorem, this is computed as

$$\mathrm{P}(cost \mid O_G(e, I)) = \alpha \mathrm{P}(cost)\mathrm{P}(O_G(e, I) \mid cost) \tag{5}$$

where α is a normalization constant.

Since the underlying distribution is assumed to be Gaussian, the likelihood of the observations is [19]

$$\begin{aligned}\mathrm{P}(O_G(e, I) \mid cost) =& \mathrm{P}(O_G(e, I) \mid cost, \sigma^2) \\ =& \prod_{(e,t,c) \in O_G(e,I)} \mathrm{P}(c \mid cost, \sigma^2)\end{aligned} \tag{6}$$

The prior probability distribution of $cost$ is unknown and is therefore estimated as [19]

$$\mathrm{P}(cost) \propto \mathrm{P}(cost \mid cost_0, \sigma_0^2) \tag{7}$$

Combining Eqs. (5) to (7), yields

$$\begin{aligned}&\mathrm{P}(cost \mid O_G(e, I)) \propto \\ &\quad \mathrm{P}(cost \mid cost_0, \sigma_0^2) \prod_{(e,t,c) \in O_G(e,I)} \mathrm{P}(c \mid cost, \sigma^2)\end{aligned} \tag{8}$$

which is a product of two Gaussian distributions and is thus itself a Gaussian distribution [19], and can be written as [19]

$$\mathrm{P}(cost \mid O_G(e, I)) \propto \mathcal{N}(cost \mid cost_n, \sigma_n^2) \tag{9}$$

where $n = |O_G(e, I)|$ is the number of obs

The parameters for $\mathcal{N}(cost \mid cost_n, \sigma_n^2)$ are computed as

$$\sigma_n^2 = \left(\frac{n}{\sigma^2} + \frac{1}{\sigma_0^2} \right)^{-1}$$

and

$$cost_n = \sigma_n^2 \left(\frac{cost_0}{\sigma_0^2} + \frac{\sum_{(e,t,c) \in O_G(e,I)} c}{\sigma^2} \right)$$

where $cost_n$ is *the updated estimate*. In our experiments we store the number of observations n and the observation mean and compute the updated estimate in constant time.

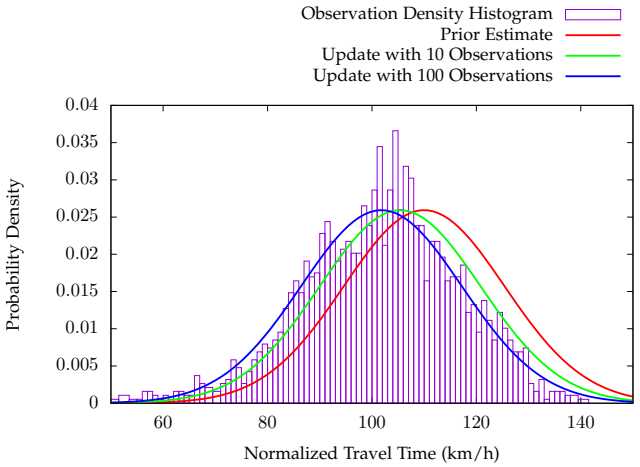


Fig. 4: Gaussian distributions estimating the density histogram representation of the distribution of observed travel times with means $cost_0$, $cost_{10}$, and $cost_{100}$, where $cost_0$, $\sigma_0^2 = 5^2$, and $\sigma^2 = 15.38^2$.

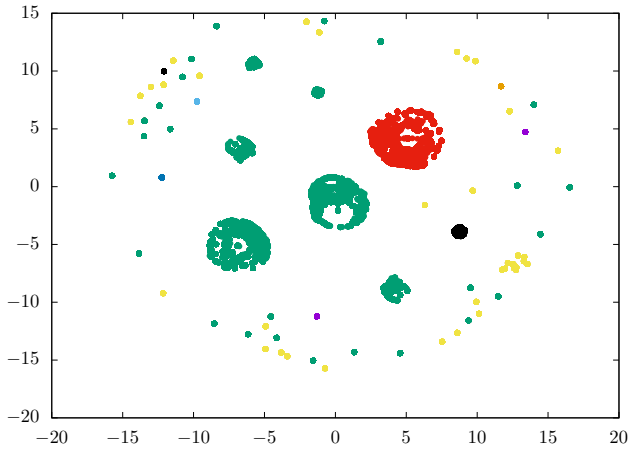


Fig. 5: 2-dimensional TSNE projection of learned feature representations of edges.

We demonstrate the framework in Fig. 4, by updating a prior estimate of the edge-interval shown in Fig. 2. The prior estimate $cost_0 = 110$ equals the speed limit on the edge. As the figure shows, the updated estimate gradually approximates the distribution mean as more observations are used to update the estimate. We refer to Fig. 34 in Appendix J for further information on how the variance of the mean σ_0^2, the observation variance σ^2, and the number of observations n affect the updated estimate $cost_n$.

In our discussion of the framework, we have focused on updating the estimate based on the distribution of individual edge-intervals, but it can in principle also be used for paths or routes in the road network.

5 Feature Learning in Road Networks

In this section, we describe how we adapt feature learning techniques from natural language modelling to learn feature representations of edges representing road segments in road networks.

Learned feature representations of edges may be used directly by existing parametric models for cost estimation in road networks [6, 11, 7, 12], but may also be used to find structural similarity in road networks. As an example, consider the TSNE [25] projection of the learned feature representations of edges in Fig. 5 which produced the best clusters for the structural similarity task in Section 7. The black cluster primarily consists of motorway segments, whereas the red cluster are important main roads through the city centers of large towns.

In the following, we first review the related backround from the language modelling domain.

5.1 Neural Language Models

Traditionally, the goal of language models is to estimate the likelihood of a sequence of words occurring in a corpus [16]. More formally, given a sequence of training words $(w_1, \cdots, w_T)$, the objective is to maximize the likelihood of a word given the n previous words, i.e. $\mathrm{P}(w_t \mid w_{t-1}, \ldots, w_{t-n})$.

Words have traditionally been represented using a *one-hot encoding* [21]. In a one-hot encoding, all the words in the vocabulary $w_i \in \mathcal{V}$ are enumerated s.t. $1 \leq i \leq |\mathcal{V}|$. The one-hot encoding of a word $w_i \in \mathcal{V}$ is a $1 \times \mathcal{V}$ row vector $\mathbf{x}_i = (x_1, \ldots, x_{|v|})$ where some $x_i = 1$ and the remaining values are 0. The intuition is that each unique word is represented by a unique vector, see Fig. 6 for an illustration.

One-hot encodings are discrete and contains no notion of similarity which limits their predictive power [21]. Recent work in natural language processing has therefore focused on using probabilistic neural networks to learn continuous feature representations of words [21, 20] from their one-hot encodings. These models use the notion of *context*, typically defined as a fixed number of previous and future words surrounding a target word in a sentence [21]. We shall discuss one such model in Section 5.2.

5.2 Distributed Memory Model of Paragraph Vectors

1 We base our feature learning framework on the Distributed Memory Model of Paragraph Vectors (PVDM) neural network architecture, which is designed to learn feature representations of both words and collections of words, such as paragraphs, jointly [20].

PVDM is an extension of the well-known CBOW architecture [21, 20]. CBOW predicts a word based on its context: the c previous and future words in a sentence [20]. As an example, consider the sentence illustrated in Fig. 6. In the CBOW architecture the word "a" is predicted based on its context. If the context size $c = 2$, the context of "a" is ("This", "is", "short", "sentence"). Formally, CBOW maximizes

$$\frac{1}{T}\sum_{t=c}^{T-c} \log(\mathrm{P}(w_t \mid w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c})) \qquad (10)$$

where c is the size of the context.

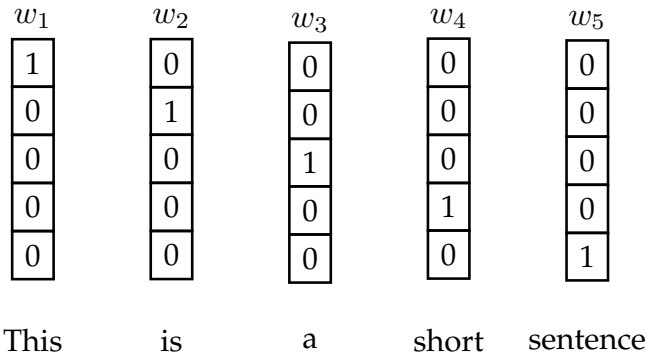


Fig. 6: The sentence "This is a short sentence" with each word w_i for $1 \leq i \leq 5$ represented as its one-hot encoding.

PVDM extends the CBOW model by providing which word collection (e.g. a paragraph) as extra context information, reflecting the intuition that the probability of a word is dependent on the topic of the word collection in which it occurred [20]. For instance, we expect the word "apple" to be more likely to occur in a cooking book, than in a computer science book. When predicting a word, PVDM therefore not only considers the surrounding words, but also the collection from which the word originates. As an example, consider again prediction of the word a in the sentence illustrated by Fig. 6. Suppose the sentence occurs in two different paragraphs, p_1 and p_2. If the context size $c = 2$, the context of "a" in paragraph p_1 is $(p_1$, "This", "is", "short", "sentence") and the context of "a" in paragraph p_2 is $(p_2$, "This", "is", "short", "sentence"). Formally, PVDM maximizes

$$\frac{1}{T}\sum_{t=c}^{T-c} \log(\mathrm{P}(w_t \mid p_t, w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c})) \quad (11)$$

where p_t is the paragraph of word w_t.

In the following, we first describe how the feature representations are learned in the PVDM architecture, and then how the posterior probability of a word, e.g. $\mathrm{P}(w_t \mid p_t, w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c})$ in Eq. (11), is computed.

5.2.1 Feature Representations of Words and Paragraphs

Fig. 7 illustrates the PVDM architecture. Let $\mathcal{V}$ and P denote the set of all words and the set of all paragraphs occurring in a text corpus, respectively. Each context word w_i is given as a one-hot encoded $1 \times |\mathcal{V}|$ vector $\mathbf{x}_i$ at the input layer along with one-hot encoded $1 \times |P|$ vector $\mathbf{x}_i^p$ representing the context paragraph p_i the context words originate from. The context words have a shared weight $|\mathcal{V}| \times d$ matrix $\mathbf{W}^I$ connecting the input layer to the (hidden) embedding layer containing d neurons, whereas the paragraph has its own $|P| \times d$ weight matrix $\mathbf{W}^P$ connecting it to the embedding layer.

Since each word $w_i \in \mathcal{V}$ is represented as a one-hot encoded vector, w_i corresponds to the ith row in $\mathbf{W}^I$, i.e. $\mathbf{W}^I_{i,*} = \mathbf{x}_i\mathbf{W}^I$. Similarly, a paragraph $p_i \in P$ corresponds to the ith row in $\mathbf{W}^P$, i.e. $\mathbf{W}^P_{i,*} = \mathbf{x}_i^p\mathbf{W}^P$. During training, PVDM adjusts the weight matrices $\mathbf{W}^I$ and $\mathbf{W}^P$ are updated (along with $\mathbf{W}^O$) to maximize the objective function in Eq. (11). In our experiments in Section 6 we train both $\mathbf{W}^P$'s and $\mathbf{W}^I$'s vectors the same way as Le and Mikolov [20], using stochastic gradient descent where the gradient is obtained via backpropagation. After training, the ith row in $\mathbf{W}^I$ and $\mathbf{W}^P$ is therefore a *learned feature representation* of word $w_i \in \mathcal{V}$ and paragraph $p_i \in P$, respectively. If two paragraphs contain the nearly the same text their learned feature representations will be similar. Likewise, if words often co-occur with the same words and in similar paragraphs, their learned feature representations will be similar.

5.2.2 Computing Word Probability

The input to the hidden layer in Fig. 7 is the feature representations of the context words and the context paragraph and the values at the hidden layer is the average of the input feature representations [20].

Let $\mathbf{f}_w^I$ denote the feature representation of a word $w \in \mathcal{V}$, and let $\mathbf{f}_p^P$ denote the feature representation of a paragraph $p \in P$. Formally, given context words $(w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c})$ from a paragraph p_t, the values at the hidden layer are [26, 20]

$$\mathbf{h} = \frac{1}{1+2c}\left(\mathbf{f}_{p_t}^P + \sum_{-c \leq i \leq c, i \neq 0} \mathbf{f}_{w_{t-i}}^I\right) \quad (12)$$

As illustrated by Fig. 7, there is a different $d \times |\mathcal{V}|$ matrix $\mathbf{W}^O$ representing the connections between the hidden layer and the output layer. Using these weights, a score u_i is computed for each word $w_i \in \mathcal{V}$ in the output layer as [26]

$$u_i = \mathbf{h}W^O_{*,i} \quad (13)$$

where $W^O_{*,i}$ is the ith column of matrix W^O. Finally, the output layer illustrated in Fig. 7 computes the posterior probability of each word $w_i \in \mathcal{V}$ using the softmax function [20]:

$$\mathrm{P}(w_i \mid p_t, w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c}) = \frac{\exp(u_i)}{\sum_{-c \leq i \leq c, i \neq 0}^{|\mathcal{V}|} \exp(u_{t+i})} \quad (14)$$

The denominator of Eq. (14) scales linearly with the size of the vocabulary, usually tens of thousands of words [21, 20]. When training on large text corpora containing millions of words, exact computation of the softmax becomes intractable and therefore it is typically approximated [21, 20]. Approximation is unnecessary in our approach to feature learning of edges, described in Section 7.3, since the vocabulary contains just a few houndred words and is therefore much smaller than that of typical text corpora from the language modelling domain.

5.3 road2vec

In this section we introduce our feature learning framework *road2vec* which learn feature representations of edges representing road segments in road networks. The framework is an adaption of the PVDM architecture. In language modelling, the context is dependent on the text corpus, such as a collection of documents. A challenge is therefore to generate a meaningful corpus for road networks.

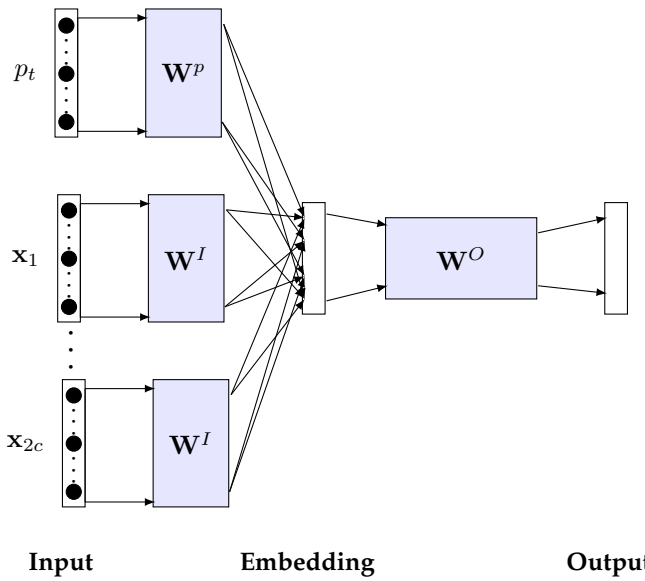


Fig. 7: An illustration of the PVDM architecture. Note that the matrix $\mathbf{W}^I$ is duplicated for each word in the context, s.t. each row yields a feature representation of each word in the vocabulary.

Intuitively, a road segment may be described by its surrounding segments. For instance, a road segment is likely in a city if surrounding segments are in a city. We therefore model an edge as a neighborhood containing surrounding edges analogue to a paragraph containing a collection of words. The notion neighborhood is not limited to immediate neighbors and we discuss neighborhood sampling strategies in Section 5.3.1.

The word representation of each edge in a neighborhood cannot be distinct for each edge, since this would limit co-occurence by topology similar to existing feature learning technique for graphs, as discussed in Section 1.2. We therefore assign a *descriptor* to each edge in the neighborhood which acts as an intermediate feature representation. The intuition is that edges which generate neighborhoods with the same edge descriptors, will be given the same learned feature representation.

The function $\delta\colon E \to \Sigma$ assigns descriptors to edges from a set of descriptors Σ. The descriptor of an edge must be *discrete* s.t. it can be represented as a word, and *topologically independent* s.t. geodesically distant but similar edges may be given theo same descriptor. To ensure topological independence, we expect $|\Sigma| \ll |E|$ s.t. each edge $e \in E$ in the neighborhood is not mapped to a distinct descriptor allowing edges across the road network to have similar neighborhoods.

We generate a corpus of edge neighborhoods s.t. each edge has at least one neighborhood with vocabulary $\mathcal{V} \subseteq \Sigma$ and paragraphs $P = E$. From the perspective of language modelling, an edge is analogue to a paragraph and a word is analogue to a descriptor. Thus we can rewrite Eq. (11) to

$$\frac{1}{T}\sum_{t=c}^{T-c} \log(\mathrm{P}(e_i^\delta \mid e, e_{i-c}^\delta, \ldots, e_{i-1}^\delta, e_{i+1}^\delta, \ldots, e_{i+c}^\delta)) \quad (15)$$

where $e_i^\delta = \delta(e_i)$.

In our experiments in Section 6 and Section 7, we simply use the labels of an edge as its descriptor, i.e. $\delta = L$. In this case a descriptor (or word) is of an edge e is of the form

$$\{category(e), limit(e), city(e), regulated_s(e), regulated_t(e)\}$$

e.g. $\{$"motorway", $110, false, false, false\}$.

Using either of the corpus generation strategy described in Section 5.3.1, the corpus contains $|P| = 215009$ edges and $|\mathcal{V}| = 214$ in our structural similarity experiment described in Section 7.

5.3.1 Corpus Generation Strategies

We consider two corpus generation strategies: k-Neighbors and k-Routes.

5.3.1.1 k-Neighbors

k-Neighbors is a simple corpus generation strategy based on the idea that an edge is described by its surroundings. To represent an edge $e \in E$, the k-Neighbors corpus generation strategy therefore samples the set of all edges within a geodesic distance of k for $k \geq 1$ from e in random order before mapping them to their descriptors. We denote the k distance neighborhood of edge e as N_e^k. For $k = 1$,

$$N_e^1 = \{e' \mid e' \in E, \text{e and e' are connected}\} \cup \{e\}$$

otherwise, for $k > 1$,

$$N_e^k = \{e' \mid e' \in N_e^{k-1}, e'' \in E, \text{e' and e'' are connected}\} \cup N_e^{k-1}$$

The sampling is illustrated in Fig. 8 for edge e_4 and $k = 1$, where edges $e \in N_{e_4}^1$ are represented as solid lines.

In practice, the order of the edges in the neighborhood is important and multiple neighborhoods may therefore be sampled from the same edge, but in different order. The complete k-Neighbors corpus generations strategy is outlined in Algorithm 1. At Line 4 the neighborhood is

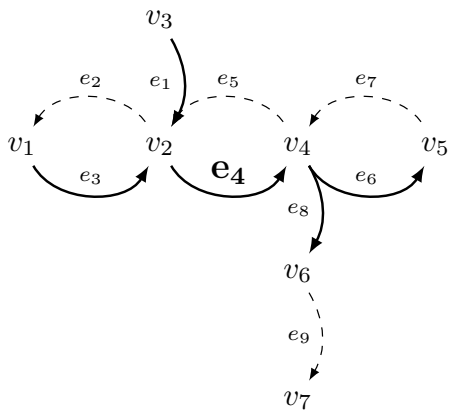


Fig. 8: All edges within a geodesical distance of $k = 1$ from e_4 represented as solid lines.

samples and then shuffled to ensure random order at Line 5. At Line 6 the edges are mapped to their descriptors before being added to the set of sentences used to represent the edge e at Line 7. The procedure is repeated n times, sampling a total of n neighbourhoods in different order. We refer to Fig. 23 in Appendix B for a detailed illustration of the k-Neighbors corpus generation strategy.

Algorithm 1 The k-Neighbors Corpus Generation Strategy.

Require: A time-dependent graph $G = (V, E, T, L, l, w)$, an edge $e \in E$ to sample a sentence for, a distance to neighbors d, number of sentences to sample n

1: **function** k-NEIGHBORS(e, n)
2: $\quad\mathcal{N} \leftarrow \emptyset$
3: $\quad$**repeat** n **times**
4: $\quad\quad neighbors \leftarrow N_e^k$
5: $\quad\quad Shuffle(neighbors)$
6: $\quad\quad neighbors \leftarrow \{\delta(e) \mid e \in neighbors\}$
7: $\quad\quad\mathcal{N} \leftarrow \mathcal{N} \cup \{neighbors\}$
8: $\quad$**return** $\mathcal{N}$

5.3.1.2 *k-Routes*

The k-Routes corpus generation strategy is based on vehicle behaviour. For each edge, the possible routes or paths across it are sampled as neighborhoods of the edge. As such it is inherently ordered, as opposed to the unordered k-Neighbors strategy.

Let $e_1 \rightarrow e_2$ denote that e_1 is connected to e_2. We use the notation $e_1 \xrightarrow{n} e_{d+1} = e_1 \rightarrow e_2 \rightarrow \cdots \rightarrow e_{n+1}$ to denote that e_1 is transitively connected to e_{d+1} through n intersections. A route $e_1 \xrightarrow{d} e_{n+1}$ is equivalent to an ordered multi-set $\{e_1, \ldots, e_{n+1}\}$.

We denote a route across an edge e with k previous and following edges as

$$e_{s-k} \xrightarrow{k} e_s \xrightarrow{k} e_{s+k} = \{e_{s-k}, \ldots, e_s, \ldots, e_{s+k}\}$$

resulting in a $2k + 1$ length route. For each edge $e \in E$, we collect the set of all such routes $\mathcal{R}_{e_s}^k$ s.t.

$$\mathcal{R}_e^k = \bigcup_{e' \in E} \bigcup_{e'' \in E} \{e' \xrightarrow{k} e \xrightarrow{k} e''\}$$

We can construct $\mathcal{R}_e^k$ by concatenating all combinations of routes of length k ending in e and routes starting from e, s.t. the concatenation of routes $e' \xrightarrow{k} e$ and $e \xrightarrow{k} e''$ yields the route $e' \xrightarrow{k} e \xrightarrow{k} e''$. We use a depth-first search to find

- the routes of length k starting from e, i.e. all routes of the form $e \xrightarrow{k} e'$ for all $e' \in E$, and
- the routes of length k e, i.e. all routes of the form $e' \xrightarrow{k} e$ for all $e'' \in E$.

This results in a time complexity of $\mathcal{O}(2|E|\mathrm{con}^{k+1})$ where con is the maixmum number of connected edges of any edge in the road network. Analytically, we have found $\mathrm{con} = 9$ in the road network of Northern Jutland used in our structural similary experiments in Section 7, but on average each edge is connected to just 3.38 edges.

Fig. 9 illustrates the different routes of k-routes of length $2k + 1$ across e_4 for $k = 1$. Observe that the k-Neighbors and k-Routes collect the same edges. In fact, $N_e^k = \bigcup_{r \in \mathcal{R}_e^k} r$. The k-Routes differs by limiting the possibilities of co-occurrence. For instance, the descriptors of edges e_1, e_5 and e_6 can co-occur when using k-Neighbors (see Fig. 8), but cannot using k-Routes.

The complete k-Routes corpus generation strategy is outlined in Algorithm 2. The loop at Line 3 iterates through all routes in $\mathcal{R}_e^k$ and maps each edge in each route to its descript at Line 4, before adding it to the set of mapped routes R. Finally, R is returned at Line 6. We refer to Fig. 24 in Appendix B for a detailed illustration of the k-Routes corpus generation strategy.

Algorithm 2 The k-Routes Corpus Generation Strategy.

Require: A time-dependent graph $G = (V, E, T, L, l, w)$, an edge $e \in E$, and a route length parameter k.

1: **function** k-ROUTES(e)
2: $\quad R \leftarrow \emptyset$
3: $\quad$**for each** $route \in \mathcal{R}_e^k$ **do**
4: $\quad\quad route^\delta \leftarrow \{\delta(e) \mid e \in route\}$
5: $\quad\quad R \leftarrow R \cup route^\delta$
6: $\quad$**return** R

6 TRAVEL TIME ESTIMATION

In this section we evaluate the general framework described in Section 4 and our feature learning approach described in Section 5 on the task of travel time estimation of trips. We evaluate the performance of travel time prediction models using Mean Absolute Percentage Error (MAPE) [27], defined as

$$MAPE = \frac{100\%}{n} \sum_{i=1}^{n} \left|\frac{a_i - p_i}{a_i}\right| \tag{16}$$

where n is the number of instances in the test set, a_i is the actual travel time of trip i and p_i is the predicted travel time given by a travel time estimation model.

