Can one use Graph Kernels for data selection ?

EE452 - Networks Machine learning project
Yixuan Xu, Justin Deschenaux

September 10, 2023

Abstract

When training a machine learning model, utilizing high-quality training data is crucial for achieving optimal performance. In this project, our objective is to explore methods for selecting the most pertinent data for congestion modelling and link prediction for city networks. We investigate the viability of employing graph kernels as a distance metric between graphs to filter out relevant data.

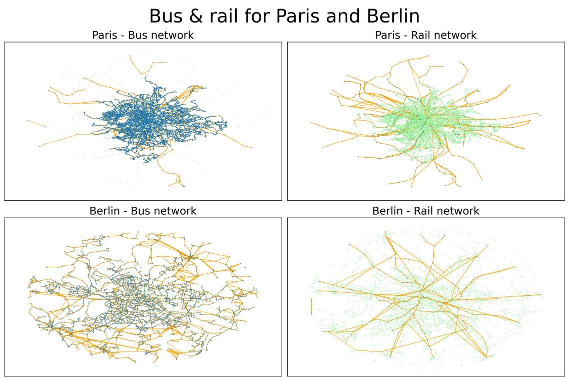


Figure 1: Comparing the topologies for the bus and rail public transports of Paris and Berlin. The blue nodes have positive degree (e.g. bus stops for the bus network) while the green ones have no edges (e.g. bus stops for the rail network). One can see that most edges are part of the bus network

1 Data exploration

We work on data for 25 cities. To maintain readability, we intentionally include only a few plots and numerical values in this report, while providing comprehensive details in the accompanying notebooks.

1.1 Dataset

In order to carry out experiments, we utilize the dataset proposed by Kujala et al. [7]. It comprises public transport (PT) networks obtained from processing the data of 25 cities. The primary objective of [7] was to provide a dataset encompassing diverse types of cities in terms of size, continent, and structure. Unfortunately, it does not include cities from South America, Asia, or Africa. The authors explained that the selection process was heavily influenced by whether city's PT agencies provided any data along clear licensing to allow research use cases.

Furthermore, the original data was available in the GTFS format, which is not widely known among data scientists. They extracted and standardized the data across various cities while also rectifying errors. Additionally, they incorporated temporal information pertaining to each route, specifically indicating the departure and arrival times of public transport vehicles at stations. It is important to note that these time-tables reflect the scheduled timings rather than the actual duration experienced by users in the real-world. In addition to the official PT modes from the GTFS format, the authors augmented the dataset by including the walking distance between nearby stops. This was achieved by extracting relevant details from OpenStreetMap. Such information is relevant when modeling users walking from one stop to an other.

We also note that the networks are typically not strongly connected. While it might look surprising at first, it is a direct consequence of the data preprocessing. Indeed, authors of [7] extracted

City	Tram	Subway	Rail	Bus	Ferry
Paris	2.41	4.92	7.40	85.27	0.00
Berlin	8.20	3.03	4.20	84.51	0.07

Table 1: Comparing the percentage of edges in each public transport network (walk excluded) for Paris and Berlin. We see that the bus network is the most developed one.

sub-regions centered around cities from the original complete PT networks. Imagine nodes a and b were kept but were only connected through a vertex c that was discarded, then, they will not be connected in the processed dataset as all edges pointing from and to excluded nodes were discarded. This is important to keep in mind for the exploitation part, as it might deteriorate performance at the border of networks.

Excluding the edges representing walk paths, we observed that in any city, at least 84 % of all PT edges represent bus paths. Moreover, for 6 cities (Detroit, Turku, Kupio, Palermo, Winnipeg and Belfast), only the bus mode is available. Therefore, we focused on this sub-network as it captures most of the combined PT networks structure and allows easier comparison between cities. Table 1 shows a typical edge density distribution for the cities of Paris and Berlin.

With regard to the project, the most important features are located on the edges and correspond to the average duration of a PT vehicle, the physical distance between endpoints and the number of vehicles that travelled on the edge over the data collection period.

1.2 Network model

Our investigation focuses on determining the adequacy of standard network models in approximating the city networks extracted from Kujala et al. [7]. In pursuit of this objective, we specifically consider two prominent models: random and scale-free networks. It is important to note that a significant caveat of our analysis is the orientation of the networks (class discussions only covered undirected networks).