The models under evaluation are all designed to estimate an expected speed across individual edge-interval pairs in the road network. These estimates are denormalized to a

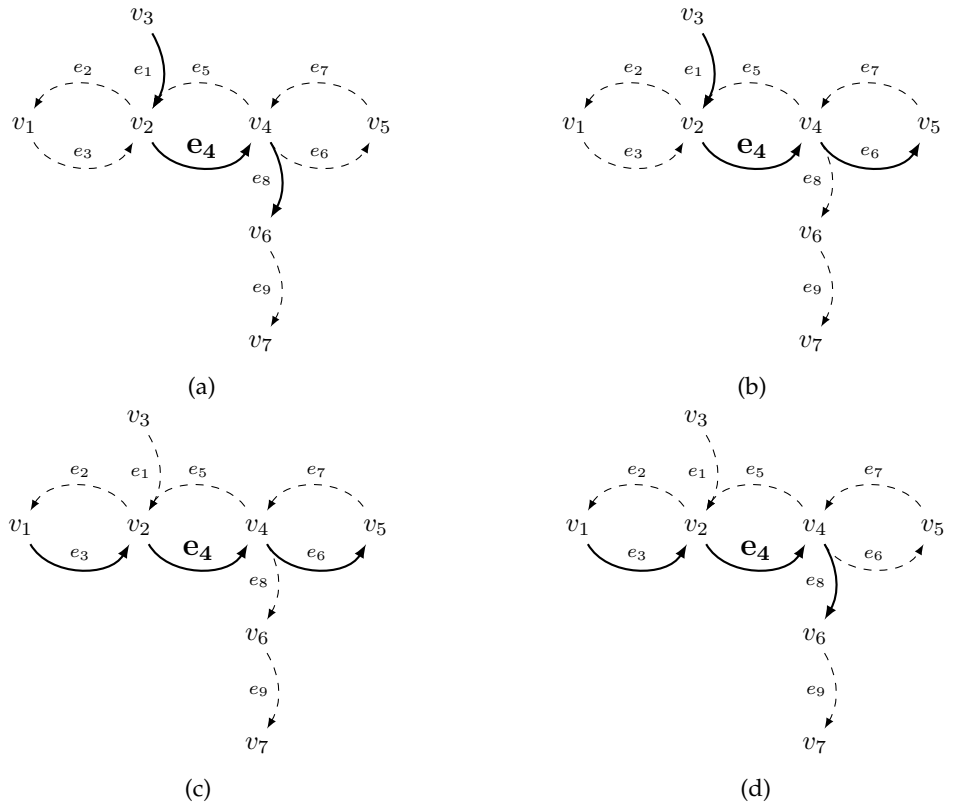


Fig. 9: An illustration of the k-Routes sampling strategy from edge e_4 with $k = 1$.

travel time. Given a road network $G = (V, E, T, L, l, w)$, w is therefore defined as

$$w(e, t) = \frac{l}{cost_n} \tag{17}$$

where $cost_n$ is the speed estimate of (e, t) provided by a model, in accordance with the notation of our general framework, see Section 4. If the model is not used with the framework the number of observations $n = 0$.

After denormalization, the prediction is performed by summing of the individual travel time estimations of edges in a trip. This procedure is illustrated in Algorithm 3. The time-dependent weight function w of the road network G is supplied by a trained cost estimation model. At each iteration, the traversal cost of the current edge in the trip e_i at time t_c is updated. t_c is initially set to the beginning of the trip at Line 2 and is incremented based on the predicted travel time at each iteration at Line 6. For instance, if t_c = 8:19 and $w(e_i, t_c)$ equals 5 minutes, t_c is incremented s.t. t_c = 8:24 at the next iteration. Note that the prediction is based only on the start time of the trip and the edges traversed, simulating the conditions of travel time estimation during routing.

Algorithm 3 The procedure for estimating a trip as a sum of edge-time predictions.

Require: A road network $G = (V, E, T, L, l, w)$ where w is supplied by a cost estimation model, a set of trips O_g recorded in G, and a trip $trip = ((e_1, t_1, cost_1), \ldots, (e_n, t_n, cost_n)) \in O_G$.

1: **function** TRIPPREDICTION(trip)
2: $t_c \leftarrow t_1$
3: $trip\_cost \leftarrow 0$
4: **for** $i = 1$ to n **do**
5: $trip\_cost \leftarrow trip\_cost + w(e_i, t_c)$
6: Increment t_c by $w(e_i, t_c)$
7: **return** $trip\_cost$

Set	*No. of Trips*
Training	17708 (60%)
Validation	2930 (10%)
Test	8773 (30%)
Total	29411 (100%)

TABLE 1: The distribution of trips in our dataset.

6.1 Dataset

We conduct our experiments on a dataset of 29411 trips primarily from Northern Jutland provided by Andersen et al. [4]. The trips are mapmatched to an OSM map of the main road network of Denmark, which covers the road segment categories found in Table 9 in Appendix E. These categories covers 14.33% of the total segments. The trips used therefore cover only the road segments in the OSM map used. We use 60%, and 30% of the trips for the training and test set, respectively; the remaining 10% of the trips are set aside as a validation set to tune model hyperparameters. The number of trips in the training, validation, and test sets can be found in Table 1.

Category	σ^2
"Motorway"	12.09^2
"Motorway Link"	9.51^2
"Expressway"	8.42^2
"Highway"	7.71^2
"Main Road"	6.92^2
"Connecting Road"	6.80^2

TABLE 2: The value of σ^2 depending on road category used in the evaluation of the general framework for cost estimation.

6.2 Baselines

We evaluate against two baseline models.

ExpandingSearch: The ExpandingSearch algorithm described in Appendix D which is an aggregation model. It is designed to predict an expected speed of an edge-interval using 15 minute intervals [5].

LIN-ENG: The linear regression model using the engineered features as described in Appendix C. As with ExpandingSearch, LIN-ENG also predict for edge-intervals using 15 minute intervals [6].

For ExpandingSearch we use the best parameters for the minimum number of trips $m = 10$ and $k = 1$ for the spatially k-nearest search based on evaluation of all combinations of $m \in \{1, 2, 3, 4, 5, 10, 20, 50\}$ and $k \in \{1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 300, 400, 500, 1000\}$.

The LIN-ENG model is trained using the Huber loss function [28] (described in Appendix H) and optimized with Limited-Memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) [29] (described in Appendix K) for a maximum of 5000 iterations with early termination if the loss does not improve more than 10^{-5}. We use $\epsilon = 2.5$ in the Huber loss, and step size $\lambda = 0.0001$ for L-BFGS optimization. The choice of these parameters is based on evaluation on the validation set with all combinations of $\epsilon \in \{2.0, 2.25, 2.5, 2.75, 3.0\}$ and $\lambda \in \{0.01, 0.001, 0.0001, 0.00001\}$.

6.3 Evaluation of the General Framework

In this section, we investigate how the general framework may improve an existing estimation model. We therefore apply the general framework to the LIN-ENG model, s.t. LIN-ENG provides a prior cost estimate $cost_0$, which is subsequently updated based on observations in the training set. We refer to this model as LIN-ENG-OBS.

We choose the expected variance of the mean $\sigma_0^2 = 8.0$ based on evaluation on the validation set with 100% of the training data using different values of $\sigma_0^2 \in \{1^2, 3^2, 5^2, 8^2, 12^2, 15^2\}$ on the validation set. We set the expected observation variance σ^2 depending on road category s.t. σ^2 is the mean variance of all edge-interval pairs in the training set with at least two trips across them. The values of σ^2 depending on category is shown in Table 2.

The results are shown in Fig. 10 using different percentages of the training set for prediction. As seen from the figure, ExpandingSearch keeps improving when given more training data, but has a significant increase in error when just 20% of the training data is used. Although LIN-ENG does not have this error increase, LIN-ENG does not necessarily improve with more data and in fact deteriorates

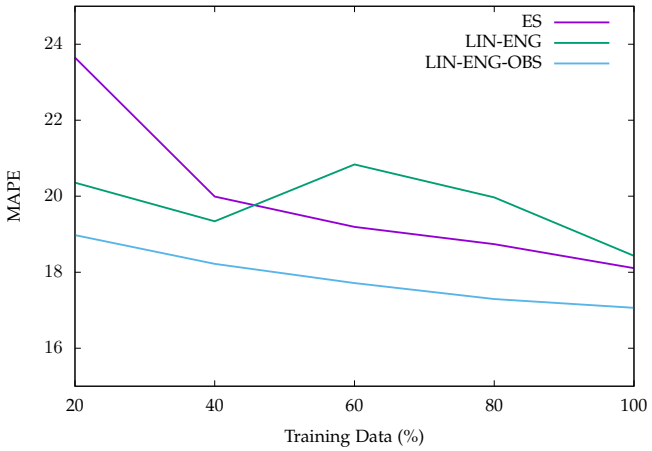


Fig. 10: The MAPE for the LIN-ENG, LIN-ENG-OBS and ExpandingSearch models on the test set when using different percentages of the training data.

when increasing the training data used from 40% to 60% or 80%. This is due to the initial weight function w_0 described Section C.3, which is used during feature construction. w_0 frequently inserts network averages into the contextual features [6] and therefore introduces a lot of noise in the feature representation. Consequently, there are more outliers in the training set which leads to overfitting, as reflected by Fig. 10.

Although we have chosen the variance of the mean σ_0^2 and observation variance σ_0^2 in a very simple manner, the general framework reduces the error of LIN-ENG as shown on Fig. 10. Using the general framework to update the estimations of LIN-ENG has two desirable effects:

1) estimation accuracy is increased s.t. the Mean Absolute Percentage Error (MAPE) is decreased by 7.43% percent (1.37 percentage points) with 100% of the training data, and
2) it smoothens the error curve s.t. more data increases estimation accuracy.

Even with just 20% of the training data, the general framework reduces the error by 6.78% (1.38 percentage points). Given that the update of the general framework is performed in constant time, it may provide a cheap performance increase to existing estimation models.

6.4 Evaluation of the Learned Features

We consider 4 permutations of learned features: the k-Routes and the k-Neighbours corpus generation strategies both with and without the edge in the context to evaluate the effect of adding it. Excluding the edge in the context is equivalent to the CBOW architecture, described in Appendix A, and only feature representations of descriptors is performed. Since no feature representation of edge is learned if the edge is omitted from the context, an edge is instead represented by the learned representation of its descriptor.

Initial experiments with the k-Neighbours corpus generation strategy both with and without the edge as context showed that it performed significantly worse than both

baselines on the validation set and exhibited very high variance due to the random sampling of the neighborhood. The experiments in this section therefore includes only the k-Routes corpus generation strategy with and without the source edge context.

6.4.1 Models

We consider two linear models trained using learned feature representations: LIN-EMB and LIN-EMB-EDGE. LIN-EMB represents an edge by the learned feature representation of its descriptor. LIN-EMB-EDGE represents an edge using the learned feature representation of the edge.

Like LIN-ENG, both LIN-EMB and LIN-EMB-EDGE are trained using the Huber loss function [28] (see Appendix H) and optimized with L-BFGS [29] (see Appendix K). We use early termination if the loss does not improve more than 10^{-5} during L-BFGS optimization.

All different combinations of parameters considered for tuning of both LIN-EMB and LIN-EMB can be found in Appendix I. Here we describe the parameters chosen based on evaluation on the validation set using 100% of the training set.

LIN-EMB: The feature representations of LIN-EMB is learned by using stochastich gradient descent to update the weights of a CBOW network, using the k-Routes corpus generation strategy. For feature learning, we use parameters $k = 3$, 100 iterations, learning rate 0.001, a context size $c = 1$, and a hidden layer size of 400. For subsequent linear regression we use the Huber loss function with $\epsilon = 1.1$, and 5000 iterations and step size of 0.0001 for L-BFGS optimization.

LIN-EMB-EDGE: The feature representations of LIN-EMB-EDGE is learned by using stochastich gradient descent to update the weights the network, similar to the feature representation used in LIN-EMB, but now for a PVDM network. Again, the k-Routes corpus generation strategy is used. For feature learning, we use parameters $k = 1, 25$ iterations, learning rate 0.1, context size parameter $c = 3$, and a hidden layer size of 400. For subsequent linear regression we again use the Huber loss function with $\epsilon = 1.1$, and 5000 iterations and step size of 0.0001 for L-BFGS optimization.

6.4.2 Results

Fig. 11 shows a comparison of the engineered and learned features when used with a linear estimation model. The figure shows that representing an edge by the learned feature representation of its descriptor (LIN-EMB) is superior to using the feature representation of the edge (LIN-EMB-EDGE) in a linear model. The edge source context does not sufficiently describe the differences between each edge and is in fact inferior to the engineered features. LIN-EMB is superior to LIN-ENG, however, and thus the learned feature representation of edge descriptors is superior to the engineered features based on domain knowledge. In addition, the learned feature representations are very robust and barely decrease with less training data.

We investigated if the concatenation of the feature representations could potentially decrease error in a linear model. The model yielded a MAPE of 17.57% on the validation set when using 100% of the training set, which is approximately

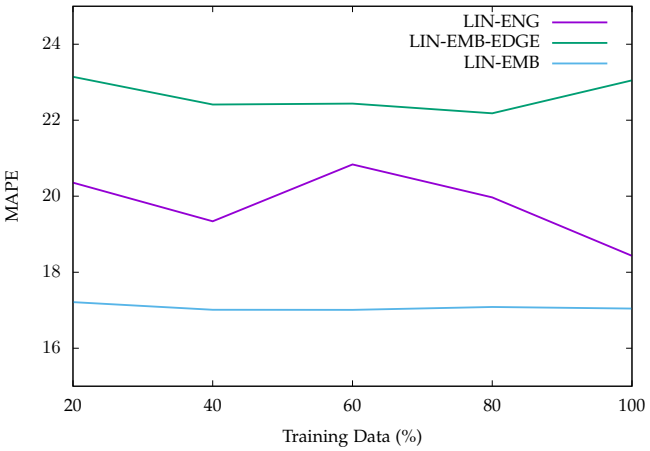


Fig. 11: A comparison of the engineered features of LIN-ENG against the learned features with and without the edge context.

the same as LIN-EMB on the validation set. The concatenation therefore does not appear to decrease error significantly from this initial experiment.

We speculate that the cause of the LIN-EMB-EDGE poor performance is due to the simple linear model used. We therefore performed an explorative experiment where a neural network is trained to estimate edge-interval costs using a single hidden layer of size 302. The median MAPE of 5 runs on the test set is 17.24% when using 100% of the training set and 24.47% when using just 20% of the training set. Although it approximates the error of LIN-EMB, the model is less robust to data sparsity. Summing or averaging the edge descriptor and edge representations in subsequent experiments did not improve the error significantly.

6.5 Updating the Learned Features using the General Framework

In this section we evaluate the effect of combining learned features with the general framework by making the LIN-EMB model provide a prior cost estimate to the framework. We refer to this model as LIN-EMB-OBS.

We choose $\sigma_0^2 = 4.0$ based on evaluation on the validation set with 100% of the training data using different values of $\sigma_0^2 \in \{1^2, 3^2, 4^2, 5^2, 8^2, 12^2, 15^2\}$. As with LIN-ENG-OBS in Section 6.3, we set σ^2 depending on road category according to Table 2.

The results are shown in Fig. 12. Using the general framework with LIN-EMB decreases MAPE at all percentages of training data, and by 3.73% at 100% of the training data (0.63 percentage points). Additionally, LIN-EMB-OBS performs, relative to LIN-EMB, better with more training data.

6.6 In-Depth Model Analysis

In this section we give a more in-depth analysis of the ExpandingSearch, LIN-ENG-OBS, and LIN-EMB-OBS models. Fig. 13 gives an overview of the estimation error of all models for different training data percentages. In the following we analyse how the ExpandingSearch, LIN-ENG-OBS, and LIN-EMB-OBS models perform depending on trip duration, trip category and at different times of day.

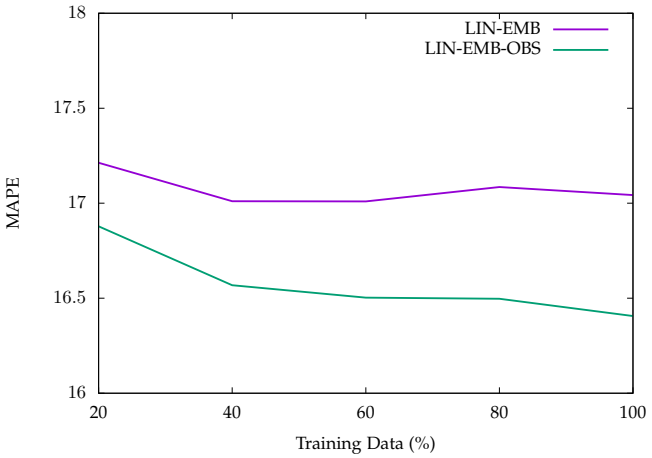


Fig. 12: A comparison of the learned features with and without the general framework.

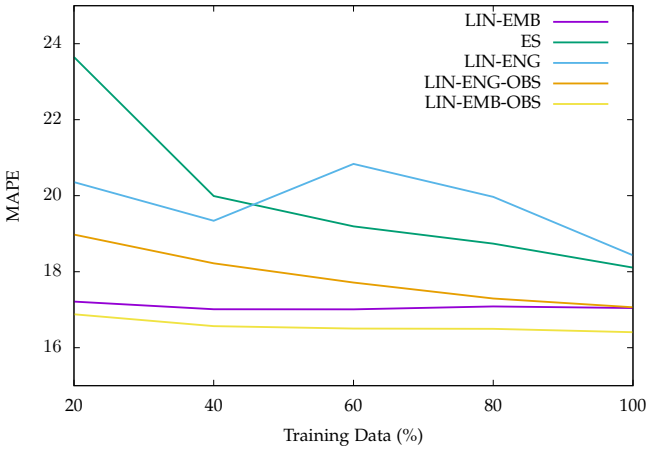


Fig. 13: A comparison of all models.

6.6.1 Trip Categorization

We divide the trips into four mutually exclusive categories: City, Municipality, Region, and Country.

City: City trips start and end *within the same city.*

Municipality: Trips, that are not city trips, which start and end *within the same municipality.*

Region: Trips, that are not municipality trips, which start and end *within the same Danish administrative region.*

Country: Trips that do not belong to any of the other categories, and therefore start and end *within the country of Denmark.*

The distribution of trips in the test set across the trip categories can be seen in Table 3. As the table shows, more than half of the trips are municipaliy trips.

6.6.2 Error by Trip Category

The purpose of this section is to give a broad overview of how each model performs depending on trip category and time of day. A more fine-grained analysis is presented in Section 6.6.3.

Table 4 show a comparison of ExpandingSearch, LIN-ENG-OBS, and LIN-EMB-OBS for the different trip categories for all times of the day. LIN-EMB-OBS achieve a lower MAPE in the *city* and *municipality* trips categories, whereas LIN-ENG-OBS has a lower MAPE in the *country* and *region* categories. The greatest difference between the three models can be seen in the *municipality* category, where LIN-EMB-OBS achieves a MAPE of 17.32% which is 1.16 percentage points lower than LIN-ENG-OBS and 2.66 percentage points lower than ExpandingSearch.

We discretize the time of day into 5 intervals to examine how the MAPE for trips in each category is related to the time of day they are begun. The intervals [7:30; 8:15) and [15:30; 16:30) are peak hour intervals, and the intervals [8:15; 15:30), [16:30; 22:00), and [22:00; 7:30) are off-peak hours. These intervals are reasonable time discretizations given the area we have data from, but more detailed plots of error as a function of trip start time can be found in .

The trend from Table 4 continues within each time interval; LIN-EMB-OBS is best at city and municipality trips and LIN-ENG-OBS at region and country trips, and the largest difference between the models occurs for municipality trips. As an example, we show the MAPE for each category in the interval [7:30; 8:15) in Table 5 the LIN-EMB-OBS. There are two exceptions to this trend:

- LIN-EMB-OBS and LIN-ENG-OBS score the same in country category between [15:30; 16:30) as shown in Table 6, and
- ExpandingSearch scores the highest of all the models in the city and region categories for interval [22:00; 07:30).

The latter is caused by a skew in the training data. As illustrated by Fig. 14, the majority of the trips in the training set begin in the interval [4:30; 22:30). LIN-ENG-OBS and LIN-EMB-OBS are therefore better fitted to this interval. Contrary to the linear models, ExpandingSearch relies on aggregation of observations and is therefore not affected by skews in the training data. In addition, there is less congestion during the night than mid day and therefore defaulting to use the speed limit for estimation may be reasonably accurate. Tables for [8:15; 15:30) and [16:30; 22:00) can be found in Appendix G.

Category	*No. of trips*
City	1845 (21.0%)
Municipality	4838 (55.1%)
Region	1654 (18.9%)
Country	436 (5.0%)
Total	8773 (100%)

TABLE 3: Distribution of trips in the test set across the trip categories.

	Category			
Model	*City*	*Municipality*	*Region*	*Country*
ExpandingSearch	21.31	19.98	11.35	9.30
LIN-ENG-OBS	21.13	18.48	**10.60**	**8.50**
LIN-EMB-OBS	**20.66**	**17.32**	10.89	9.04

TABLE 4: MAPE of ExpandingSearch, LIN-ENG-OBS and LIN-EMB-OBS for all trips categories.

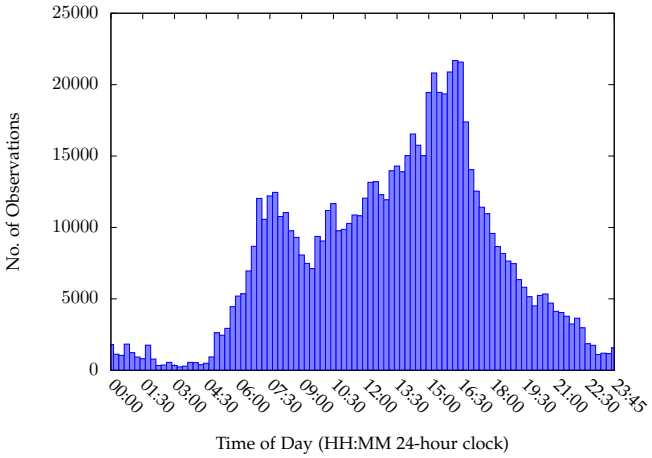


Fig. 14: No. of observations as a function of the time of the day with a time granularity of 15 min.

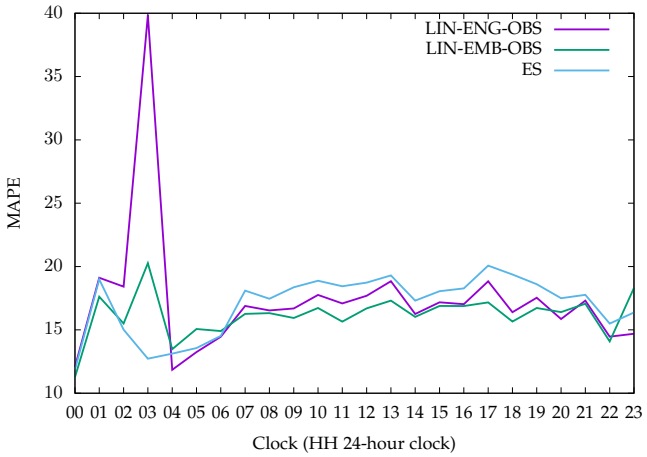


Fig. 15: MAPE as a function of the time of a day, for all trips. The MAPE is calculated for each 1 hour interval during the day.

	Category			
Model	*City*	*Municipality*	*Region*	*Country*
ExpandingSearch	23.90	21.19	11.53	6.96
LIN-ENG-OBS	23.16	19.49	**11.11**	**6.11**
LIN-EMB-OBS	**22.66**	**19.10**	11.72	7.08

TABLE 5: MAPE of ExpandingSearch, LIN-ENG-OBS and LIN-EMB-OBS for all trips categories between 07:30-08:15.

6.6.3 Error by Time of Day

Fig. 15 illustrates the MAPE of each model for each hour of the day. LIN-ENG-OBS and LIN-EMB-OBS has between [2:00; 4:00) compared to ExpandingSearch due to the skew in the training data discussed in Section 6.6.2. As Fig. 15 illustrates, LIN-EMB-OBS performs the best of the models at the hours with most trip data i.e. between [6:00; 22:00). The same tendency can also be seen in Fig. 16. The figure illustrates the MAPE of each trip for each model as a function of the trips start time of the day. LIN-EMB-OBS generally estimates less varying than the other 2 models between [6:00; 22:00). Another version of the graph in Fig. 16 without scaling of the y-axis can be found in Appendix G.

	Category			
Model	*City*	*Municipality*	*Region*	*Country*
ExpandingSearch	20.53	21.40	9.86	9.08
LIN-ENG-OBS	20.47	19.87	**9.23**	**8.23**
LIN-EMB-OBS	**20.21**	**19.31**	9.66	**8.23**

TABLE 6: MAPE of ExpandingSearch, LIN-ENG-OBS and LIN-EMB-OBS for all trips categories between 15:30-16:30.

	Category			
Model	*City*	*Municipality*	*Region*	*Country*
ExpandingSearch	**19.58**	15.56	**11.19**	8.58
LIN-ENG-OBS	21.04	15.24	11.24	**7.98**
LIN-EMB-OBS	20.28	**14.95**	11.45	8.43

TABLE 7: MAPE of ExpandingSearch, LIN-ENG-OBS and LIN-EMB-OBS for all trip categories between 22:00-07:30.

6.6.4 Error by Trip Duration

Fig. 17 illustrates MAPE as a function of municipality trip's duration for the three models LIN-EMB-OBS, LIN-ENG-OBS and ExpandingSearch. LIN-EMB-OBS estimates more accurate at shorter duration trips than both of the baselines and LIN-ENG-OBS more accurate than ExpandingSearch. LIN-EMB-OBS and LIN-ENG-OBS are both comparable at trips longer than 2 minutes. Observe, that ExpandingSearch's estimates are in general more varying compared to the other two models. Plots for the remaining categories can be found in Appendix G.

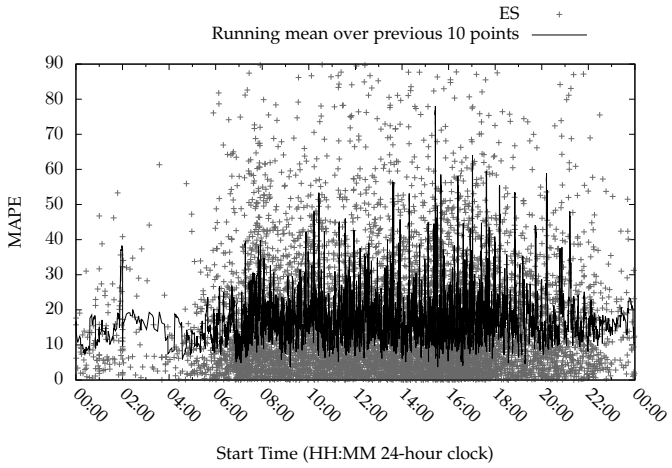
6.6.5 Summary

LIN-EMB-OBS displays superior performance at all training data percentages when using all the test data. In our analysis of the models error for specific trip categories independent of time of day, LIN-EMB-OBS achieves the lowest error in the city and municipality categories and LIN-ENG-OBS in the region and country categories. ExpandingSearch performs better than the LIN-ENG-OBS and LIN-EMB-OBS models during the night when the amount of data is low, despite being an aggregation model. This is likely attributed to less congestion during the night than mid day and therefore defaulting to use the speed limit as the estimate may be reasonably accurate. LIN-EMB-OBS achieved the highest MAPE in the two largest categories: city and municipality. LIN-ENG-OBS achieved the highest MAPE in the two remaining categories: region and country.

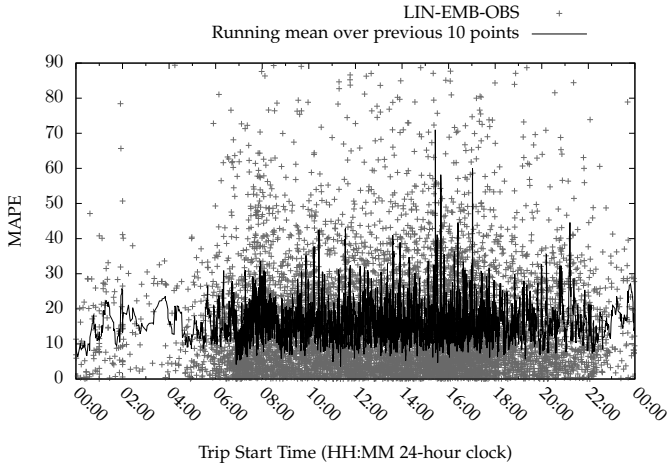
6.7 Computation Time Comparison

In addition to the error of the models, we investigate the computation time of the different models on 20% and 100% of the training data.

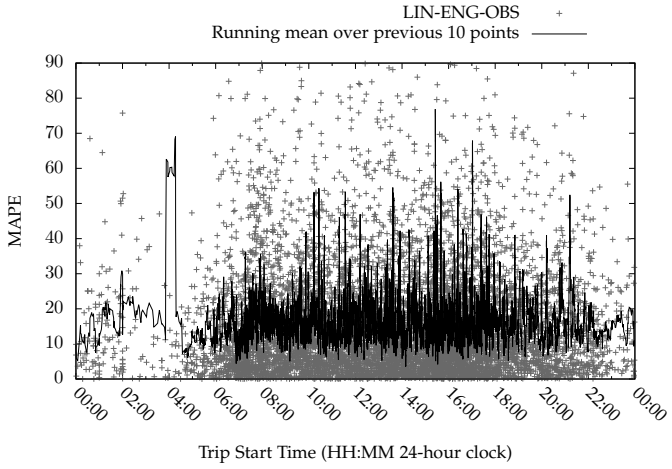
The estimation update incurred by the general framework has constant cost and therefore there is no significant difference in computation time for LIN-ENG and LIN-ENG-OBS, but the computation time displays high variance since the number of iterations required for convergence during



(a) ExpandingSearch

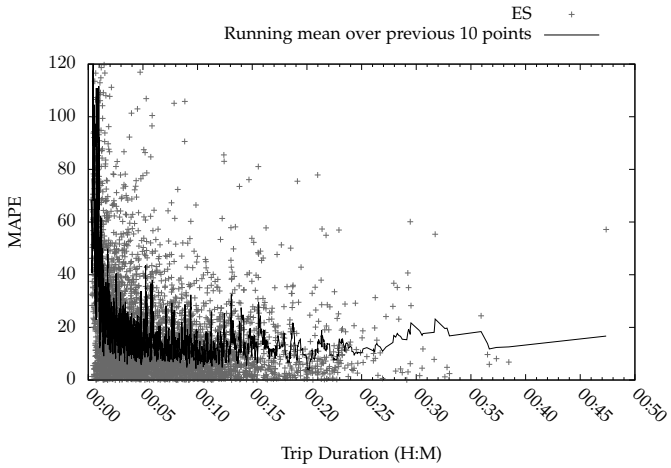


(b) LIN-EMB-OBS

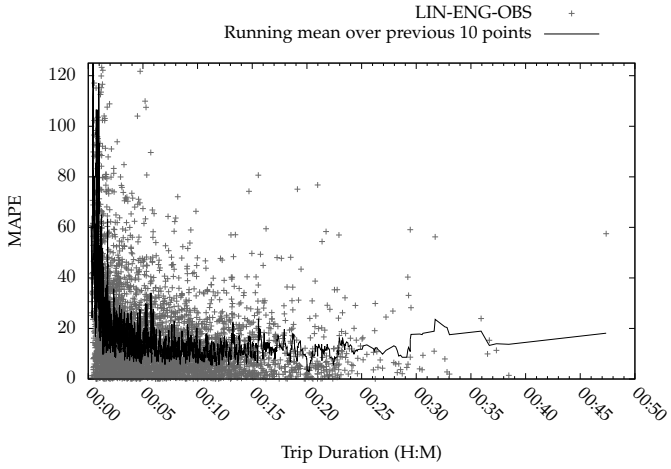


(c) LIN-ENG-OBS

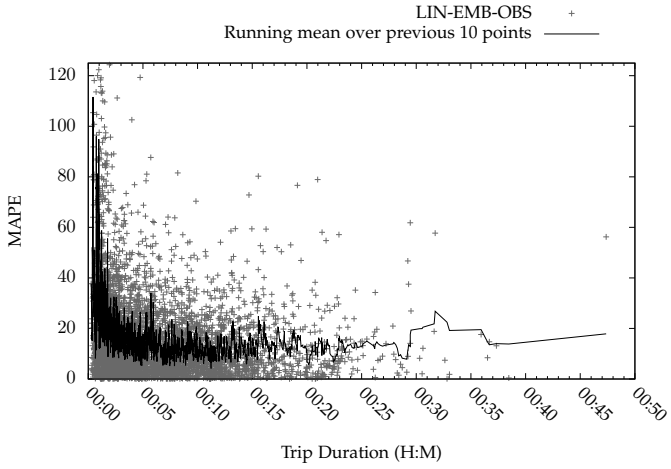
Fig. 16: MAPE of each trip for the three models ExpandingSearch, LIN-EMB-OBS and LIN-ENG-OBS depending on trips start time, with y-axis (MAPE) maxed at 90.



(a) ExpandingSearch.



(b) LIN-EMB-OBS.



(c) LIN-EMB-OBS.

Fig. 17: MAPE as a function of the duration of municipality trips for the three models ExpandingSearch, LIN-EMB-OBS and LIN-ENG-OBS.

L-BFGS optimization varies. We therefore report the mean and standard deviation for 10 runs of LIN-ENG-OBS.

The spatial k-nearest search in the ExpandingSearch model is delegated to a remote spatio-temporal database for ease of implementation. An edge prediction using k-nearest takes an arithmetic mean of 0.22 seconds. The arithmetic mean of edges per trip in our test set is 43 with a standard deviation of 46.79. Worst case, k-nearest needs to run two times per edge, that is, for each predicate and therefore takes $(0.22 * 46.49) * 2 = 20.58$ seconds for the average trip. This incurs a significant overhead in network transfer and therefore the computation time of the spatial k-nearest search is not included in prediction time.

We measure the computation time required to perform various parts of the experiments process for the various models when using 20% and 100% of the training data. Preprocessing includes preparing the dataset and transforming edge intervals to features. For LIN-EMB it also includes training the embedding.

6.7.1 Results

As shown in Table 8, the ExpandingSearch algorithm has both the least amount of preprocessing time and training time. Preprocessing consists of constructing the in-memory graph representation for prediction later on and the model has no training time since it is an aggregation models. Almost all computation time is therefore spent on prediction, which increases by 41.06% when decreasing the training data from 100% to 20%.

The LIN-ENG and LIN-EMB-OBS are both parametric models and therefore require more time spent on preprocessing and training, as evident from Table 8. In both cases, the increase in preprocessing time is due to the construction of the feature representations of the training set. Curiously, both the preprocessing and feature learning of LIN-EMB takes longer on average and displays larger variance when using 20% of the training data as opposed to 100%. This is contrary to LIN-ENG-OBS where the preprocessing time increases with the amount of data, as we would expect, and the variance is largely the same. We attribute this discrepancy to other processes being run in parallel, when measuring the preprocessing time of LIN-EMB using 20% of the training data thus affecting the measurement in CPU time, e.g. due to more CPU cache misses.

As shown in Table 8, LIN-ENG-OBS and LIN-EMB both spent considerably less time on prediction than ExpandingSearch, but LIN-EMB is also considerably faster than LIN-ENG-OBS. We attribute this difference in prediction time to the on-the-fly feature representation construction required when necessary to when summing up the predictions. In the case of LIN-ENG, the constructing the features requires several lookups into an in-memory representation of the road network for on-the-fly feature construction, whereas the learned feature representation requires just two lookups, one for to construct the time feature and one to retrieve the learned embedding. The difference in prediction time of LIN-ENG-OBS using 20% and 100% of the training data is due to the way zero observations are represented and handled in our experimental framework.

6.7.2 Summary

Both the LIN-ENG-OBS and LIN-EMB models require more time spent on preprocessing and subsequent training. Once trained these models can be stored, however, and thus the time required to answer routing requests is expressed by their prediction time. In this regard, the ExpandingSearch is inferior to LIN-ENG-OBS and LIN-EMB, since it is several times slower at prediction time. In addition, the prediction time of ExpandingSearch increases as the size of the training set decreases, since the first search strategies are more likely to fail. Although Table 8 shows that LIN-ENG-OBS has the shortest total time, most of the total time spent by LIN-EMB is spent on preprocessing, learning, and training which can be done once. LIN-EMB is several times faster than LIN-ENG-OBS at prediction time, but this is likely attributed to the implementation of the on-the-fly feature construction process in our experimental framework , since it could in principle be cached in the same manner that the learned feature presentations used in LIN-EMB are cached.

7 Structural Similarity of Edges

In addition to the travel time estimation experiment described in Section 6, we investigate the capability of the learned feature to capture structural similarity. We do this by performing K-means clustering [30] (see Appendix L) on the learned feature representations of edges. The hypothesis is that edges which have similar roles in the road network will appear in the same cluster.

We evaluate the clustering based on both cluster validation metrics and visual inspection of the clusters mapped to the OSM map.

7.1 Dataset

The travel time estimation experiment described in Section 6 is limited to trips on the most important road segment categories, described in Section 6.1, and due to memory constraints, feature learning is also based on just these road segments. In then structural similarity we remove the restriction on road segment categories, but restrict the road network to Northern Jutland, Denmark, which includes the categories found in Table 10 in Appendix F. We learning feature representations of edges and their descriptors based on the OSM road network of Northern Jutland.

7.2 Cluster Validation Metrics

Cluster validation metrics are measures for the quality of a cluster. The Calinski-Harabasz Index (CHI) [31] is an internal cluster validation metric which has been shown to be superior for choosing the number of clusters across different datasets compared to other interval validation metrics [32]. We therefore choose the number of clusters $K \in \{8, 10, 12, 14, 16, 18, 20, 25, 30, 35, 40\}$ for the K-means clustering which yields the high CHI.

We use CHI to choose the number of clusters, where CHI measures the ratio between *cluster cohesion* and *cluster separation* where a higher CHI is better [31]. Cluster cohesion is measured in terms of Within-Cluster Sum of Squares (WCSS) and cluster separation in terms of Between-Cluster

Model	*Training Data*	*Time Spent (seconds)*			
		Preprocessing (Feature Learning)	Training	Predicting	Total
ExpandingSearch	20%	**2.54**	**0**	13510.78	13513.32
	100%	**13.89**	**0**	7962.98	7976.87
LIN-ENG-OBS	20%	302.77 ± 9.66	2086.04 ± 530.80	1227.71 ± 30.68	$\mathbf{3616.53 \pm 547.16}$
	100%	960.73 ± 12.97	2121.24 ± 951.09	991.63 ± 12.76	$\mathbf{4073.60 \pm 960.41}$
LIN-EMB	20%	669.84 ± 170.73 (619.84)	4001.95 ± 126.84	$\mathbf{166.86 \pm 4.67}$	4838.67 ± 346.14
	100%	521.55 ± 10.49 (354.93)	16634.71 ± 10.49	$\mathbf{167.84 \pm 3.64}$	17324.11 ± 24.64

TABLE 8: Computation time in seconds for different parts of the prediction process using 20% and 100% of the training data.

Sum of Squares (BCSS). We therefore first define these metrics.

Given a set of K clusters $\mathscr{C} = \{C_1, \ldots, C_K\}$, the WCSS is defined as [31]

$$WCSS = \sum_{k=1}^{K} \sum_{\mathbf{x}_k \in C_k} ||\mathbf{x}_k - \mu_k||^2 \tag{18}$$

where $||$ is euclidean distance, $\mathbf{x}_k \in C_k$ is the learned feature representation of an edge $e_k \in E$ and $\mu_k = \frac{1}{|C_k|} \sum_{\mathbf{x}_k \in C_k} \mathbf{x}_k$ is the centroid of cluster C_k. WCSS is related to variance, since it is a weighted sum of cluster variance.

The BCSS is defined as [31]

$$BCSS = \sum_{k=1}^{K} |C_k| \cdot ||\mu_k - \mu||^2 \tag{19}$$

where $\mu = \frac{1}{|C|} \sum_{\mathbf{x} \in C} \mathbf{x}$ is the mean of all learned feature representations $x \in C = \bigcup_{k=1}^{K} C_k$ of edges $e \in E$.

Finally, the CHI is defined in terms of WCSS and BCSS as [31]

$$CHI = \frac{BCSS}{K-1} \Big/ \frac{WCSS}{N-K} \tag{20}$$

where $N = |\bigcup_{k=1}^{K} C_k|$.

7.3 Feature Representations

The number of distinct descriptors (214) and edges (215009) in the dataset is similar to that of the road network used in Section 6 (225 descriptors and 204778 edges). We therefore reuse the parameters (Section I.2) for learning the representations using the k-Routes corpus generation strategy and including the source edge contex.

We consider 3 feature representations of edges for clustering.

DESCRIPTOR: The feature representation of an edge is the learned feature representation of its descriptor. This naturally implies that edges can be described by their descriptor alone and that edges with the same descriptor are identical.

EDGE: The feature representation of an edge is its distinct feature representation.

EDGE+DESCRIPTOR: The feature representation of an edge is the sum of its EDGE and DESCRIPTOR representation. Similar to the DESCRIPTOR representation, the EDGE+DESCRIPTOR representation implies that edges are similar if they have the same descriptor, but can be differentiated due the contribution of the EDGE representation.

7.4 Analysis

The EDGE representation alone creates clusters that are not cohesive or well-separated, an although the clusters appear to capture some meaning, they are very noisy. We therefore forego further analysis of clustering using the EDGE representation.

7.4.1 *Cluster Quality*

Fig. 18 shows the CHI depending on the number of clusters. The CHI when using just the learned feature representation of a descriptor is erratic and appears to decrease as the number of clusters increase. When using the concatenation of the feature representations the CHI smoothly increases as the number of clusters increase. This suggests that using the concatenation can generate higher quality clusters than simply the feature representation of the descriptor.

Plots for cluster cohesion (WCSS) and separation (BCSS) can be seen on Fig. 35 in Appendix M.

7.4.2 *Cluster Visualisation*

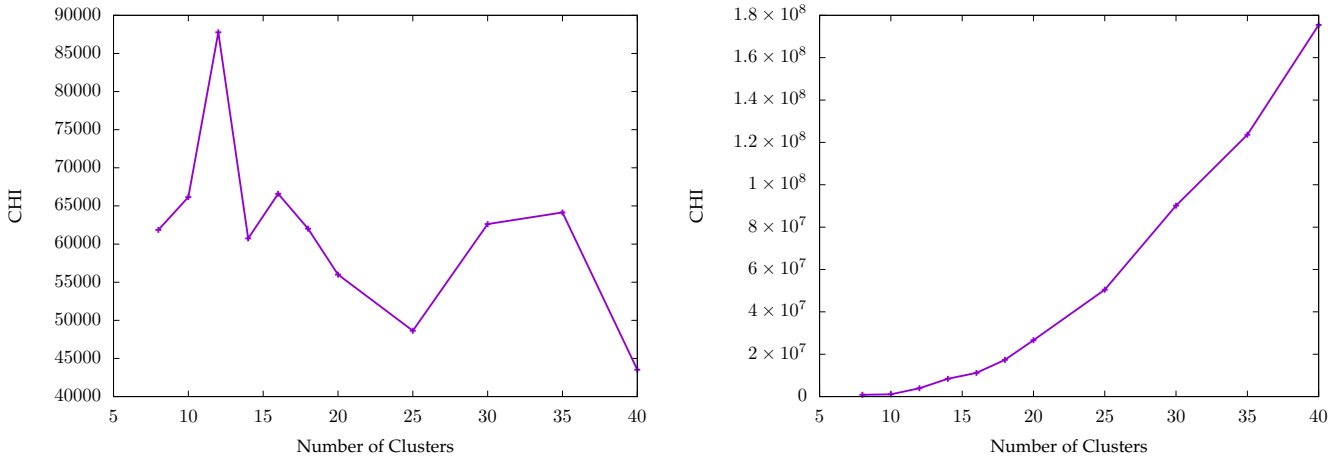
We choose $K = 35$ for clustering with DESCRIPTOR representations and $K = 40$ for clustering EDGE+DESCRIPTOR representations based on the CHI score shown in Fig. 18. We refer to this clustering as EDGE+DESCRIPTOR-40. Although $K = 12$ has the highest CHI for clustering with DESCRIPTOR representations, no meaningful clusters were found during visual inspection. We therefore choose $K = 35$ instead, the best cluster separation (see Appendix M), and has more meaningful clusters upon visual inspection than $K = 12$. We refer to this clustering as DESCRIPTOR-35.

Each of the clusters in DESCRIPTOR-35 and EDGE+DESCRIPTOR-40 was inspected and we found similar clusters using both representations. The following categories of road segments clusters in one or few clusters, we associate a color with each of these clusters:

- **Black** motorways
- **Blue** highways connecting provinces on the island of Vendsyssel
- **Red** the main roads through the city of Aalborg
- **Yellow** roads going through city centers of smaller towns only found by EDGE+DESCRIPTOR-40
- **Cyan** residential road segments

7.4.2.1 *Alborg*

Fig. 19 visualizes the clustering of road segments in the center of Aalborg along with the E45 motorway north of the fjord. The main roads through the city center, e.g. Vesterbro,



(a) Descriptor

(b) Edge + Descriptor

Fig. 18: The CHI depending on the number of clusters using (a) the learned feature representation of an edge's descriptor (b) the concatenation of the learned feature representations of an edge and its descriptor.

are in the same cluster as the motorways for DESCRIPTOR-35 and are therefore highlighted with black. In contrast, EDGE+DESCRIPTOR-40 puts these into separate clusters, with the exception of motorway segments close to cities. The clusters appear virtually identical otherwise, with residential road segments giving access to houses being in separate clusters. Although most of the residential road segments (**cyan**) are placed within a single cluster, a few are contained in other clusters. In total, they are spread across 2 clusters for DESCRIPTOR-35 and 3 clusters for EDGE+DESCRIPTOR-40 although there is no readily apparent difference between the road segments.

7.4.2.2 *Vendsyssel*

Fig. 20 shows the clustering on most of Vendsyssel. Notice again that motorway segments close to cities are grouped with the main roads through the center of Aalborg, such as the last part of the E45 motorway leading into Frederikshavn on the eastern coast and most of E39 north of Hjørring (at the center top of the figure) leading up to Hirtshals (not shown). The **blue** highways far from coasts are generally within one large cluster for both DESCRIPTOR-35 and EDGE+DESCRIPTOR-40 with few exceptions. Highways close to coasts are in different, smaller clusters: 1 other for DESCRIPTOR-35 and 3 others for EDGE+DESCRIPTOR-40. DESCRIPTOR-35 clusters some of the highways with motorway segments, such as the highway connecting Løkken to Hjørring on the western coast and the highway going southwest out of the illustration from Aabybro.

7.4.2.3 *Small Towns*

The EDGE+DESCRIPTOR-40 clustering identifies a cluster unlike any found in DESCRIPTOR-35. The Yellow cluster shown on Fig. 21 contains roads which go through smaller towns, giving acess to residential areas. These road segments therefore appear to have the same role in the road network. As can be seen on the figure, road segments belonging to the same cluster can also be found in Aalborg where they play a similar role. More figures can be found in Fig. 40 in Appendix N.

7.4.3 *Summary*

The EDGE+DESCRIPTOR representation appears to capture more meaningful clusters than DESCRIPTOR both in terms of CHI and based on visual inspection where main roads going through city centers are distinct from motorways connecting larger cities. We primarily attribute this to the DESCRIPTOR representation rather than the noisy EDGE representation, which indicates that edges with the same descriptor are closely related and that this relation is not accounted for in the EDGE representation.