We initiated our analysis by examining the degree distribution of each city, and upon plotting it

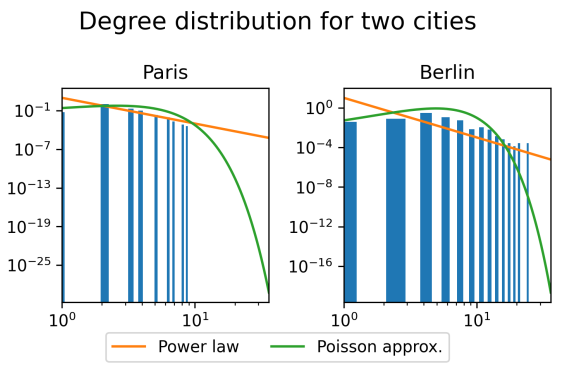


Figure 2: Comparing the degree distribution against a power-law and a Poisson distribution for two cities with lighter and heavier tail.

on a log-log scale, we observed a near-linear pattern. It is important to recall that for a network with a power-law degree distribution $p(k) = Ck^{-\gamma}$, there exists a linear relationship between the x and y axes: $\log(p(k)) = -\gamma \log(k) + \log(C)$. Empirically, we found that while a value of $\gamma = 4$ aligns with the trend for most cities, the empirical distribution's tail diminishes too rapidly to exhibit a power-law behavior. Additionally, an ideal scale-free network with such a high value of γ is adequately approximated by a random network due to the absence of hubs. Hence, we further investigated whether the Poisson approximation of the random network's distribution is suited. We used $\mu = \langle k \rangle$ as parameter of the Poisson distribution and extended its domain to the real numbers by using the gamma function instead of factorials for smoother visualization. Once again, we observed that the tail of the empirical distribution dies too quickly. See figure 2 for the cities of Paris and Berlin. Despite Berlin being slightly fatter-tailed than Paris, both distribution are not well approximated by power-laws as their support is way too small. The Poisson distribution is a better fit for Berlin but doesn't work for Paris.

We proceeded to examine other pertinent network statistics, namely the clustering coefficient, average distance (for small and ultra-small world properties), and average squared degree. First, we observed that, except for Luxembourg and Berlin, all cities exhibited values of $\langle k^2 \rangle < 2 \cdot \langle k \rangle$, with an average degree $\langle k \rangle$ below 1.5. Such close and relatively small values of $\langle k^2 \rangle$ are indicative of in random networks (or scale-free with large γ). Additionally, we noted a remarkably accurate fit between the empirical average distance and the average distance expected in Erdős–Rényi (ER) networks of similar size and average degree ($\frac{\ln N}{\ln \langle k \rangle}$). In contrast, the clustering coefficient consistently exceeded the expected values of the random model and closely resembled those of a Barabási–Albert (BA) network ($\frac{\ln N}{\ln \ln N}$). The observed clustering coefficient values deviated from the BA value by no more than half an order of magnitude for 20 out of the 25 cities.

Consequently, we believe that neither of the models are a suitable fit for the cities. Many natural phenomena give rise to scale-free networks, and we hoped to observe this characteristic in the dataset. However, the networks are man made. Stations are not connected randomly; rather, efficiency dictates that connections should include intermediate stops rather than long stretches without halts. This leads to each node being connected solely and more tightly to physically adjacent locations, which, in turn, explains the absence of hubs, low average degree, high average distance, and relatively high clustering coefficient (not absolutely high, but according to a scale-free model with similar number of nodes). From an exploitation perspective, our key takeaway is that the high average distance may pose challenges for message passing neural networks (MPNN) in propagating global information effectively. Consequently, incorporating global attributes (or a master node) could be vital for graph classification tasks and prove beneficial for others as well.

1.3 Spectral analysis

One can extract useful representations from the eigen-decomposition of the Laplacian. It is important to note that the Laplacian we studied in class is designed for undirected graphs, whereas we are working with directed edges. To address this, a simple solution is to symmetrize the adjacency matrix before conducting spectral analysis. Consequently, in the initial step, we computed the symmetrized and normalized Laplacian for the bus network, considering only the nodes with non-zero degree. Such nodes exist for the other (not bus) PT modes only.

We initially considered retaining only the largest strongly connected component (SCC) for each city, excluding the border nodes that are disconnected due to the original dataset preprocessing. Our rationale was that it might yield better results, as borders do not accurately reflect the characteristics of a real PT network due to the manual pruning step. However, in certain cities like Lisbon, the largest SCC of the original graph comprises only 36% of all nodes, resulting in significant graph loss if we were to pursue this approach. Consequently, we discarded nodes with degree zero in the symmetrized graph.