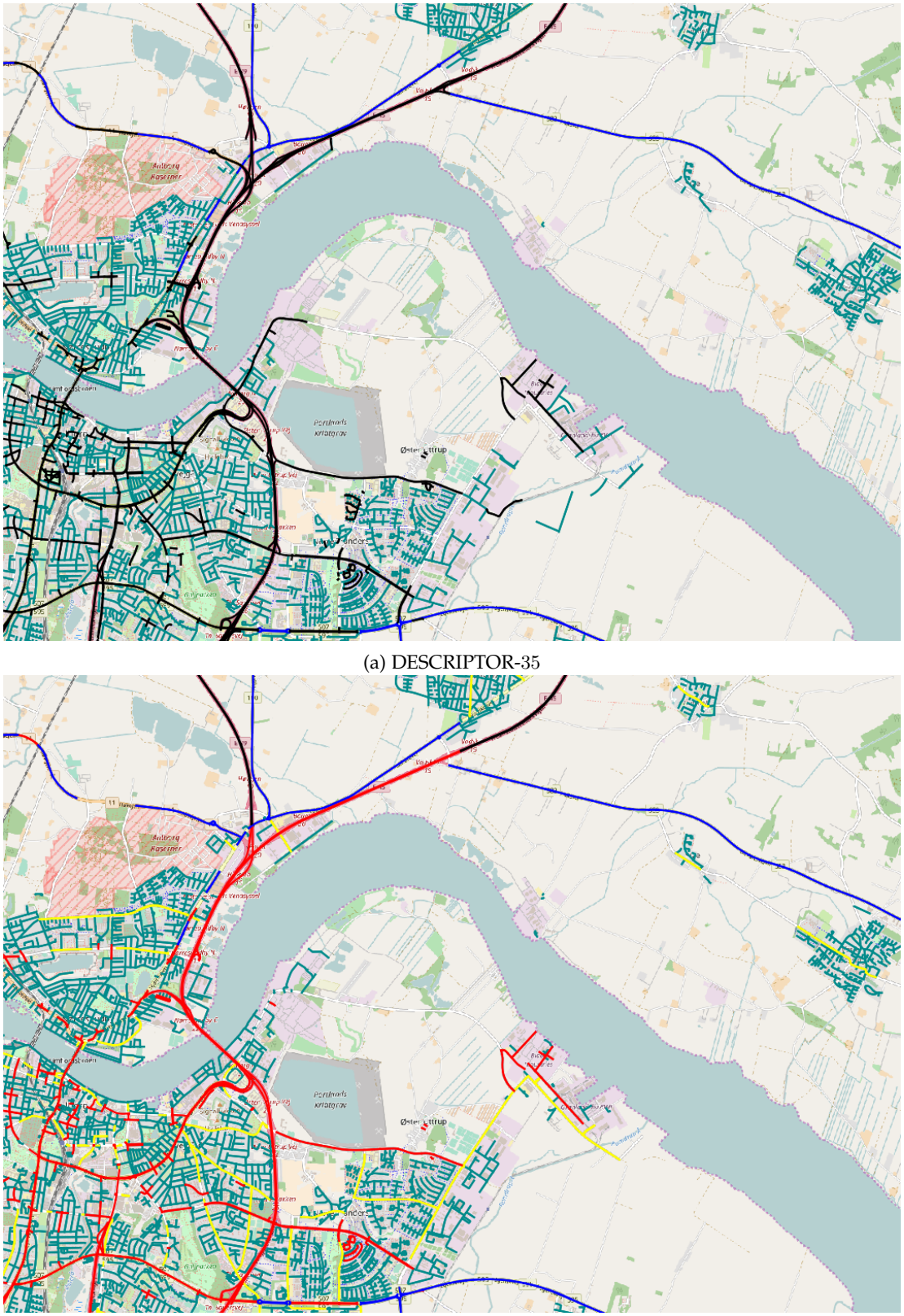
The DESCRIPTOR representation in particular appears to reveal a pitfall of our feature learning approach since we would not intuitively expect motorways, highways, and main roads to be similar. The EDGE-DESCRIPTOR representation displays the problem to a lesser extent, but still clusters motorway segments and main roads through cities together. The motorways close to cities co-occur with the same descriptors as main roads in cities and thus yields similar feature representations.

8 Conclusion

Routing algorithms rely on weighted graph representations of road networks, but finding accurate weights for edges or paths is not trivial. A number of weight assignment models have therefore been proposed which can broadly be categorised as aggregation models, which use aggregation of GPS observations for estimation, and parametric models, which fit a function to the feature representations of edges and paths. This paper makes two main contributions which can be used to improve estimation accuracy of both parametric and aggregation models: a general cost estimation framework and road2vec, a framework for learning feature representations of road segments in road networks.

8.1 General Cost Estimation Framework

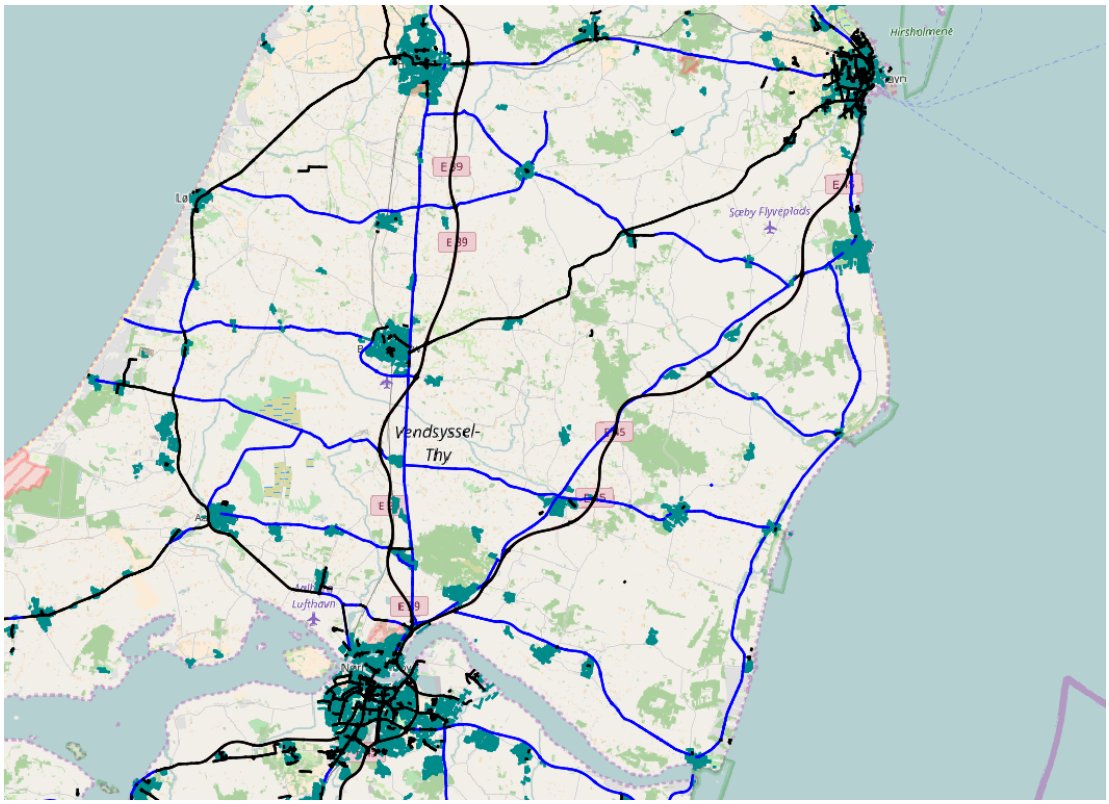
Our general cost estimation framework is based in Bayesian statistics and updates a prior cost estimate of an edge-interval pair provided by a cost estimation model. The



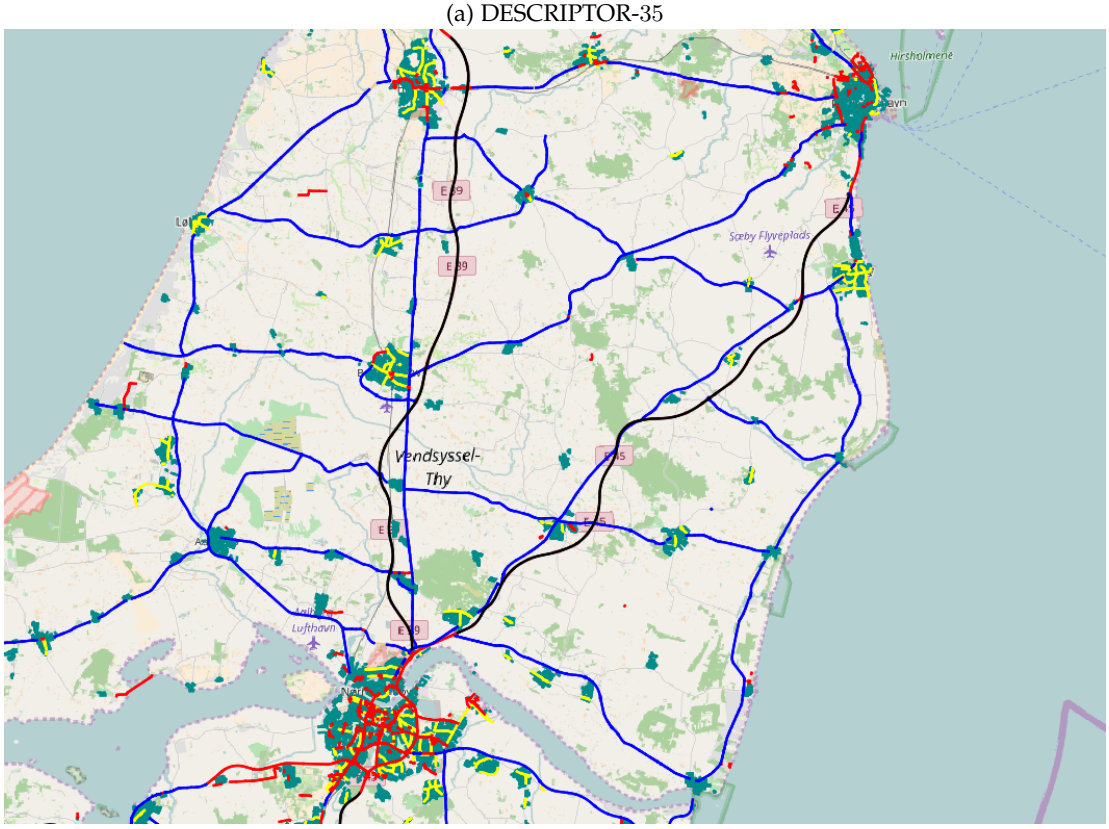
(a) DESCRIPTOR-35

(b) EDGE+DESCRIPTOR-40

Fig. 19: Cluster visualization of the center of Aalborg and the E45 motorway going out from the city using (a) DESCRIPTOR-35 and (b) EDGE+DESCRIPTOR-40.

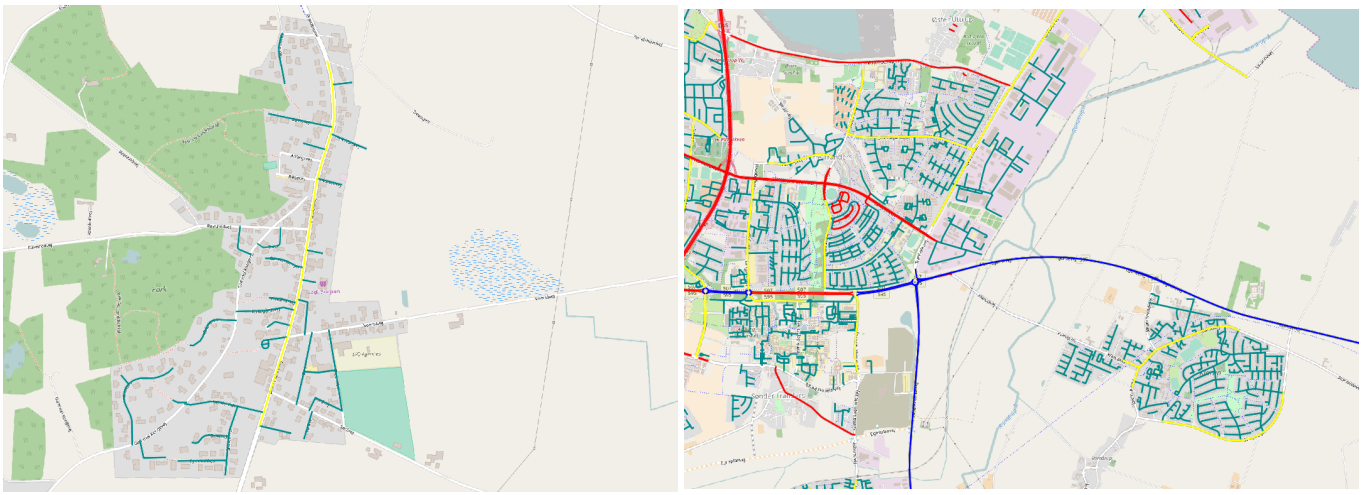


(a) DESCRIPTOR-35



(b) EDGE+DESCRIPTOR-40

Fig. 20: Cluster visualization of most of Vendsyssel along with the city of Aalborg south of the fjord using (a) DESCRIPTOR-35 and (b) EDGE+DESCRIPTOR-40.



(a) Voerså (b) Aalborg

Fig. 21: Cluster visualization of (a) the small town of Voerså and (b) the eastern outskirts of Aalborg.

framework uses any available observations of an edge-interval pair to update a cost estimate of the edge-interval pair provided by any given cost estimation model. The observations are assumed to follow a Gaussian distribution, and the update can be performed in constant time. The framework allows parametric cost estimation models to inherit a property of aggregation models: the updated cost estimate of a parametric model approaches the population mean as the number of observations goes towards infinity.

We evaluate the cost estimation framework on a travel time estimation task where a prior estimate of an edge-interval pair is provided by a linear cost estimation model. The cost estimation framework decrease the MAPE by 7.43% (1.37 percentage points) using 100% of the training set. Much of the improvement is gained with just 20% of the training data, however, which decreases MAPE by 6.78% (1.38 percentage points). Although we have only considered updates of cost estimates for edge-interval pairs in this paper, it can in principle be extended to cost estimation of paths in general, and continuous time by sampling observations within some interval surrounding a point in time.

8.2 road2vec

In addition to our general cost estimation framework, we present road2vec, an adaption feature learning techniques from the domain of language modelling, which learn representations of edges and edge descriptors (a representation of an edge's labels in our experiments). We replace the engineered features of linear cost estimation baseline model with feature representations learned using road2vec and evaluate them both on a travel time estimation task. Using the learned feature representations reduces the MAPE by 7.54% (1.39 percentage points). In addition, the learned feature representations are very robust; reducing the training data from 100% to 20% increases the MAPE by just 1.0% (0.17 percentage points). Using the learned feature representations with a linear model in conjunction with the general cost estimation framework yields the most accurate model with a MAPE of 16.41%: a 9.39% (1.70 percentage points) lower MAPE than a baseline aggregation model and a 10.99% (2.03 percentage points) lower MAPE than a baseline parametric model. After tuning, the learned feature representations for the model are learned in under 10 minutes, without any need for domain knowledge.

We also investigate the learned feature representations capability to capture structural similarity in road networks by using K-means clustering to find meaningful clusters on the road network of Northern Jutland, Denmark. The quality of the clusters are measured using Calinski-Harabasz Index (CHI) and produce both coherent and well-separated clusters. Visualizing the clusters on the road network of Northern Jutland revealed several clusters containing structurally similar road segments, such as a cluster for important main roads in large towns, a cluster containing motorway segments, and a cluster containing roads through the town center of smaller towns. The clusters still contains some noise, however: motorway segments near cities are clustered with segments of important main roads in large towns even though they display no apparent similarity. This is likely because spatially close edges will generate similar neighborhoods and thus be given similar feature representations.

The learned feature representation of edges alone proved inadequate in both the travel time estimation task and structural similarity task. The learned feature representations of edges are evaluated using a non-linear model. Although the non-linear model makes the feature representations of edges competitive with the feature representation of the edge descriptors using 100% of the dataset, the feature representation is not as robust to data sparsity. In the travel time estimation task using the learned feature representation of an edge descriptor as the feature representation is superior, and their concatenation does not appear to lead to a decrease in estimation error with a linear model.

Representing an edge as a combination of its learned feature representation and the feature representation of its edge descriptor proved superior in the structural similarity task. The combined representation yielded better cluster cohesion and separation than using either the edge representation

or edge descriptor representation in isolation and found more meaningful clusters. The closest competitor is the edge descriptor representation. This suggests that the inherent similarity between edge descriptors is not captured by the learned edge representation and its expressivity in isolation is therefore limited.

9 Future Work

A natural extension to the general cost estimation framework is to update the cost estimation of an edge-interval pair not only based on its own observations, but also that of adjacent edges or intervals. In a sense, this would resemble the ExpandingSearch algorithm, the baseline aggregation model used in our experiments.

The k-Routes corpus generation strategy for road2vec proved the most reliable and the most accurate on the travel time estimation task. It is therefore compelling to utilize the trips in the training set for corpus generation in some sense. The challenge is that not all edges or edge descriptors are guaranteed to occur in a training trip and they can therefore not be used in a straight-forward manner to generate representations of all edges in a road network.

Another important aspect of corpus generation is the choice of descriptor, but it is not clear what constitutes a good descriptor beyond the basic requirements that they must be discrete, topologically independent, and not be distinct for each edge in a road network. Further research in this direction is therefore of interest.

In our adaption of the PVDM architecture to feature learning in road networks, we use edges as the analogy to paragraphs. The feature representation of an edge therefore captures distinguishing characteristics local to the edge, as demonstrated by the results of the structural similarity task. In the same manner, using a region as the analogy to paragraphs will yield feature representations which capture the distinguishing characteristics of a region. Using such a representation of a region may allow a regression model to account for regional differences in traffic, such as different peak hours.

Finally, classification techniques from the language modelling domain, such as word averages [20], may also prove applicable to the road network domain. Using an average of learned edge representation may be used to represent any path or route in a road network.

APPENDIX A
CONTINUOUS BAG OF WORDS

This section contains a description of the Continuous Bag of Words (CBOW) architecture for learning of feature representations of words. The PVDM described in Section 5.2 is an extension of the CBOW model and there is therefore a significant amount of overlap between this section and Section 5.2. We have however included it for documentation purposes, since the CBOW model is used in the travel time experiment of Section 6.

CBOW is a well-known neural network architecture for learning feature representations of words [20]. CBOW predicts a word based on its context: the c previous and future words in a sentence [20]. As an example, consider the sentence illustrated in Fig. 6. In the CBOW architecture the middle word "a" is predicted based on its context. If $c = 2$ the context of "a" is ("This", "is", "short", "sentence").

Fig. 22 illustrates the CBOW architecture. Each context word w_i is represented as a one-hot encoding $\mathbf{x}_i$ or $1 \leq i \leq 2c$. Therefore each context word requires as many neurons in the input layer as there are words in the vocabulary $\mathcal{V}$. The hidden layer size is set to the desired dimensionality d of the learned feature representations. Finally, the output layer has a neuron for each word in the vocabulary. The connections between the input layer and the embedding are represented by a $|\mathcal{V}| \times d$ matrix $\mathbf{W}^I$. Similarly, the connections between the embedding layer and the output layer are represented by a $d \times |\mathcal{V}|$ matrix $\mathbf{W}^O$. The feature representation of a word w_i is the ith row in $\mathbf{W}^I$.

Formally, CBOW maximizes

$$\frac{1}{T}\sum_{t=c}^{T-c} \log(\mathrm{P}(w_t \mid w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c})) \tag{21}$$

where c is the size of the context.

The values of the neurons in the neural network layers in the CBOW architecture can be written as a matrix product. Let $\mathbf{x}_i$ denote the one-hot encoding of word $w_i \in \mathcal{V}$ as a $1 \times |\mathcal{V}|$ row vector $\mathbf{x}_i = (x_1, \ldots, x_{|\mathcal{V}|})$ where $x_i = 1$ and $x_j = 0$ if $j \neq i$. Let $\mathbf{W}^I$ be the $\mathcal{V} \times d$ weight matrix connecting a context word in the input layer to the embedding layer with d neurons. The feature representation of w_i is then $\mathbf{x}_i(\mathbf{W}^I)^T$ which is equivalent to the ith row of $\mathbf{W}^I$ denoted as [26]

$$(\mathbf{v}^I_{w_1})^T = \mathbf{x}_i(\mathbf{W}^I)^T \tag{22}$$

The representation of the context words at the hidden layer is simply an average of the vectors in the context [26].

$$\begin{aligned}\mathbf{h} &= \frac{1}{2c}(\mathbf{W}^I)^T(\mathbf{x}_1 + \mathbf{x}_2 + \cdots + \mathbf{x}_{2c}) \\ &= \frac{1}{2c}((\mathbf{v}^I_{w_1})^T + (\mathbf{v}^I_{w_2})^T + \cdots + (\mathbf{v}^I_{w_{2c}})^T)\end{aligned} \tag{23}$$

where $2c$ is the number of words in the context.

Let $\mathbf{W}^O$ be the $d \times \mathcal{V}$ weight matrix connecting the embedding layer to the output layer. A score u_j for each word $w_j \in \mathcal{V}$ can be calculated as [26]

$$u_j = (\mathbf{v}^o_{w_j})^T\mathbf{h} \tag{24}$$

where $\mathbf{v}^o_{w_j}$ is the jth column of matrix W^O.

Finally, the probability of a word at the output layer can be computed by applying the softmax function to the scores [26]. Formally,

$$\mathrm{P}(w_t \mid w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c}) = \frac{\exp(u_t)}{\sum_{-c\leq i\leq c, i\neq 0}^{|\mathcal{V}|} \exp(u_{t+i})} \tag{25}$$

APPENDIX B
CORPUS GENERATION STRATEGIES EXAMPLES

An illustration of the k-Neighbor and k-Routes strategies are shown in Fig. 23 and Fig. 24

APPENDIX C
LINEAR MODEL FOR TRAVEL TIME ESTIMATION

Fruensgaard et al. [6] engineer a set of features for the purpose of training a linear regression model to estimate travel times for edges in a road networks to be used as weights. The model outperformed an existing baseline [6]. These features represent edge-interval pairs, rather than individual edges, to support uni-variate models. In this section, we describe the representation of these features, which we categorize as either *intrinsic* or *relational*, borrowing the terminology from Neville and Jensen [33].

C.1 Intrinsic Features

Intrinsic features refer to characteristics of an edge-interval pair in isolation. The intrinsic features of an edge-interval pair is a combination of the intrinsic features of the edge and the interval.

The intrinsic features of an edge are in our case synonymous its labels. As in previous work [6], map data from OSM [22] and *PlansystemDK* [23] is used to derive features of an edge from its corresponding road segment. These features are

- the *category* of a road segment,
- the *speed limit* of a road segment,
- whether or not the road segment is *within a city zone*, and
- whether or not the road segment is *connected to an intersection regulated by traffic lights*.

The category of an edge is simply represented as a one-hot encoding indicating which of the 6 categories see Paragraph C.4.1.2 an edge is classified. This representation conveys that there is no definition of closeness between two categories, i.e. main road is not more similar to a highway than a motorway is. Since an edge has exactly one category, the sum of these values is always one.

The speed limit is simply its integer value, but may be missing. In that case, we use a substitute speed limit derived from other attributes of the edge, based on OSM recommendations [34]. Whether the edge is within a city zone or not is represented by a single value of either 0 (not within a city zone) or 1 (within a city zone). The representation of whether an edge is connected to an intersection regulated by traffic lights used to be represented similarly (by a single

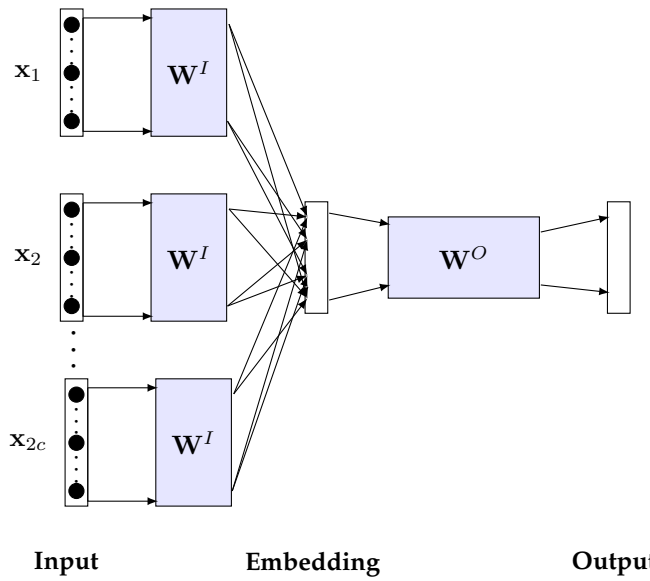


Fig. 22: An illustration of the CBOW architecture. Note that the matrix $\mathbf{W}^I$ is duplicated for each word in the context, s.t. each row yields a unique feature representation of each word in the vocabulary.

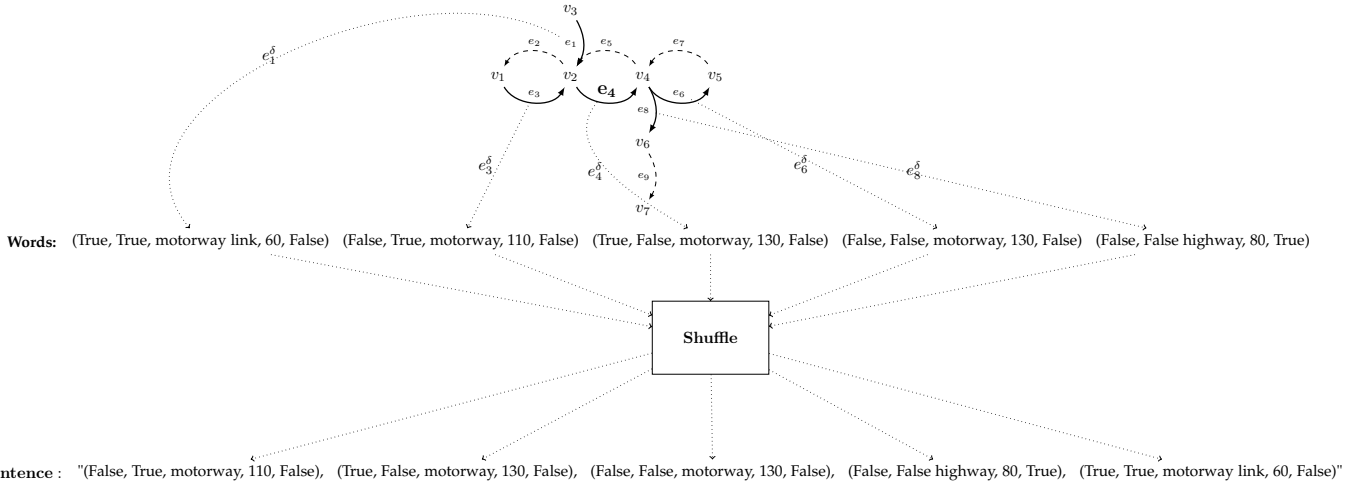


Fig. 23: An illustration of the mapping from graph to a sentence using the k-Neighbors sampling strategy for edge e_4 with a $k = 1$. The labels of a word/descriptor are (has traffic light begin, has traffic light end, category, speedlimit, within city). The words are shuffled before the sentence construction.

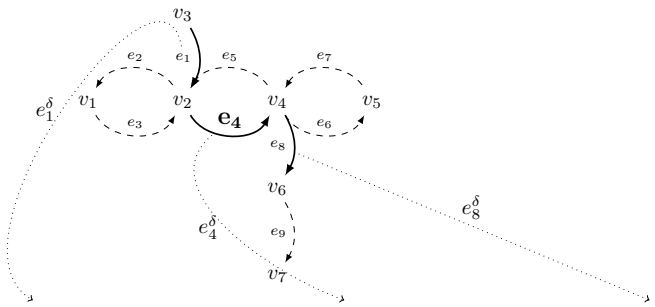
value, 0 or 1), but instead we now represent it as two values; one for each end of the edge. This allows us to distinguish which end of the segment that is connected to a traffic light regulated intersection.

Intervals are enumerated sequentially from 0 to $g - 1$, where g is the number of intervals the time of day has been split into, indicating the number of intervals since midnight. Previously, an interval was simply represented by its enumeration [6]. This representation is inherently linear, however it cannot represent that 11 PM and 1 AM are equally distant from midnight; that is, the distance function from midnight is not *symmetric*. This representation may also be used with a continuous, rather than discrete representation of time by substituting g with the number of hours per day and i with hours since midnight, e.g. 2.1. Fig. 25 shows that using this feature representation of time, the Euclidean distance from midnight is symmetric around midday (12).

C.2 Relational Features

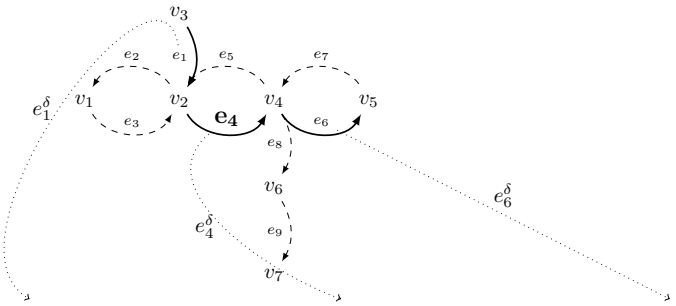
A relational feature summarize characteristics of related edge-intervals, in this case edge-intervals that are *spatially adjacent*, i.e. edge-intervals of a connected edge with the same interval, or *temporally adjacent*, i.e. edge-interval on the same edge with a different interval.

The expectation is that the speed of an edge-interval pair (e, I) is correlated with that of its spatially adjacent edge-interval pairs, given they are reasonably similar. For instance, if e represents a motorway and a connected edge



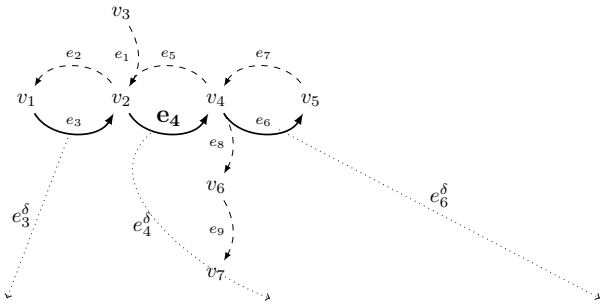
Sentence: "(True, True, motorway link, 60, False), (True, False, motorway, 130, False), (False, False highway, 80, True)"

(a)



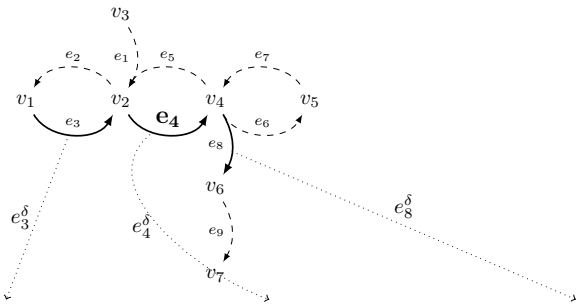
Sentence: "(True, True, motorway link, 60, False), (True, False, motorway, 130, False), (False, False, motorway, 130, False)"

(b)



Sentence: "(False, True, motorway, 110, False), (True, False, motorway, 130, False), (False, False, motorway, 130, False)"

(c)



Sentence: "(False, True, motorway, 110, False), (True, False, motorway, 130, False), (False, False highway, 80, True)"

(d)

Fig. 24: An illustration of the mapping from graph to sentences using the k-Routes sampling strategy for edge e_4 with a $k = 1$. The labels of a word/descriptor are (has traffic light begin, has traffic light end, category, speedlimit, within city).

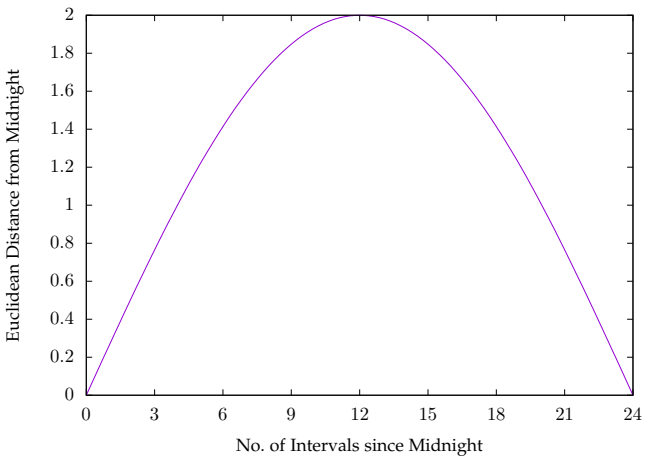


Fig. 25: Euclidean distance from the representation of midnight for different 1 hour intervals.

e' represents a turnoff, (e, I) and (e', I) are not expected to be similar. The speed of similar spatially adjacent edge-intervals is summarized in the spatial context.

The relational feature which summarizes the speed of spatially adjacent edge-interval pairs is referred to as the *spatial context*.

Definition C.1. *Given an unweighted road network G=(V, E, T, L, l , w), an* initial weight function w_0*, a similarity measure Definition C.1, and a similarity threshold ω, the spatial context of (e, I) defined as*

$$SpatialContext(e, I, sim) = \langle\{w_0(e', I) | e' \in E \wedge sim((e, I), (e', I)) \geq \omega\}\rangle \quad (26)$$

where $sim\colon E \times T \times E \times T \to \mathbb{R}^+$ is a similarity measure.

The initial weight function w_0 referred to in Definition C.1 is an initial estimate of w. The initial weight function used in this work is from our previous work [6] and assigns an initial weight to an edge-interval pair (e, I) based trajectories crossing the (e, I) if available and otherwise uses network averages depending on its labels and the interval I. We refer to Section C.3 for details.

The relational feature which summarizes the speed of temporally adjacent edge-interval pairs is referred to as the *temporal context*.

Definition C.2. *Given a discretized road network $G_g = (V, E, T_g, L, l, w)$, an initial weight function w_0 the temporal context is a set*

$$TemporalContext(e, I) = \{w_0(e, I') | (e, I') \in E \times T_g \setminus \{I\}\} \quad (27)$$

We have made two modifications to Definition C.2 compared to previous work. First, the temporal context was previously defined with a size parameter, but parameter tuning showed higher values of this parameter (up to g) are always better. Second, given a large enough size parameter $\geq g$, an edge-interval pair (e, I) the temporal context could contain $w_0(e, I)$. Due to the choice of w_0 (see Section C.3), which is based on trajectories from the training set, the value of $w_0(e, I)$ was always an aggregate of the trajectories found in the training set and would lead to overfitting.

C.3 The Initial Weight Function

In this section we describe the initial weight function w_0 used for computing the spatial and temporal context (see Section C.2) first defined in our previous work [6]. Although the weight function is the same, this section is a complete rewrite of the original description.

The initial weight function uses aggregates of decreasingly relevant trips in the road network. We therefore define the initial weight function in terms of a *network average function*.

Given a discretized graph $G_g = (V, E, T_g, L, l, w)$, let the mean of an edge e across a set of intervals $\mathcal{I} \subseteq T_g$ be computed as

$$edge\_mean^{G_g}_{O_{G_g}}(e, \mathcal{I}) = \left\langle \left\{ \langle\{cost \mid (e, t, cost) \in O'_G(e, I)\}\rangle \mid I \in \mathcal{I} \right\} \right\rangle$$

such that each interval is weighted evenly in the mean.

Definition C.3. *A network average function for a set of trips O observed in a discretized road network $G_g = (V, E, T_g, L, l, w)$ is a function*

$$avg^{G_g}_{O_{G_g}}(E', \mathcal{I}) = \langle\{edge\_mean(e, \mathcal{I}) \mid e' \in E'\}\rangle$$

where $e \in E$, $\mathcal{I} \subseteq T_g$, and $E' \subseteq E$.

Note that $avg^{G_g}_{O_{G_g}}(E', \mathcal{I})$ is undefined if no there are no trips for any edge $e \in E'$ within any of the intervals $I \in \mathcal{I}$.

The most relevant trips of an edge is those that have been observed on it, but such trips are not guaranteed to exist, i.e. $O_G(e) = \emptyset$. We therefore define two suitable sets on which to perform network averages. We first consider the set $E^e_1 \subseteq E$, which is the set of edges which has the same category, speed limit, and city zone labels as $e \in E$.

$$E^e_1 = \left\{ e' \;\middle|\; \begin{array}{r} e' \in E, \\ l(e) = (c, l, t_{start}, t_{end}, z), \\ l(e') = (c, l, t'_{start}, t'_{end}, z) \end{array} \right\}$$

Next, we consider a slightly less restrictive set $E^e_2 \subseteq E$, which is the set of edges with the same speed limit and city zone labels as $e \in E$.

$$E^e_2 = \left\{ e' \;\middle|\; \begin{array}{r} e' \in E, \\ L(e) = (c, l, t_{start}, t_{end}, z), \\ L(e') = (c', l, t'_{start}, t'_{end}, z) \end{array} \right\}$$

We can now define the initial weight function.

Definition C.4. *Given a distrectized road network $G_g = (V, E, T_g, L, l, w)$ and a set of observed trips O_G, the initial weight function w_0 is defined as*

$$w_0(e, I) = \begin{cases} avg^G_{O_G}(\{e\}, \{I\}) & \textit{if } avg^G_{O_G}(\{e\}, \{I\}) \textit{ is defined} \\ avg^G_{O_G}(\{e\}, T_g) & \textit{else if } avg^G_{O_G}(\{e\}, T_g) \textit{ is defined} \\ avg^G_{O_G}(E^e_1, \{I\}) & \textit{else if } avg^G_{O_G}(E^e_1, \{I\}) \textit{ is defined} \\ avg^G_{O_G}(E^e_2, \{I\}) & \textit{else if } avg^G_{O_G}(E^e_2, \{I\}) \textit{ is defined} \\ limit(e) & otherwise \end{cases}$$

When using w_0 to assign an initial weight to an edge e for an interval I, the intution is that the most relevant trips are those recorded on e during I, then those recorded on e throughout the day, then trips from edges similar to e. Finally, if all else fails, the speed limit of e is simply used, which ensures that w_0 is complete, i.e. it can assign a weight to any edge and interval.

C.4 Feature Representations

This section describes the vector representations of the intrinsic and relational features and is largely the same as in [6] with only minor rewrites. We have included it here for completeness.

C.4.1 Intrinsic Features

An edge-interval in our graph has the following intrinsic features:

1) Category - String
2) Speed Limit - Integer
3) Traffic Signal - Boolean
4) Within City - Boolean

C.4.1.1 Category

The categories we consider are the following (OSM type mapping shown in Table 9):

- *motorway*
- *motorway link*
- *expressway*
- *highway*
- *main road*
- *connecting road*

We represent the category feature as a binary vector of length six, where each binary value, 0 or 1 represents membership to a category. This representation conveys that there is no definition of closeness between two categories, i.e. we cannot say that a *main road* is more similar to a *highway* than a *motorway* is. Since an edge has exactly one category, the sum of these values is always one.

Formally the feature representation of a category is

$$\phi_c(e, t) = \begin{cases} (1, 0, 0, 0, 0, 0) & \text{if } e.category \text{ is } motorway \\ (0, 1, 0, 0, 0, 0) & \text{if } e.category \text{ is } motorway\ link \\ (0, 0, 1, 0, 0, 0) & \text{if } e.category \text{ is } expressway \\ (0, 0, 0, 1, 0, 0) & \text{if } e.category \text{ is } highway \\ (0, 0, 0, 0, 1, 0) & \text{if } e.category \text{ is } main\ road \\ (0, 0, 0, 0, 0, 1) & \text{if } e.category \text{ is } connected\ road \end{cases}$$

C.4.1.2 Speed Limit

The set of speed limits consists of all possible 13 speed limits: U, 15, 20, 30, 40, 45, 50, 60, 70, 80, 90, 110, and 130, where U indicates an unknown speed limit. Speed limit is given in $\mathrm{km\,h^{-1}}$. We use the OSM [34] rules encoded in SPEEDLIMIT on Line 1 to 5 function in Algorithm 4 to assign a speed limit in case of an unknown speed limits, yielding 12 unique speed limits. We encode the speed limit as a vector of length one containing a single integer value, the speed limit.

C.4.1.3 Traffic Signal and Within a City Zone

We include features to represent that a traffic signal is within 10 meters of an entire edge and whether the segment is within a city zone. We represent each of these values as a binary vector of length one with a binary value of 1 or 0. We define

$$\phi_{ts}(e, t) = \begin{cases} 1 & \text{if } traffic\_signal(e) \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

and

$$\phi_{wc}(e, t) = \begin{cases} 1 & \text{if } e.within\_city(e) \\ 0 & \text{otherwise} \end{cases} \tag{29}$$

C.4.2 Relational Features

In this section we describe the relational features which are based on connected edges and adjacent intervals. We focus on the temporal and spatial context of an edge-interval, which we define below. The relational features we have chosen for this work are motivated by the correlations found in Fruensgaard et al. [6].

C.4.2.1 Temporal and Spatial Context

The temporal context is based on the fact that traffic from one point in time flows into the next and the correlation between time intervals found in [6].

We define the temporal context as

Definition C.5. *Given an edge-interval (e, t) and the i-adjacent intervals to t $(t_{i,p}, \ldots, t_{1,p}, t_{1,s}, \ldots, t_{i,s})$ where p is the predecessor and s is the successor, which are the previous and following intervals respectively,*

$$\begin{aligned} context^i_T(w_0, e, t) = &(w_0(e, t_{i,p}), \ldots, w_0(e, t_{1,p}), \\ &w_0(e, t_{1,s}) \ldots, w_0(e, t_{i,s})) \end{aligned}$$

where w_0 is a complete weight function.

The challenge is that we do not have an accurate complete weight function. We therefore instead use an estimate of the complete weight function instead. We will describe how we compute this estimate in Section C.1.

In [6] there is a strong correlation between the speeds of connected edges. We therefore encode the speeds of connected edges as the spatial context of an edge-interval pair. The spatial context for an edge-interval pair $(e, t) \in \mathbb{E} \times \mathbb{T}$ is a singleton given by

Definition C.6.

$$context^{pred}_S(w_0, e, t) = (\sum_{e' \in \mathbb{E}'} \frac{w_0(e', t)}{|\mathbb{E}'|})$$

where $pred \colon \mathbb{E} \times \mathbb{E} \to \{true, false\}$ is a predicate function, w_0 is a total weight function, and $\mathbb{E}' = \{e' \in \mathbb{E} | pred(e, e') = true\}$

In other words, we calculate the arithmetic mean of speed at time t of all connected edges that fulfill the predicate $pred$. The predicate function allows some flexibility for the spatial context. We may for instance only consider edges e' connected to e with the same road category and speed limit. As with the the temporal context, w_0 is not given.

APPENDIX D
THE EXPANDINGSEARCH ALGORITHM

In this section we describe the ExpandingSearch algorithm by al. [5] shown in Algorithm 4. The algorithm takes as input an edge-interval $(e,t) \in E \times T_g$ and returns the cost of traversing edge e at time t. ExpandingSearch assume high spatial and temporal autocorrelation in the road network. The first part of this section gives a description of the algorithm, followed by a thorough explanation.

Algorithm 4 The ExpandingSearch algorithm.

Require: A time-dependent graph $G = (V, E, T, L, l, w)$, a target edge $e \in E$ to find a cost for, an interval $t \in T_g$ where time granularity $g \in \mathbb{R}^+$, an integer $k \in \mathbb{N}^+$ used for KNearest, two predicates $pred_1, pred_2 \colon E \times E \rightarrow \{true, false\}$ and a minimum number of trips $m \in \mathbb{N}^+$
Ensure: A weight assignment for $(e,t) \in E \times T_g$

```
1: function EXPANDINGSEARCH(e, t)
2:     i ← 0
3:     repeat
4:         i ← i + 1
5:     until nil ≠ STRATEGY(i, e, t)
6:     return STRATEGY(i, e, t) · length(e)

1: function STRATEGY(i, e, t)
2:     if i = 1 then return TEMPORAL(e, t)
3:     else if i = 2 then return CONNECTED(e, pred_1)
4:     else if i = 3 then return KNEAREST(e, pred_1, k)
5:     else if i = 4 then return CONNECTED(e, pred_2)
6:     else if i = 5 then return KNEAREST(e, pred_2, k)
7:     else return SPEEDLIMIT(e)

1: function SPEEDLIMIT(e)
2:     if speed_limit(e) ≠ 0 then return speed_limit(e)
3:     else if within_city(e) then return 50
4:     else if category(e) = "motorway" then return 130
5:     else return 80
```

D.1 Description

ExpandingSearch invokes, in predefined order, six different strategies until one returns a cost of (e,t). The first strategy searches for temporally close trips, if the strategy fails, the search is expanded to search for spatially close trips. In case all previous strategies fail, the algorithms will return a predefined cost based on e's .

D.2 ExpandingSearch Explained

On Lines 3 to 5, the algorithm calls one strategy after another until a speed of (e,t) is returned, such that the cost of (e,t) can be inferred.

The function STRATEGY represents an implicit distance metric or search area from the edge e to some trips. Each strategy outlined in function STRATEGY considers trips across increasing distance. In other words, there is no explicit distance metric; it is implied by the ordering of the strategies and their execution.

The first strategy TEMPORAL is invoked on Line 2 in Strategy , and is shown in detail in Algorithm 5. TEMPORAL performs a temporal search on the specified edge e. At Line 2 the current interval to search I is initially set to t. At Line 5 the function

$$get\_trips(e, I) = \{trip \mid trip = (e, t, cost) \in O_G(e) \wedge t \in I\} \quad (30)$$

is invoked, returning all trips that has occurred over the given edge e during the given interval I. If the returned number of trips is greater than or equal to m, then the mean cost of the trips in m is returned. This mean cost is computed as

$$trip\_mean(e, trips) = \frac{1}{|trips|} \sum_{trip \in trips} \sum_{(e,t,cost) \in trip} \frac{length(e)}{cost} \quad (31)$$

on Line 8. If the number of trips is below m, then the interval is expanded on Lines 9 to 11.

Algorithm 5 The temporal search strategy invoked by the ExpandingSearch algorithm.

```
1: function TEMPORAL(e, t)
2:     I ← t
3:     trips ← ∅
4:     repeat
5:         trips ← get_trips(e, I)
6:         start ← I.start; end ← I.end
7:         if |trips| ≥ m then
8:             return trip_mean(trips)
9:         start ← predecessor(I)
10:        end ← successor(I)
11:        I ← [start; end)
12:    until I.end < I.start
13:    return nil
```

The temporal search fails if less than m trips have occurred on the edge. In that case, the algorithm expands the search spatially by searching for trips on nearby similar edges. The spatially search is invoked by calling either CONNECTED or KNEAREST at Lines 3 to 6 in STRATEGY of Algorithm 4. The functions are called with one of the two globally specified predicates $pred_1$ or $pred_2$ where $pred_1$ is expected to be more strict than $pred_2$. KNEAREST and CONNECTED are both defined in Algorithm 6.

The CONNECTED function finds all edges connected to e with at least m trips across that fulfil the predicate $pred$ at Line 2, by calling function

$$get\_connected\_edges(e, m, pred) = \left\{ \begin{array}{c} e' | e' \in E \wedge connected(e, e') \wedge \\ |O_e| \geq m \wedge pred(e, e') \end{array} \right\} \quad (32)$$

If any connected edges are found, the mean speed of the edges is calculated and returned at Lines Line 4 and Line 6 according to function

$$edge\_mean(E') = \frac{1}{|E'|} \sum_{e \in E'} trip\_mean(e, O_G(e)) \quad (33)$$

If any trips has occurred on edge e, i.e. $|O_e| > 0$, these are included in the mean at Line 4.

KNEAREST is identical to CONNECTED, except instead of storing connected edges in E' at Line 9, the k spatially nearest edges with at least m trips that fulfil the input predicate $pred$. These are returned by the function $get\_k\_nearest$ are.

Algorithm 6 The spatial search strategies invoked by the ExpandingSearch algorithm.

1: **function** CONNECTED(e, $pred$)
2: $E' \leftarrow get\_connected\_edges(e, m, pred)$
3: **if** $|E'| > 0 \wedge |O_e| > 0$ **then**
4: **return** $edge\_mean(E' \cup \{e\})$
5: **else if** $|E'| > 0 \wedge |O_e| = 0$ **then**
6: **return** $edge\_mean(E')$
7: **else return** nil
8: **function** KNEAREST(e, $pred$)
9: $E' \leftarrow get\_k\_nearest(k, e, m, pred)$
10: **if** $|E'| > 0 \wedge |O_e| > 0$ **then**
11: **return** $edge\_mean(E' \cup \{e\})$
12: **else if** $|E'| > 0 \wedge |O_e| = 0$ **then**
13: **return** $edge\_mean(E')$
14: **else return** nil

We extend the ExpandingSearch algorithm slightly in Algorithm 4 by adding the function Speedlimit , s.t. it defaults to the speed limit of the edge e if m trips are not found by the preceding steps. This can happen

- if the number of trips across edge e is smaller than m and
- there are no connected or nearby edges with m across them who fulfil the $pred_1$ or $pred_2$.

Defaulting to the speed limit ensures that the ExpandingSearch algorithm is complete; i.e. it can assign a weight to any edge-interval.

The speed limit may also be missing, however. We therefore assign a speed limit according to the recommendations of *OpenStreetMap* [22], as encoded in function SPEEDLIMIT in Algorithm Algorithm 4.

APPENDIX E
OSM TYPES

Table 9 shows the mapping between road types and the corresponding type in OSM.

APPENDIX F
ALL OSM CATEGORIES FOR NORTHERN JUTLAND

Table 10 shows all road segment categories for Northern Jutland.