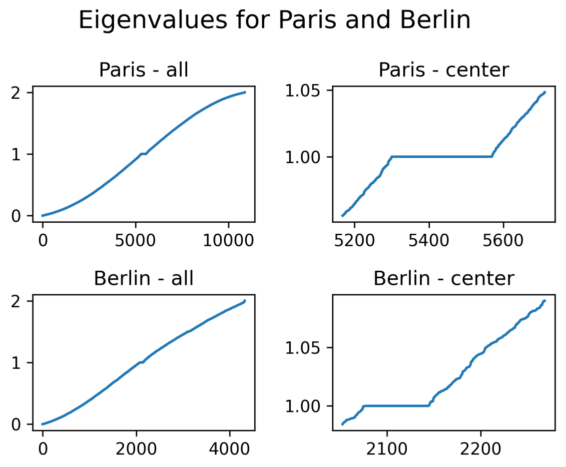


Figure 3: Eigenvalues of the symmetrized and normalized graph Laplacian for the cities of Paris and Berlin. When zooming in the middle of the curve, one can see that it is not completely smooth. All cities exhibit this pattern and have many eigenvalues close to one. We did not find a satisfying explanation for this phenomenon.

When plotting the eigenvalues for each city, we observed that it looked quite differently from the plot seen during the class, where the eigenvalues increased much faster before plateauing close to the maximal value (e.g. for the Stanford bunny graph). We do not have an explanation so far as for why it is the case. See Figure 3 for the phenomenon for Paris and Berlin.

2 Exploitation part

2.1 Overview

As indicated in the abstract, our objective is to assess the feasibility of extracting subgroups of structurally similar graphs. Our exploration of comparison methods for graphs began with a comprehensive survey presented in [13]. This survey describes several techniques, including some using Optimal Transport as demonstrated in [8]. Regrettably, numerous methods, including [8], make the assumption that the graphs being compared have an identical number of vertices, which is too limitations.

We encountered references to the utilization of optimal transport for comparing graphs of different sizes within the domain of neural architecture search, as highlighted in [6]. Their approach represents an architecture as a directed graph connecting layers, leveraging information about each computation unit (e.g. convolution or dense layers), such as the number of parameters, to facilitate graph matching. Unfortunately, this methodology cannot be directly applied to our scenario since our graphs lack comparable labels.

Our expectation was to discover an unsupervised method for comparing graphs of varying sizes. The most closely aligned contender in this regard are the graph kernel methods, extensively reviewed in [9].

It is challenging for humans to visually determine the similarity between arbitrary graphs, as it deviates from the regular priors ingrained in our cognitive processes. In contrast, when dealing with images, for instance, our intuition tends to be more effective. For example, if an image classifier performs poorly on certain images, we can easily formulate possible explanations such as inadequate training data for specific conditions. A self-driving system trained solely on sunny images, for instance, is likely to struggle when faced with night images. Therefore, we believe our question is very relevant to enhance the efficacy of machine learning models. An alternative solution would be to perform unsupervised data augmentation (e.g. see [11] for images). We however believe that no clearly beneficial method exists for graphs.

Formal setting Given a source graph $\mathcal{G}$ and a collection of graphs $\mathcal{H}_1, \mathcal{H}_2, ..., \mathcal{H}_n$ all representing various cities, our objective is to extract a subset $\mathcal{H}_{i_1}, \mathcal{H}_{i_2}, ...\mathcal{H}_{i_k}$ such that a model trained on this subset outperforms a model trained on the entire set $\mathcal{H}_1, \mathcal{H}_2, ..., \mathcal{H}_n$. Our intention is to leverage graph kernels as a means to compare graphs and see whether classical machine learning methods and graph neural networks (GNN) achieve better performance than when trained on all cities.

2.2 Data preparation

As observed in the exploration section, we believe the most relevant public transport to analyze is the bus network, since it is available in all cities and amounts to most edges of all networks (when excluding walk). Therefore, to keep things simple, we discard all edges and vertices from other modes of transportation.

Our machine learning models are exclusively trained on handcrafted features, intentionally avoiding the use of learned node embeddings. The primary reason for this choice is to ensure the non-transductive nature of our method, enabling us to apply our model effectively to new graphs. While certain domains, such as molecules as exemplified in [3], may possess pre-existing features attached to their nodes, this is not the case for us. As a result, we rely on three distinct sources: node-level statistics, spectral properties (eigenvectors), and Node2Vec features.