Segment Type	*OSM Type*
Motorway	motorway
Motorway Links	motorway_link
Expressway	trunk trunk_link
Main Road	primary primary_link
Highway	secondary secondary_link
Connecting Road	tertiary tertiary_link

TABLE 9: The mapping between road types and the corresponding type in OSM.

Segment Type
service
primary link
tertiary link
motorway link
unpaved
trunk link
primary
secondary
unclassified
residential
trunk
track
living street
tertiary
secondary link
road
motorway

TABLE 10: All road segment categories for Northern Jutland.

APPENDIX G
COMPARISON OF ALL MODELS EKSTRA TABLES AND GRAPHS

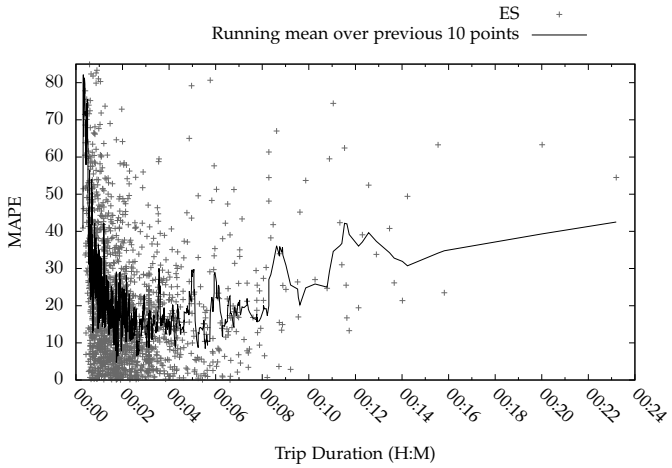
This section includes Table 11, Table 12, Fig. 26, Fig. 27, Fig. 28, Fig. 29 and Fig. 30.

	Category			
Model	*City*	*Municipality*	*Region*	*Country*
ExpandingSearch	21.48	20.27	11.03	8.79
LIN-ENG-OBS	21.20	18.96	**9.83**	**8.57**
LIN-EMB-OBS	**21.08**	**17.30**	10.21	8.78

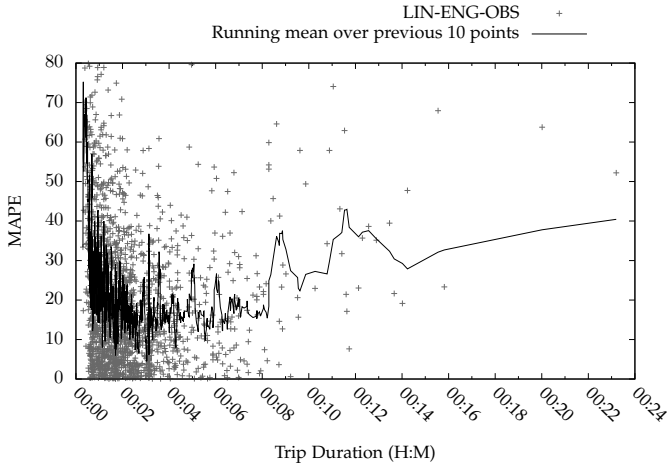
TABLE 11: MAPE of ExpandingSearch, LIN-ENG-OBS and LIN-EMB-OBS for all trips categories between 08:15-15:30.

	Category			
Model	*City*	*Municipality*	*Region*	*Country*
ExpandingSearch	21.59	20.68	12.71	10.87
LIN-ENG-OBS	20.99	18.29	**12.17**	**8.97**
LIN-EMB-OBS	**19.96**	**17.32**	12.23	10.32

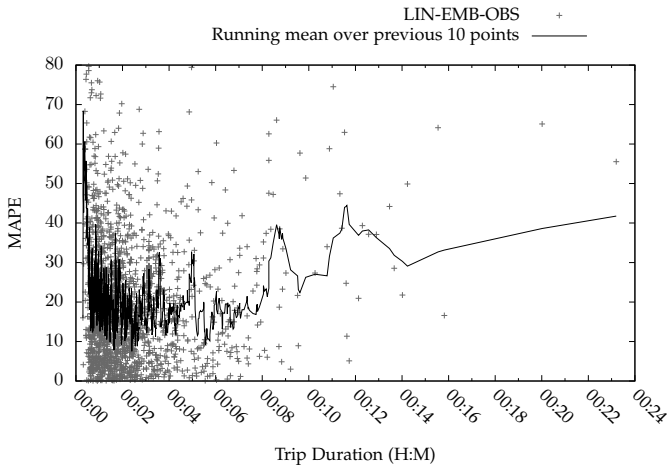
TABLE 12: MAPE of ExpandingSearch, LIN-ENG-OBS and LIN-EMB-OBS for all trips categories between 16:30-22:00.



(a) ExpandingSearch.

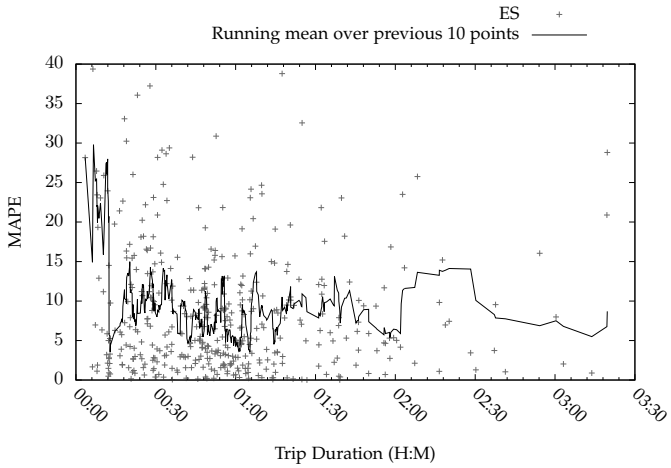


(b) LIN-ENG-OBS.

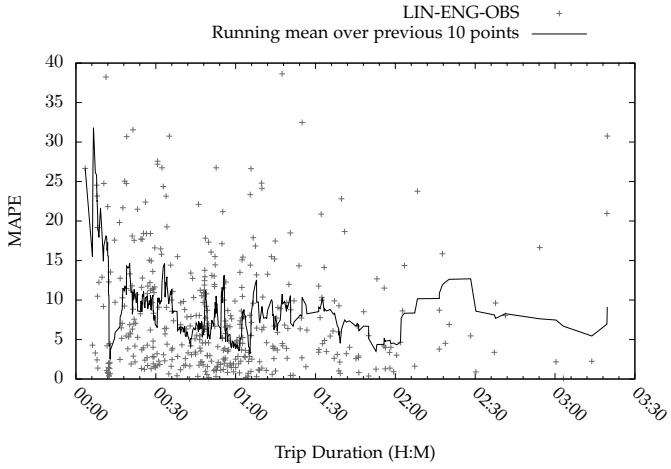


(c) LIN-EMB-OBS.

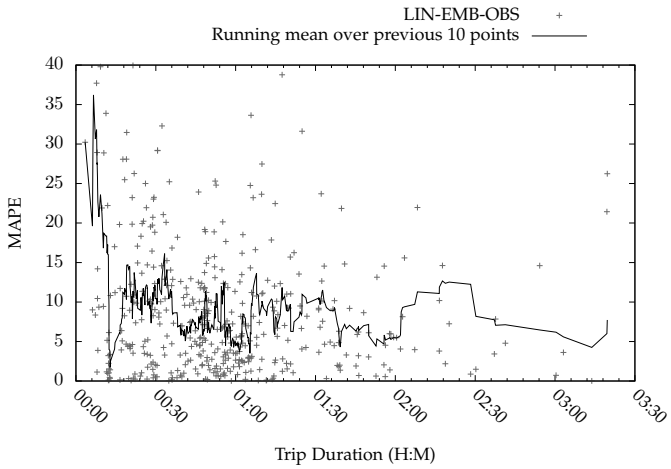
Fig. 26: MAPE as a function of the duration of city trips for the three models ExpandingSearch, LIN-EMB-OBS and LIN-ENG-OBS.



(a) ExpandingSearch.

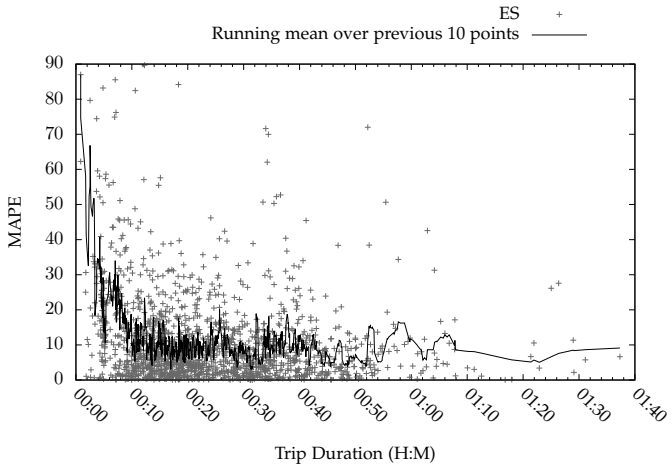


(b) LIN-ENG-OBS.

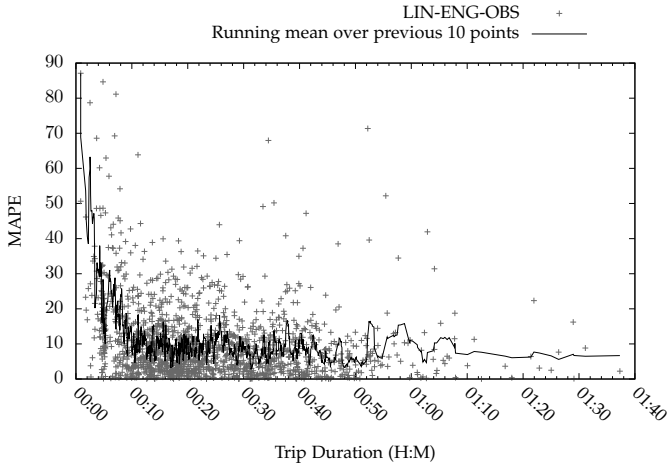


(c) LIN-EMB-OBS.

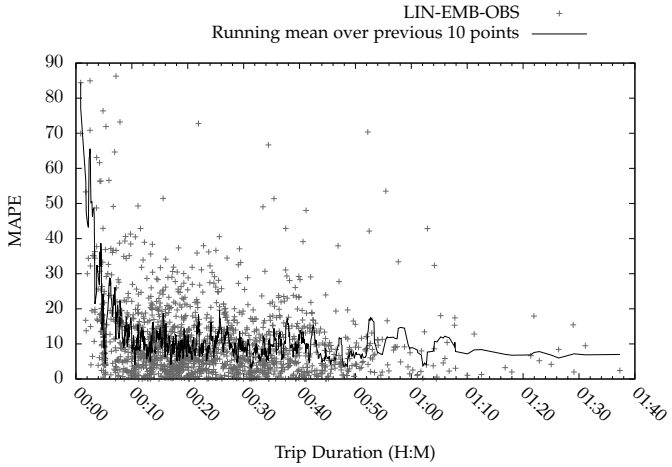
Fig. 27: MAPE as a function of the duration of country trips for the three models ExpandingSearch, LIN-EMB-OBS and LIN-ENG-OBS.



(a) ExpandingSearch.

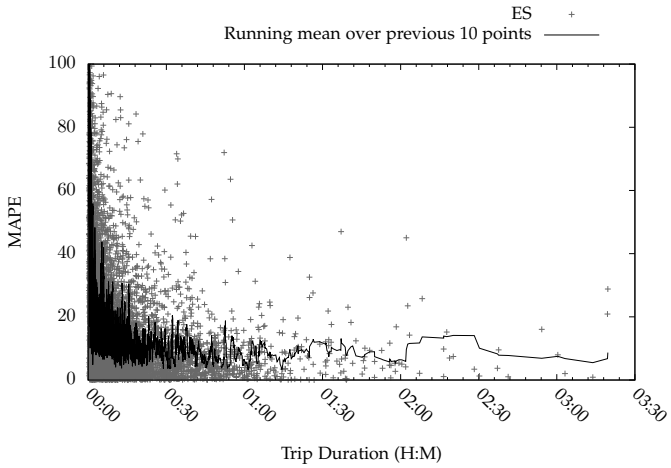


(b) LIN-ENG-OBS.

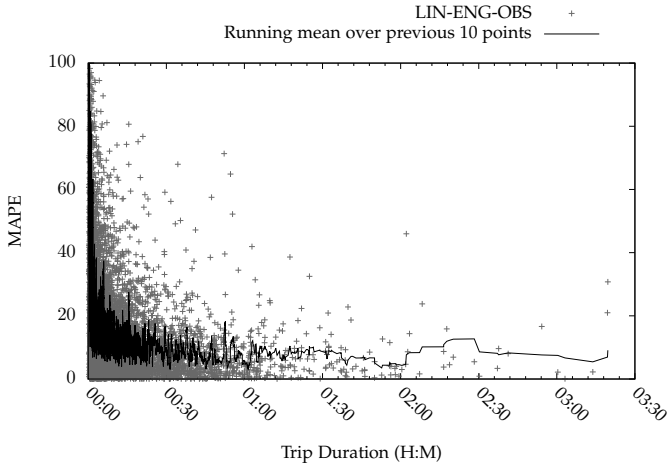


(c) LIN-EMB-OBS.

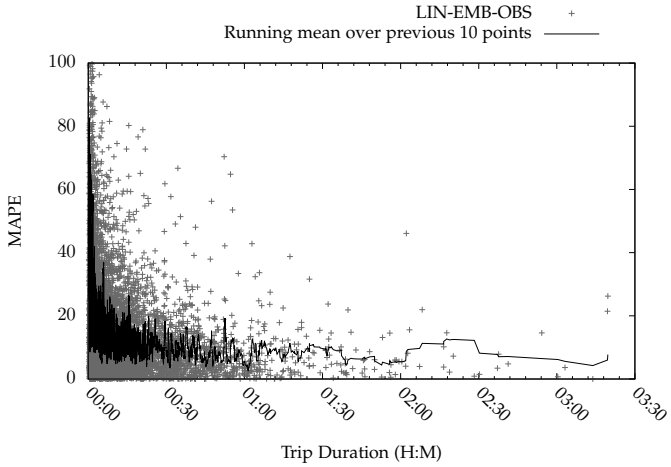
Fig. 28: MAPE as a function of the duration of region trips for the three models ExpandingSearch, LIN-EMB-OBS and LIN-ENG-OBS.



(a) ExpandingSearch.

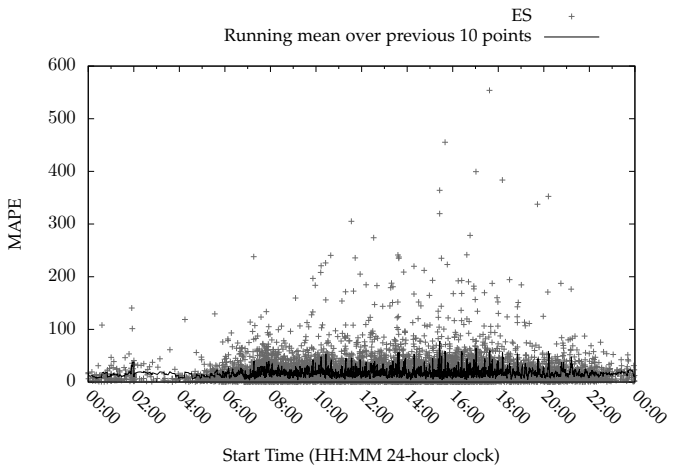


(b) LIN-EMB-OBS.

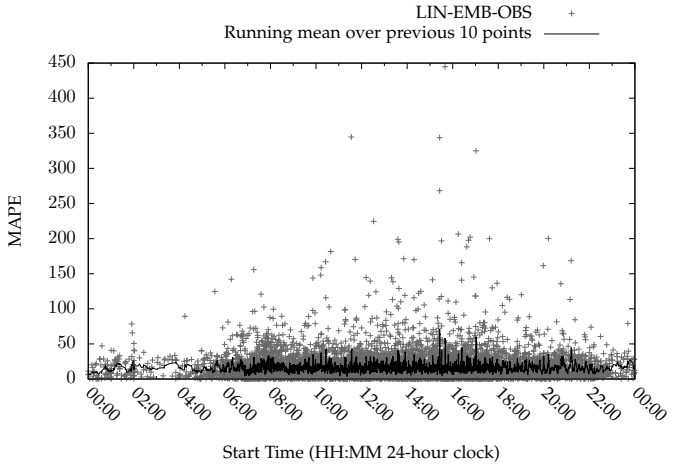


(c) LIN-EMB-OBS.

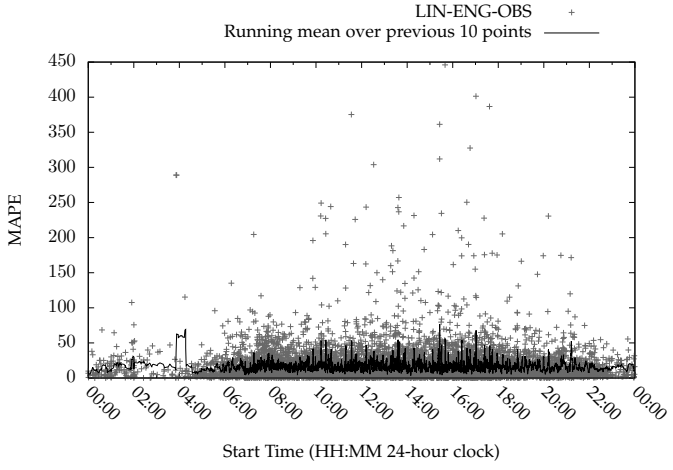
Fig. 29: MAPE as a function of the duration of trips in all trips categories for the three models ExpandingSearch, LIN-EMB-OBS and LIN-ENG-OBS.



(a) ExpandingSearch



(b) LIN-EMB-OBS



(c) LIN-ENG-OBS

Fig. 30: MAPE of each trip for the three models ExpandingSearch, LIN-EMB-OBS and LIN-ENG-OBS depending on trips start time.

APPENDIX H
THE HUBER LOSS FUNCTION

The Huber loss function is designed to be robust to outliers [28]. Formally, it is defined as

$$L_\epsilon = \begin{cases} \frac{1}{2}|y - f(x)| & \text{if } |y - f(x)| \leq \epsilon \\ \epsilon(|y - f(x)| - \frac{1}{2}\epsilon^2) \end{cases} \quad (34)$$

where y is the target value of a training instance and $f(x)$ is predicted value. Observe that if the absolute difference between the target value and the predicted value $|y - f(x)|$ exceeds epsilon, the function uses linear loss rather than squared loss. Thus the loss is less affected by outliers in the training, giving them less weight during model training.

APPENDIX I
PARAMETER TUNING

We use MAPE to determine the quality of the parameters, where a lower MAPE means higher quality parameters.

I.1 LIN-EMB

We initially ran an exhaustive parameter tuning, trying all unique combinations of the following parameters

- $Iterations \in \{1, 5, 10, 20, 40, 100, 1000, 5000\}$
- $|\mathbf{h}| \in \{10, 50, 100, 200, 300, 400, 500\}$
- $Learning\ rate \in \{0.1, 0.001, 0.0001, 0.00001\}$
- $Context(c) \in \{1, 2, 3, 4\}$
- $Distance(k) \in \{1, 2, 3, 4\}$

yielding

- $Iterations \in \{100\}$
- $|\mathbf{h}| \in \{200\}$
- $Learning\ rate \in \{0.001\}$
- $Context(c) \in \{1\}$
- $Distance(k) \in \{3\}$

as the ones with lowest MAPE.

We tested for $k > 6$ as we where limited to a server with 32GB RAM. $d = 5$ cost $\approx 10,6GB$ RAM and $d = 6 >$ 32GB.

We illustrate the impact of $|\mathbf{h}|$, Distance and Window by using the best parameters, shown above, and then change one parameter at the time on the validation set. This is shown in Fig. 31, Fig. 32 and Fig. 33.

I.1.1 Huber

We initially ran an exhaustive parameter tuning, trying all unique combinations of the following parameters

- $Epsilon \in \{1, 1.00001, 1.0001, 1.001, 1.01, 1.1, 1.2, 1.3, 1.4, 1.5, 2, 3, 4, 5\}$
- $Iterations \in \{25, 100, 250, 500, 1000, 5000\}$
- $Alpha \in \{0.0000001, 0.000001, 0.00001, 0.0001, 0.001, 0.1\}$

yielding

- $Epsilon \in \{1.1\}$
- $Iterations \in \{5000\}$
- $Alpha \in \{0.0001\}$

as the ones with lowest MAPE.

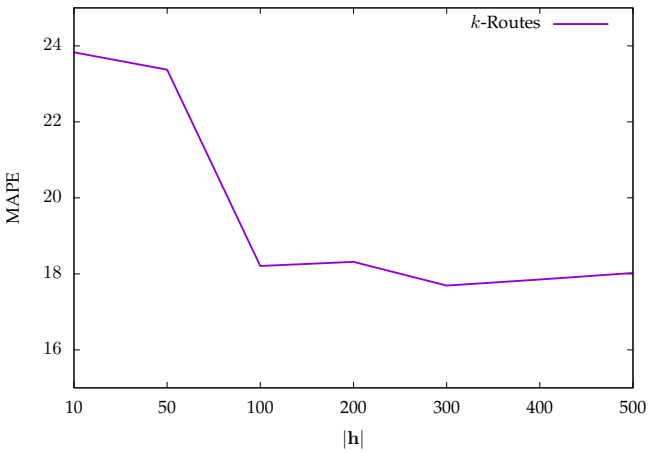


Fig. 31: Mape of k-Routes using increasing $|\mathbf{h}|$ size, the rest of the parameters are as follow $Iterations = 100$, $learning\ rate = 0.001$, $Context = 1$, $Distance = 3$.

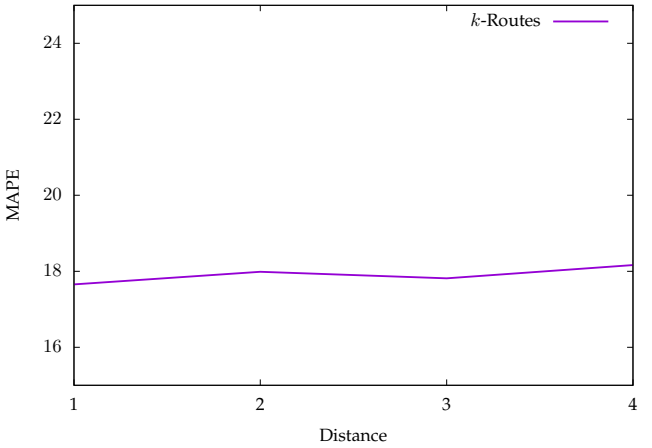


Fig. 32: Mape of k-Routes using increasing Distance size, the rest of the parameters are as follow $Iterations = 100$, $learning\ rate = 0.001$, $Context = 1$, $|\mathbf{h}| = 200$.

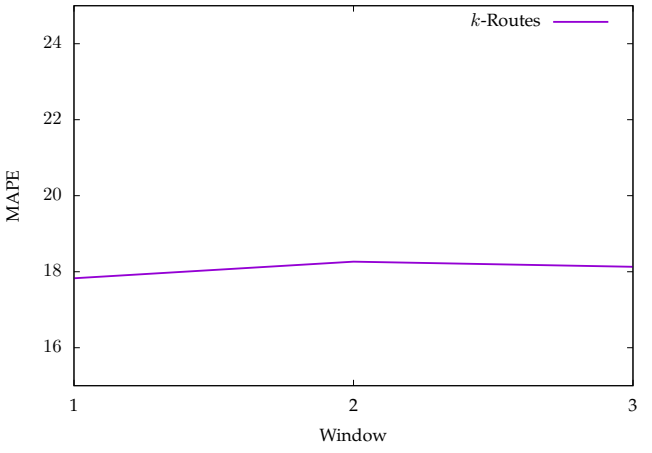


Fig. 33: Mape of k-Routes using increasing Context size, the rest of the parameters are as follow $Iterations = 100$, $learning\ rate = 0.001$, $Distance = 3$, $|\mathbf{h}| = 200$.

I.2 LIN-EMB-EDGE

We initially ran an exhaustive parameter tuning, trying all unique combinations of the following parameters

- $Iterations \in \{1, 5, 10, 20, 25, 40, 50, 100, 1000\}$
- $|\mathbf{h}| \in \{200, 400, 500, 600, 700, 1000\}$
- $Learning\ rate \in \{0.1, 0.05, 0.001, 0.0001, 0.0000, 0.00001\}$
- $Context(c) \in \{1, 3, 5, 10\}$
- $Distance(k) \in \{1, 3, 5\}$

yielding

- $Iterations \in \{25\}$
- $|\mathbf{h}| \in \{400\}$
- $Learning\ rate \in \{0.1\}$
- $Context(c) \in \{3\}$
- $Distance(k) \in \{1\}$

as the ones with lowest MAPE.

We tested for $k > 6$ as we where limited to a server with 32GB RAM. $d = 5$ cost $\approx 10,6GB$ RAM and $d = 6 >$ 32GB.

I.2.1 Huber

We initially ran an exhaustive parameter tuning, trying all unique combinations of the following parameters

- $Epsilon \in \{1, 1.00001, 1.0001, 1.001, 1.01, 1.1, 1.2, 1.3, 1.4, 1.5, 2, 3, 4, 5\}$
- $Iterations \in \{25, 100, 250, 500, 1000, 5000\}$
- $Alpha \in \{0.0000001, 0.000001, 0.00001, 0.0001, 0.001, 0.1\}$

yielding

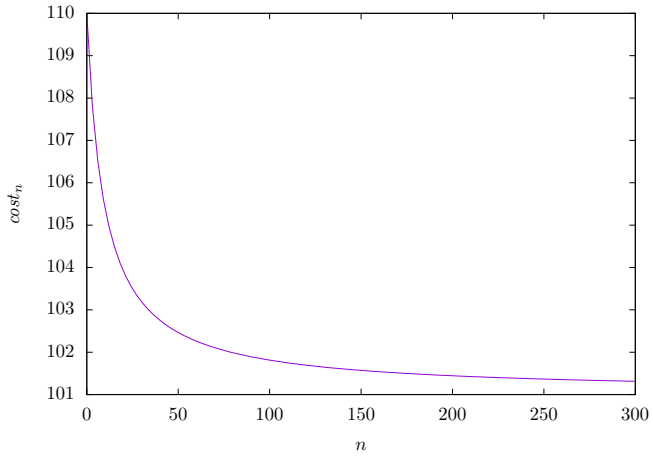
- $Epsilon \in \{1.1\}$
- $Iterations \in \{5000\}$
- $Alpha \in \{0.0001\}$

as the ones with lowest MAPE.

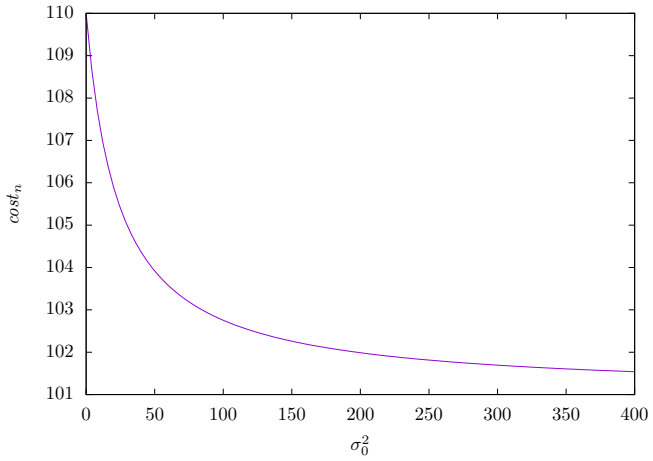
APPENDIX J
EFFECT OF THE PARAMETERS ON THE UPDATED ESTIMATE

This appendix supplements Section 4 by demonstrating how the variance of the mean σ_0^2, the observation variance σ^2, and the number of observations n affect the updated estimate $cost_n$.

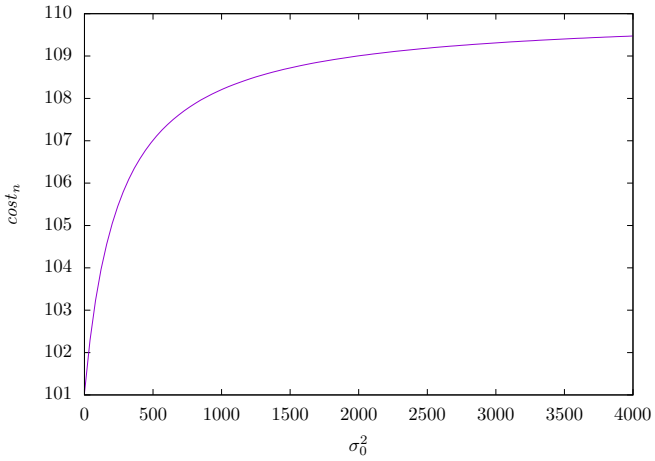
Fig. 34 shows how a prior (normalized) cost estimate of the distribution shown in Fig. 2 is updated. The prior cost estimate is simply the speed limit of the edge, i.e. $cost_0 = 110$. Fig. 34a shows that the prior estimate dominates with low n, since the updated estimate is based on very few observations and is thus less reliable. Similary, Fig. 34b shows that the prior estimate dominates if the faith in the estimate is very high, i.e. the estimation variance is low. Conversely, if the observation variance σ^2 is low we have more faith in the observations, as illustrated by Fig. 34c, and require only a few observations to approximate the mean cost of the observations.



(a) $cost_n$ as a function of n with $\sigma_0^2 = 5^2$ and $\sigma^2 = 15.38^2$.



(b) $cost_n$ as a function of σ_0^2 with $n = 10$ and $\sigma^2 = 15.38^2$.



(c) $cost_n$ as a function of σ^2 with $n = 10$ and $\sigma_0^2 = 5^2$.

Fig. 34: The effect of (a) the number of observations n, (b) the estimation variance σ_0^2, and (c) the observation variance σ^2 on the updated cost estimate of $cost_n$ of the edge-interval shown in Fig. 2 with a prior cost estimate $cost_0 = 110$.

APPENDIX K
THE LIMITED MEMORY BROYDEN–FLETCHER–GOLDFARB–SHANNO ALGORITHM

The Limited-Memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm is an optimization method which approximates the inverse Hessian matrix used in Newton's method[29, 35]. We therefore first give a brief primer on Newton's method.

K.1 Newton's Method

Newton's method iteratively constructs a parameter vector x_i for a function f from an initial guess x_0 that converges towards a local optimum $x*$ where the gradient $f'(x*) = 0$ [35]. In the context of learning a cost estimation model, the function f is the loss across the training set.

Given a function $f : \mathbb{R}^n \to \mathbb{R}$,

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \lambda[\mathbf{H}f(\mathbf{x}_i)]^{-1}\nabla f(\mathbf{x}_i), i \geq 0 \tag{35}$$

where $\lambda \in (0, 1]$ is the step size, $\mathbf{H}f$ is the Hessian matrix of f, and ∇f is the gradient of f. x_0 is an initial guess at the optimal parameters.

The Hessian matrix is a $n \times n$ square matrix containing all second-order partial derivatives of f s.t. $\mathbf{H}f_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$ [35]. Thus it is a matrix of functions and intended to be evaluated. A second-order partial derivative is the derivative with regards to 2 variables, treating remaining variables as constants.

The gradient is the generalization of the derivative to multiple dimensions and points in the direction of the greatest increase. The gradient of a function $f((x_1, \ldots, x_n))$ is

$$\nabla f = (\frac{\partial f}{\partial x_1}, \ldots \frac{\partial f}{\partial x_n}) \tag{36}$$

K.1.1 Example

Let us consider a small example. Let $f((x_1, x_2, x_3)) = x_1 + x_2^2 + x_3^3$, $\mathbf{x}_i = (1, 2, 3)$, and $\lambda = 1$. We now compute $\mathbf{x}_{i+1}$ according to Eq. (35).

The Hessian matrix of f is

$$\mathbf{H}f = \begin{bmatrix} 0 & 1 + 2x_2 & 1 + 3x_3^2 \\ 1 + 2x_2 & 2 & 2x_2 + 3x_3^2 \\ 1 + 3x_3^2 & 2x_2 + 3x_3^2 & 6x_3 \end{bmatrix}$$

and $\mathbf{H}f$ evaluated with regards to the parameters $\mathbf{x}_i = (1, 2, 3)$ is

$$\mathbf{H}f(\mathbf{x}_i) = \begin{bmatrix} 0 & 5 & 28 \\ 5 & 2 & 31 \\ 28 & 31 & 18 \end{bmatrix}$$

and its inverse is

$$[\mathbf{H}f(\mathbf{x}_i)]^1 = \begin{bmatrix} -0.139 & 0.117 & 0.015 \\ 0.117 & -0.118 & 0.021 \\ 0.015 & 0.021 & -0.004 \end{bmatrix}$$

The gradient of f is $\nabla f = (1, 2x_2, 3x_3^2)$ and $\nabla f(\mathbf{x}_i) = (1, 4, 27)$. We can now compute $\mathbf{x}_{i+1}$ as

$$\begin{aligned} \mathbf{x}_{i+1} =& (1, 2, 3)^\intercal - [\mathbf{H}f((1, 2, 3))]^{-1}(1, 4, 27)^\intercal \\ =& (0.27, 1.79, 3.02) \end{aligned} \tag{37}$$

K.2 Approximation of the Hessian

The direction of the update $[\mathbf{H}f(\mathbf{x}_i)]^{-1}\nabla f(\mathbf{x}_i)$ is expensive to compute because the dimensionality the inverse Hessian of f is the number of model parameters [35]. The L-BFGS algorithm is a specialization of the Broyden–Fletcher–Goldfarb–Shanno (BFGS) estimation.

The update at iteration i is stored as $\mathbf{s}_i = \mathbf{x}_{i+1} - \mathbf{x}_i$ and $\mathbf{y}_i = \nabla f(\mathbf{x}_{i+1}) - \nabla f(\mathbf{x}_i)$.

Broyden–Fletcher–Goldfarb–Shanno (BFGS) iteratively updates the Hessian H of f as [29, 35]

$$\begin{aligned} \mathbf{H}_{i+1} =& \mathbf{H}_i + \frac{\mathbf{s}_i\mathbf{s}_i^\intercal}{\mathbf{y}_i^\intercal\mathbf{s}_i}[\frac{\mathbf{s}_i\mathbf{s}_i^\intercal}{\mathbf{y}_i^\intercal\mathbf{s}_i} + 1] - \frac{1}{\mathbf{y}_i^\intercal\mathbf{s}_i}[\mathbf{s}_i\mathbf{y}_i^\intercal\mathbf{H}_i + \mathbf{H}_i\mathbf{y}_i\mathbf{s}_i^\intercal] \\ =& (\mathbf{I} - \mathbf{p}_i\mathbf{s}_i\mathbf{y}_i^\intercal)^T(\mathbf{I} - \mathbf{p}_i\mathbf{y}_i\mathbf{s}_i^\intercal) + \mathbf{p}_i\mathbf{s}_i\mathbf{s}_i^\intercal \\ =& \mathbf{V}_i^\intercal\mathbf{H}_i\mathbf{V}_i + \mathbf{p}_i\mathbf{s}_i\mathbf{s}_i^\intercal \end{aligned} \tag{38}$$

where $\mathbf{p}_i = \frac{1}{\mathbf{y}_i^\intercal\mathbf{s}_i}$

Expanding the recursion of Eq. (38) yields [29]

$$\begin{aligned} \mathbf{H}_{i+1} =& \mathbf{V}_i^\intercal \dots \mathbf{V}_0^\intercal\mathbf{H}_i\mathbf{V}_0 \dots \mathbf{V}_i \\ & \vdots \\ & + \mathbf{V}_i^\intercal\mathbf{p}_{i-1}\mathbf{s}_{i-1}\mathbf{s}_{i-1}^\intercal\mathbf{V}_i^\intercal \\ & + \mathbf{p}_i\mathbf{s}_i\mathbf{s}_i^\intercal \end{aligned} \tag{39}$$

Rather than considering all updates from 0 to i, L-BFGS uses only the m previous updates. Therefore, for $i > m$, Eq. (39) becomes [29]

$$\begin{aligned} \mathbf{H}_{i+1} =& \mathbf{V}_i^\intercal \dots \mathbf{V}_{i-m+1}^\intercal\mathbf{H}_i\mathbf{V}_{i-m+1} \dots \mathbf{V}_i \\ & \vdots \\ & + \mathbf{V}_i^\intercal\mathbf{p}_{i-1}\mathbf{s}_{i-1}\mathbf{s}_{i-1}^\intercal\mathbf{V}_i^\intercal \\ & + \mathbf{p}_i\mathbf{s}_i\mathbf{s}_i^\intercal \end{aligned} \tag{40}$$

APPENDIX L
K-MEANS CLUSTERING

The K-means algorithm partitions a set of observations $\mathbf{x}_1, \dots, \mathbf{x}_n$ into a set of K clusters $\mathscr{C} = \{C_1, \dots, C_K\}$, where $n \leq K$, s.t. each observation is in exactly one cluster [30, 36]. Formally, the goal is to minimize the Within-Cluster Sum of Squares (WCSS) [30, 36], described in Section 7.2. This objective cannot be minimized directly and is therefore instead minimized using an iterative update algorithm which converges to a local minima, as follows [36]:

Initialize: Pick K observations at random to use as initial clusters means (or centroids) μ_k for $1 \leq k \leq K$.

Assign: Assign each observation $\mathbf{x}_i$ for $1 \leq i \leq n$ to the nearest cluster s.t. $\mathbf{x}_i \in C_k$ if $C_k = \arg\min_{C_k \in \mathscr{C}} ||\mathbf{x}_i - \mu_k||$.
Update: Recalculate the cluster means

$$\mu_k = \frac{1}{|C_k|}\sum_{\mathbf{x} \in C_k} \mathbf{x} \tag{41}$$

The Assign and Update steps are repeated until a maximum number of iterations have been reached or the clusters have reached convergence [36], i.e. the position of any cluster mean does not change within some tolerance level.

APPENDIX M
CLUSTER COHESION AND SEPARATION

This appendix is supplementary to Section 7 and contains information about cluster cohesion in terms of WCSS and cluster separation in terms of BCSS. Fig. 35 shows the WCSS, BCSS, and CHI depending on the number of clusters.

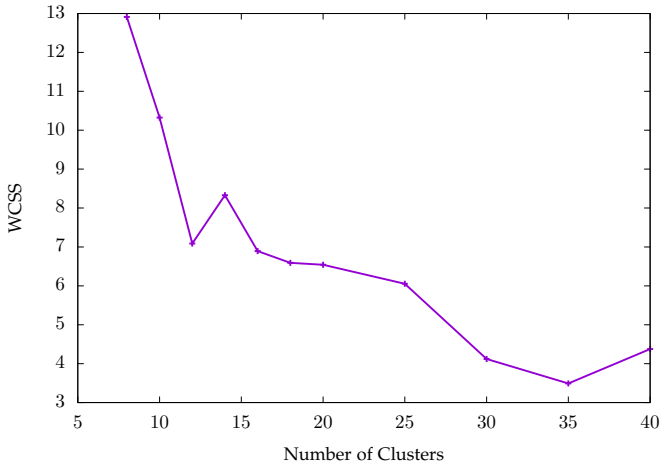
APPENDIX N
ADDITIONAL CLUSTER VISUALISATIONS

This appendix supplements the cluster visualisations in Section 7.4.2.

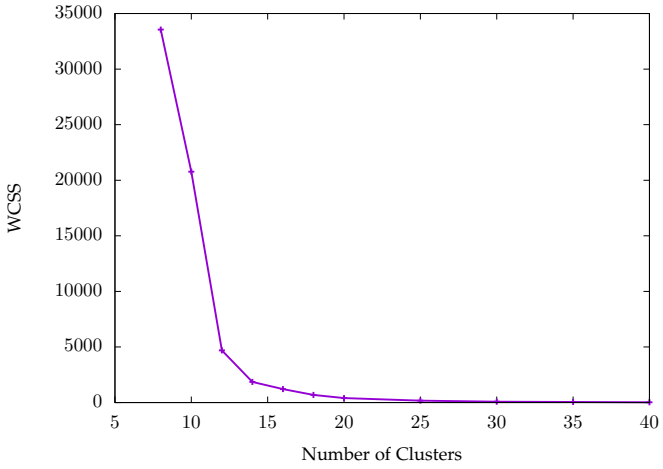
Figs. 36 and 37 show a broad view of the clusters generated by DESCRIPTOR-35 and EDGE+DESCRIPTOR-40, respectively.

Figs. 38 and 39 shows the clustering on the entirety of Aalborg.

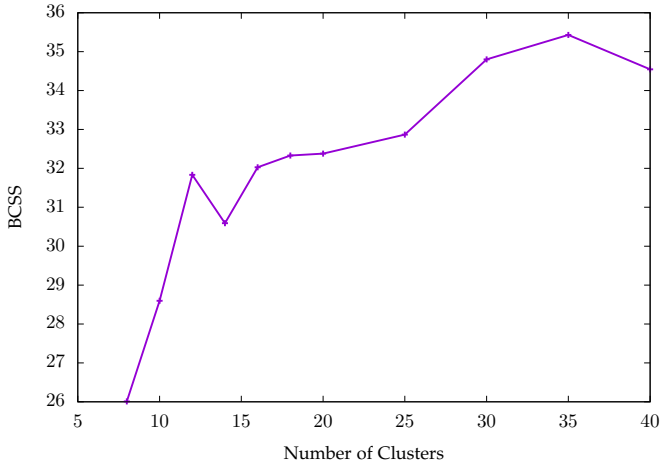
Fig. 40 show the clustering of several smaller towns in Northern Jutland where the road segments belonging to the yellow cluster appear to fulfil a similar role as road segments on the eastern outskirts of Aalborg.



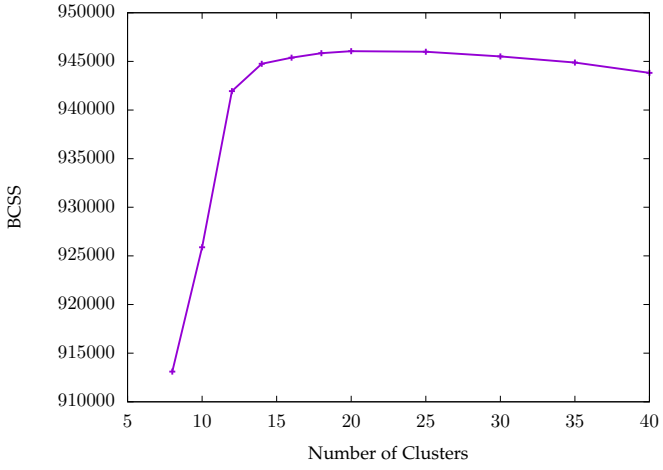
(a) Descriptor



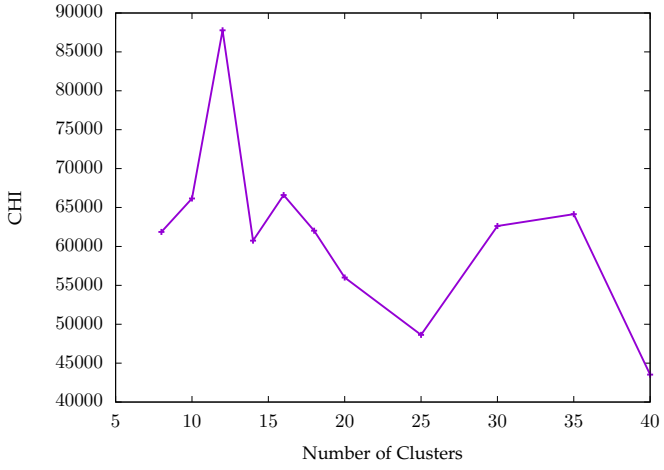
(b) Edge + Descriptor



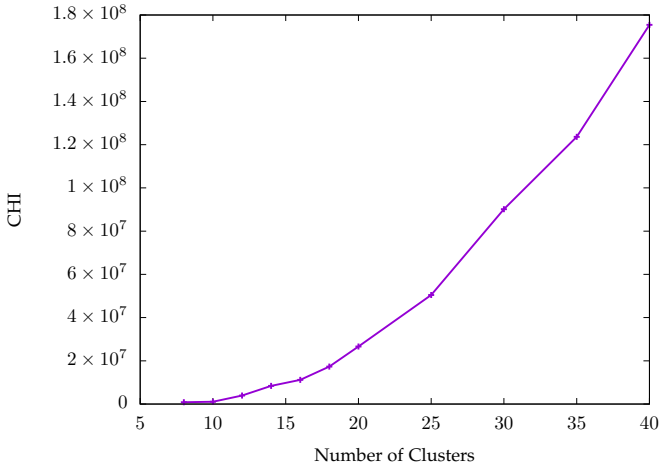
(c) Descriptor



(d) Edge + Descriptor



(e) Descriptor



(f) Edge + Descriptor

Fig. 35: WCSS, BCSS, and CHI depending on the number of clusters for the experiment described in Section 7.

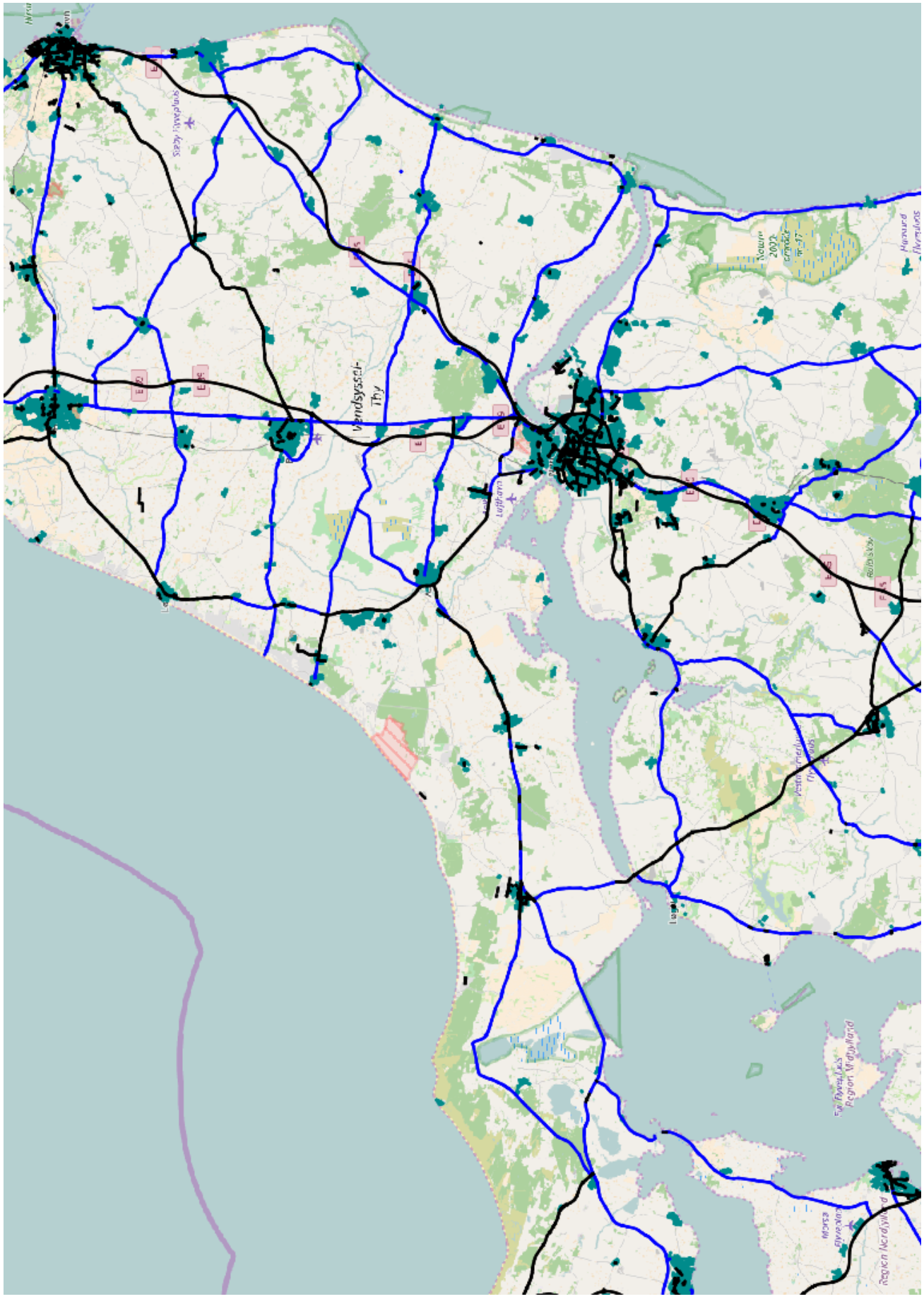


Fig. 36: A broad view showing the clusters of road segments in Northern Jutland generated by DESCRIPTOR-35.

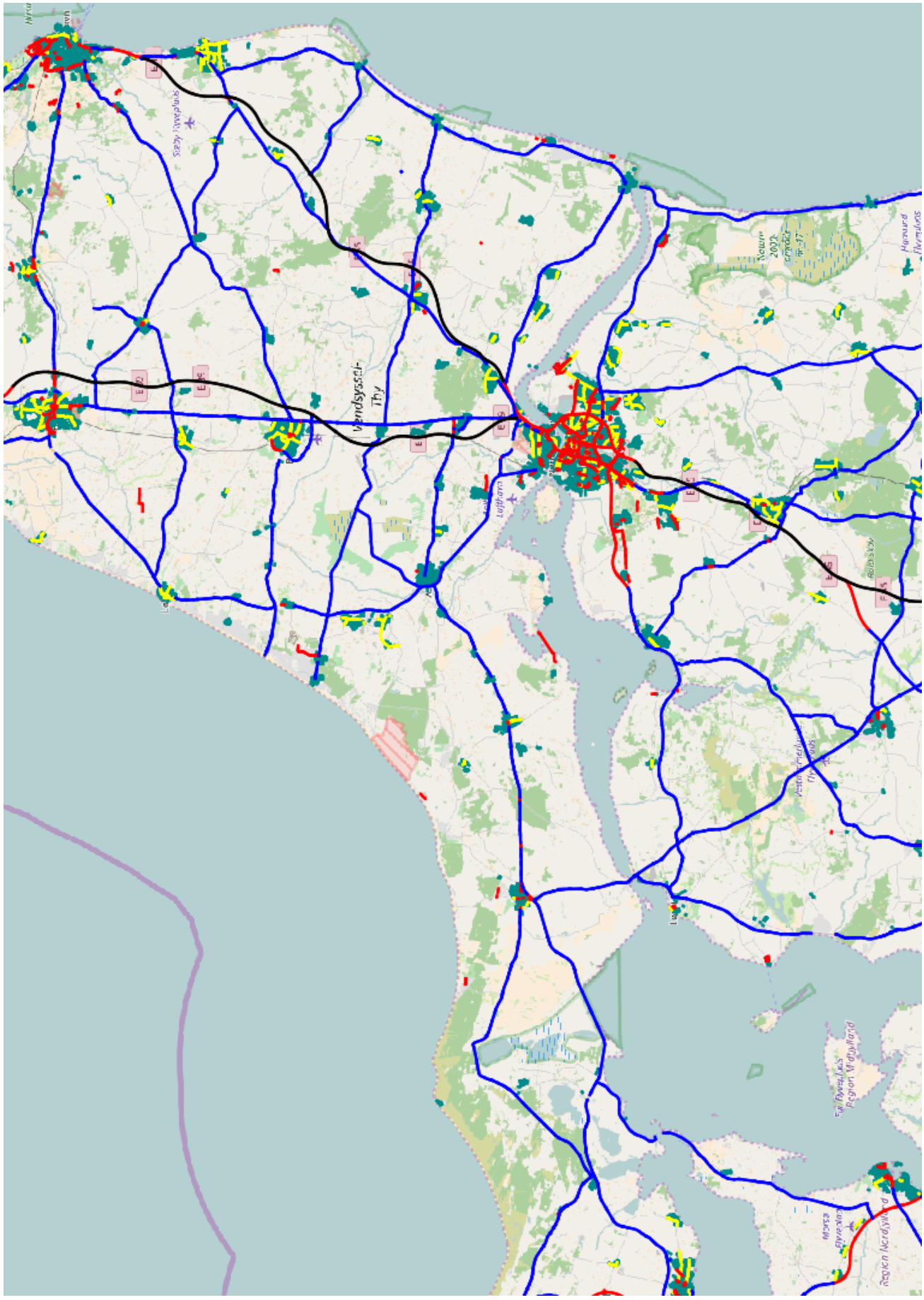


Fig. 37: A broad view showing the clusters of road segments in Northern Jutland generated by EDGE+DESCRIPTOR-40.

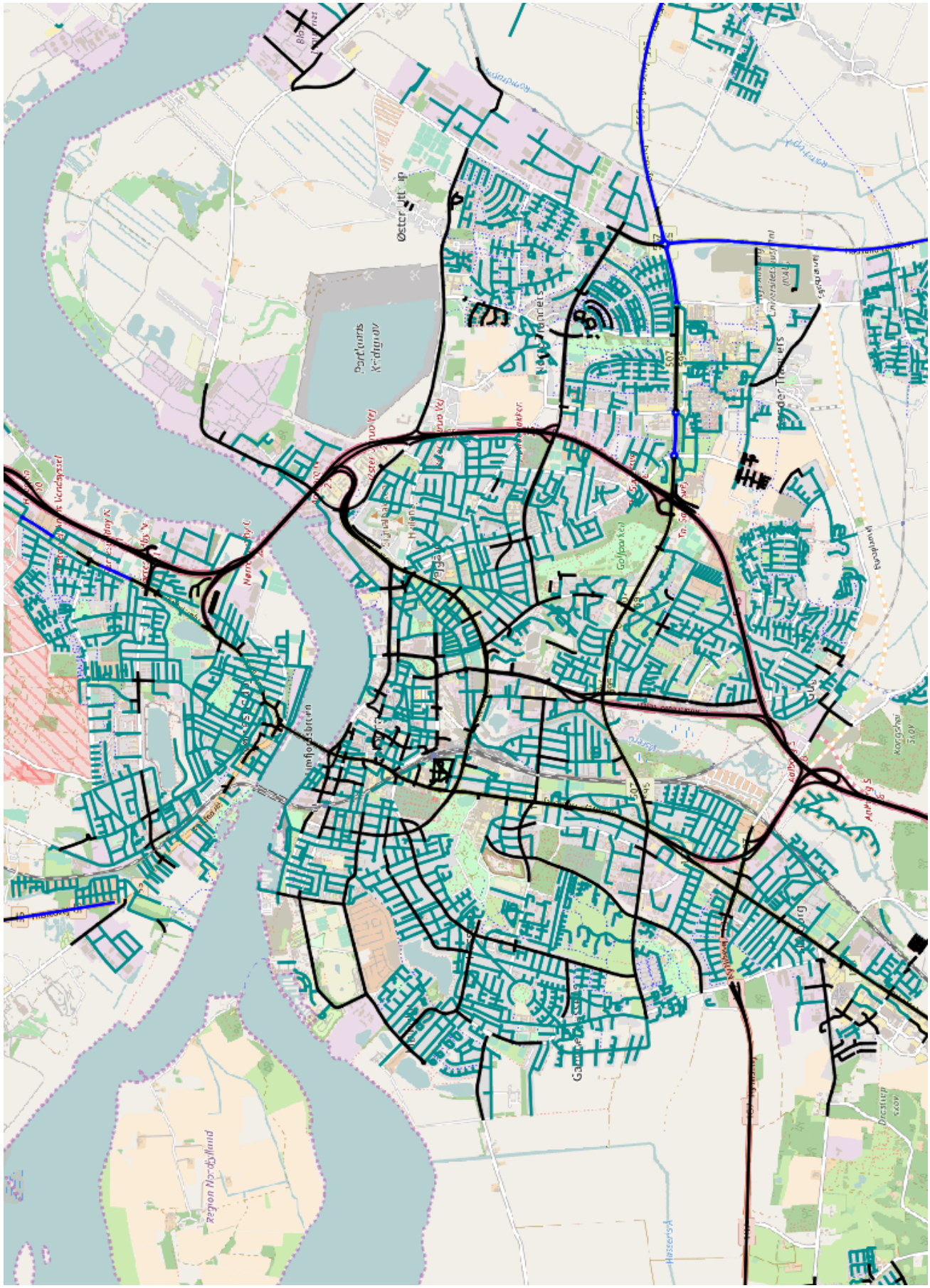


Fig. 38: Visualisations of clusters in Aalborg generated by DESCRIPTOR-35.

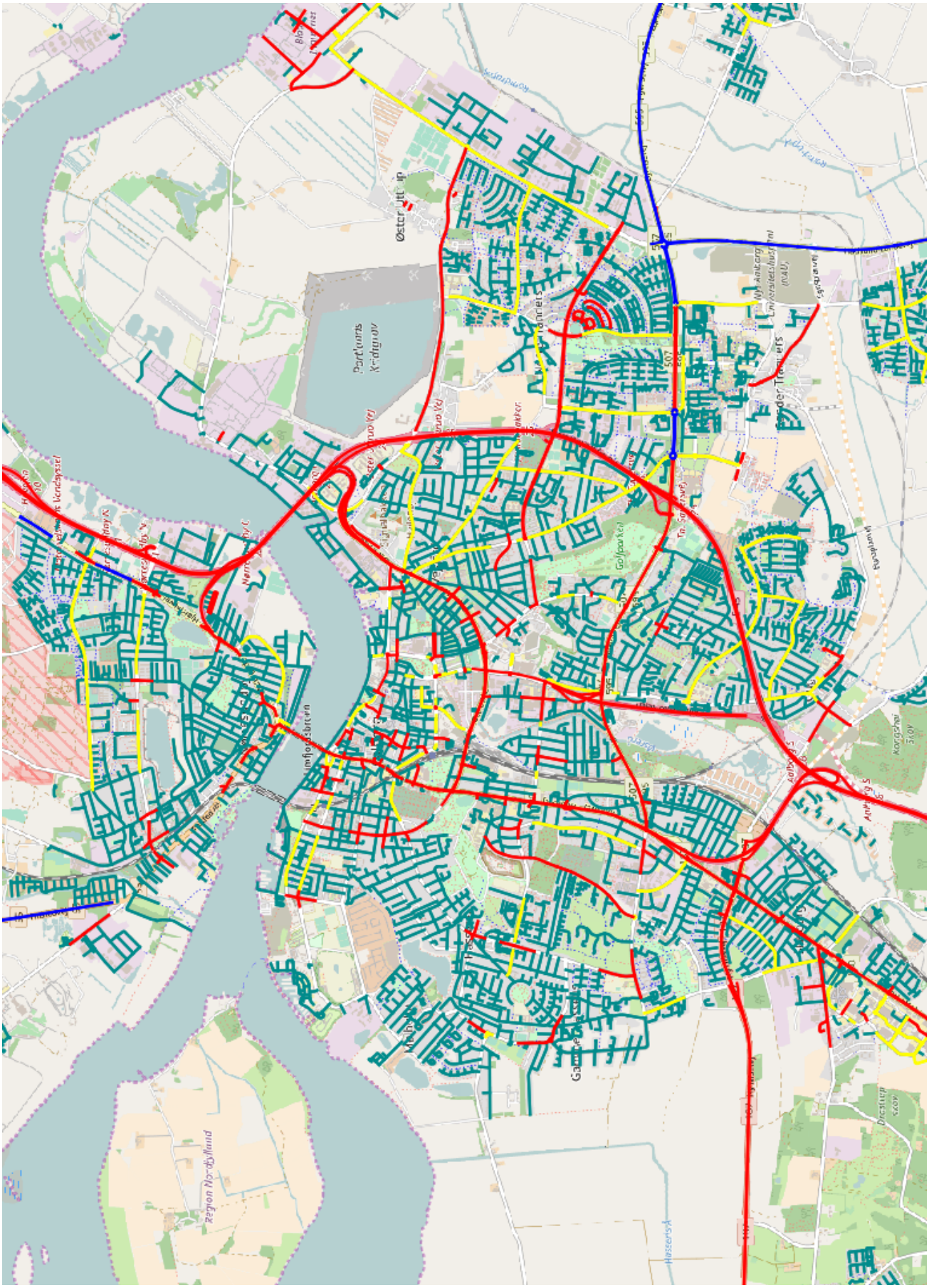
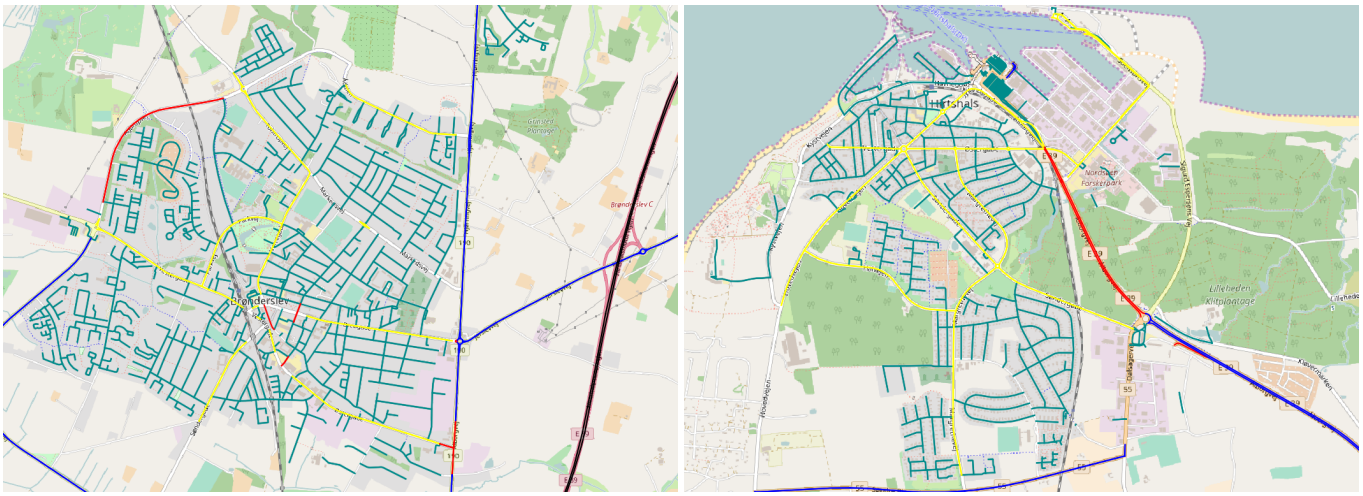
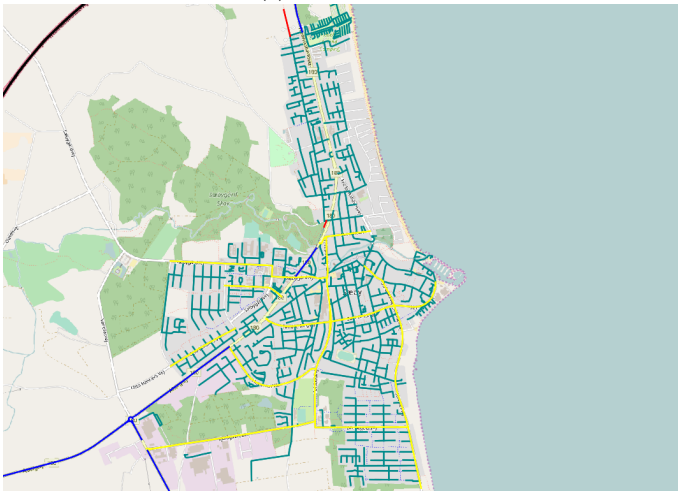


Fig. 39: Visualisations of clusters in Aalborg generated by EDGE+DESCRIPTOR-40.

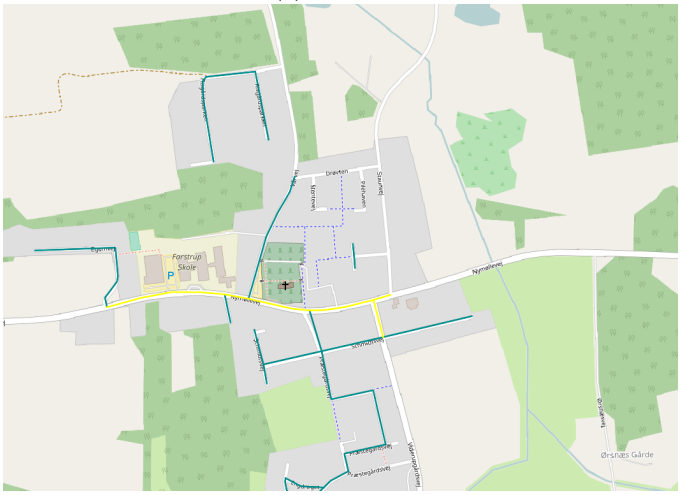


(a) Brønderslev

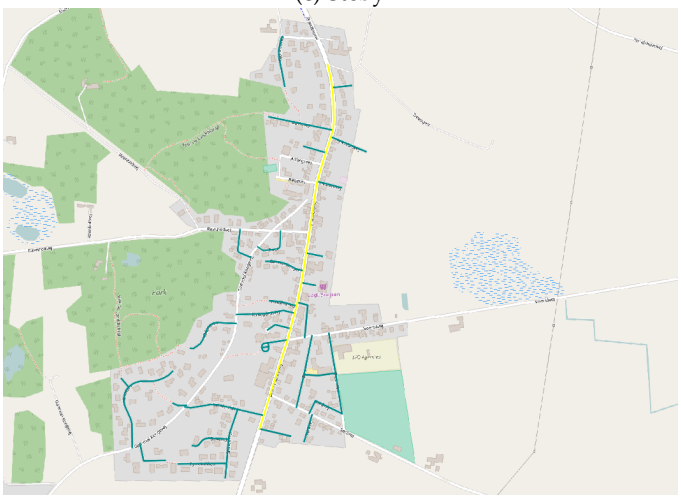
(b) Hirtshals



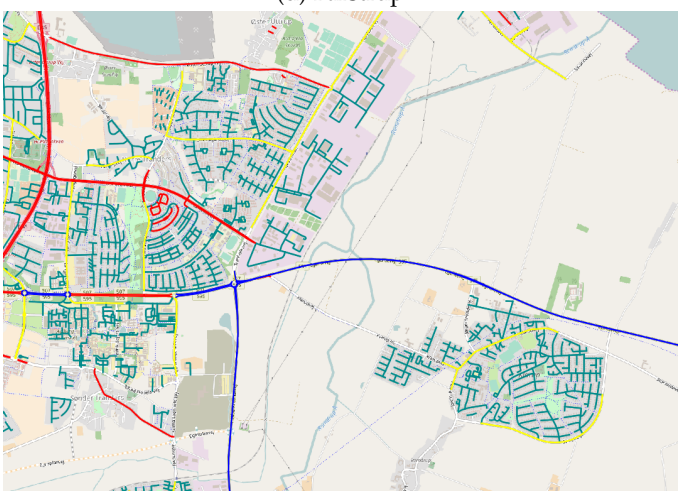
(c) Sæby



(d) Farstrup



(e) Voerså



(f) Eastern Outskirts of Aalborg

Fig. 40: Cluster visualisation of (a, b, c) medium sized towns, (d, e) small towns (f) the eastern outskirts of Aalborg.

REFERENCES

[1] https://www.google.dk/maps. Accessed: 2017-05-6.

[2] https://www.bing.com/maps. Accessed: 2017-05-6.

[3] Bin Yang, Manohar Kaul, and Christian S Jensen. "Using incomplete information for complete weight annotation of road networks". In: *IEEE Transactions on Knowledge and Data Engineering* 26.5 (2014), pp. 1267–1279.

[4] Ove Andersen et al. "An Advanced Data Warehouse for Integrating Large Sets of GPS Data". In: *Proceedings of the 17th International Workshop on Data Warehousing and OLAP*. ACM. 2014, pp. 13–22.

[5] Torp et. al. "The ExpandingSearch Algorithm".

[6] Martin Fruensgaard et al. "Speed Prediction in Road Networks under Conditions of Data Sparsity". Project from pre-specialization. Jan. 2017.

[7] Jiangchuan Zheng and Lionel M Ni. "Time-Dependent Trajectory Regression on Road Networks via Multi-Task Learning." In: *AAAI*. 2013.

[8] Jian Dai et al. "Efficient and accurate path cost estimation using trajectory data". In: *arXiv preprint arXiv:1510.02886* (2015).

[9] Jing Yuan et al. "T-drive: driving directions based on taxi trajectories". In: *Proceedings of the 18th SIGSPATIAL International conference on advances in geographic information systems*. ACM. 2010, pp. 99–108.

[10] Jilin Hu et al. "Enabling time-dependent uncertain eco-weights for road networks". In: *GeoInformatica* 21.1 (2017), pp. 57–88. ISSN: 1573-7624. DOI: 10.1007/s10707-016-0272-z. URL: http://dx.doi.org/10.1007/s10707-016-0272-z.

[11] Tsuyoshi Idé and Masashi Sugiyama. "Trajectory Regression on Road Networks." In: *AAAI*. 2011.

[12] Tsuyoshi Idé and Sei Kato. "Travel-time prediction using Gaussian process regression: A trajectory-based approach". In: *Proceedings of the 2009 SIAM International Conference on Data Mining*. SIAM. 2009, pp. 1185–1196.

[13] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828.

[14] Aditya Grover and Jure Leskovec. "Node2Vec: Scalable Feature Learning for Networks". In: *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 855–864. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939754. URL: http://doi.acm.org/10.1145/2939672.2939754.

[15] Li Deng. "A tutorial survey of architectures, algorithms, and applications for deep learning". In: *APSIPA Transactions on Signal and Information Processing* 3 (2014), e2.

[16] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "DeepWalk: Online Learning of Social Representations". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. New York, New York, USA: ACM, 2014, pp. 701–710. ISBN: 978-1-4503-2956-9. DOI: 10.1145/2623330.2623732. URL: http://doi.acm.org/10.1145/2623330.2623732.

[17] Xiaofei Sun et al. "A General Framework for Content-enhanced Network Representation Learning". In: *CoRR* abs/1610.02906 (2016). URL: http://arxiv.org/abs/1610.02906.

[18] Annamalai Narayanan et al. "subgraph2vec: Learning Distributed Representations of Rooted Sub-graphs from Large Graphs". In: *CoRR* abs/1606.08928 (2016). URL: http://arxiv.org/abs/1606.08928.

[19] Kevin P Murphy. "Conjugate Bayesian analysis of the Gaussian distribution". In: *def* $1.2\sigma 2$ (2007), p. 16.

[20] Quoc Le and Tomas Mikolov. "Distributed Representations of Sentences and Documents". In: *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*. Ed. by Tony Jebara and Eric P. Xing. JMLR Workshop and Conference Proceedings, 2014, pp. 1188–1196. URL: http://jmlr.org/proceedings/papers/v32/le14.pdf.

[21] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[22] *OpenStreetMap*. URL: http://www.openstreetmap.org/.

[23] *PlansystemDK*. URL: https://erhvervsstyrelsen.dk/plansystemdk.

[24] Nornadiah Mohd Razali, Yap Bee Wah, et al. "Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests". In: *Journal of statistical modeling and analytics* 2.1 (2011), pp. 21–33.

[25] Laurens van der Maaten. "Accelerating t-SNE using Tree-Based Algorithms". In: *Journal of Machine Learning Research* 15 (2014), pp. 3221–3245. URL: http://jmlr.org/papers/v15/vandermaaten14a.html.

[26] Xin Rong. "word2vec parameter learning explained". In: *arXiv preprint arXiv:1411.2738* (2014).

[27] DG Mayer and DG Butler. "Statistical validation". In: *Ecological modelling* 68.1-2 (1993), pp. 21–32.

[28] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. Springer series in statistics Springer, Berlin, 2001, p. 349.

[29] Jorge Nocedal. "Updating quasi-Newton matrices with limited storage". In: *Mathematics of computation* 35.151 (1980), pp. 773–782.

[30] James MacQueen et al. "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297.

[31] Tadeusz Caliński and Jerzy Harabasz. "A dendrite method for cluster analysis". In: *Communications in Statistics-theory and Methods* 3.1 (1974), pp. 1–27.

[32] Glenn W Milligan and Martha C Cooper. "An examination of procedures for determining the number of clusters in a data set". In: *Psychometrika* 50.2 (1985), pp. 159–179.

[33] Jennifer Neville and David D. Jensen. "Iterative Classification in Relational Data". In: *Proceedings of the Workshop on Learning Statistical Models from Relational Data, Seventeenth National Conference on Artificial Intelli-*

gence. Ed. by Lise Getoor and David D. Jensen. Austin, TX: AAAI Press, Menlo Park, CA, 2000, pp. 42–49.

[34] *Denmark*. URL: http://wiki.openstreetmap.org/wiki/OSM_tags_for_routing/Maxspeed#Denmark.

[35] Joseph-Frédéric Bonnans et al. *Numerical optimization: theoretical and practical aspects*. Springer Science & Business Media, 2006, pp. 51–52, 55.

[36] Greg Hamerly and Charles Elkan. "Alternatives to the k-means algorithm that find better clusterings". In: *Proceedings of the eleventh international conference on Information and knowledge management*. ACM. 2002, pp. 600–607.