These three sources are concatenated to form the node-level representation. Before feeding these features into our models, we standardize them. Specifically, we normalize each training feature by subtracting the mean and dividing by the standard deviation. This normalization process is crucial in mitigating biases arising from differing magnitudes.

Node-level statistics We carefully selected relevant metrics, including degree, average degree of neighbors, clustering coefficient, PageRank coefficient, and centrality measures such as betweenness, harmonic, and closeness centralities. These statistics hold relevance and can be computed for directed graphs.

Spectral features We perform eigendecomposition of the symmetrized and normalized graph Laplacian, retaining the 100 eigenvectors associated with the smallest eigenvalues, following the approach employed in previous assignments.

Node2Vec We leverage Node2Vec [5] to extract structural properties of the graph in an unsupervised manner. We adopt three parameter pairs (p, q): $(1, 1)$ represents a simple random walk (RW), $(1, 2)$ corresponds to a breadth-first-search-like RW, and $(2, 1)$ denotes a depth-first-search-like RW. Although these configurations may appear simplistic, we deliberately adhere to basic settings as our primary focus lies not in hyper-optimizing the exact feature extraction parameters for achieving state-of-the-art performance. We extract 50 features per node from walks of length 80, with a skip-gram window of size 5.

Train, validation, test split In order to assess the performance of our models, we selected the cities of Paris and Berlin for evaluation, as we observed distinct structural characteristics during the exploratory phase. To begin, we tune hyperparameters as follows: for a given city (e.g., Paris), we employed a distance measure (details of the kernel will be elucidated later) to compute the distances between this city and all others. Subsequently, we sorted the cities in descending order of their distances. From this ranked list, we extracted the top 5 closest cities, employing the bottom 4 as training data while reserving the top one for evaluation. Finally, we evaluated the performance of our data selection policy on the city of interest itself (e.g., Paris).

As a baseline for comparison, we trained models using all cities except the city of interest, employing identical hyperparameters as found previously. Note that the hyperparameters were tuned separately for each model architecture and task.

2.3 Tasks

Average travel time regression The first task we undertake is the prediction of average travel time along each edge. Accurately estimating the time of arrival (ETA) holds paramount importance across numerous industries, with Google Maps employing Graph Neural Networks (GNNs) to enhance its results, as highlighted in [2]. However, for the sake of simplicity, we concentrate on the comparatively easier task of predicting the average value for individual edges. We train our model to minimize the mean squared error (MSE) and mean absoluve error (MAE) over all edges.

Link prediction Our aim is to detect the presence or absence of edges. Since most industrial application of GNN are link predictions (recommender systems, duplicate account detection, cross-platform user recognition), we deemed it pertinent to consider this task. As observed in the exploration part, our graphs are very sparse, hence the dataset is highly unbalanced and an easy solution to obtain almost perfect accuracy is to always predict there is no edge between nodes. We include negative samples in the training procedures. For a graph with k existing edges, we include $20 \cdot k$ nonexistent edges. Our rationale was to include as many non-existent edges as we could, so that GCP's Colab machines do not crash, as we did not manage to to fit the graphs with all potential edges (fully connected, existing edges labelled with 1 and nonexistent labelled as 0). We train our models to maximize the accuracy and F1-score of the binary classification problem.

2.4 Graph kernels

In order to compute graph kernels, we use the library GraKeL (https://ysig.github.io/GraKeL/0.1a8/) and use the graphlet, shortest path and random walk kernels. GraKeL also allows for combining kernels and does it in the following way: each kernel specified in the kernel list operates independently on the input graphs and produces a kernel matrix. The final kernel in this list serves a special role as the base kernel. This base kernel functions on top of the previously computed kernels, executing comparisons across them. The outcome of this process is then normalised, so that the kernel value of a graph compared to itself is always 1, thus standardizing the range of potential outcomes.

Graphlet kernel The graphlet kernel operates by counting the frequency of occurrences of predetermined small sub-graphs present in a given graph $\mathcal{G}$. The main problem of this method is its computational complexity. For a given graphlet with k

vertices, a naive counting of all its occurences require checking $\binom{n}{k}$ set of nodes, for a grah of size n. Fortunately, as proved in [10], it is possible to approximate the true graphlet kernel by sampling a much smaller number of k-tuples of nodes from the graph.

Geometric random walk The random walk kernel works by coupling two graphs $\mathcal{G}_1 = (V_1, E_1)$ and $\mathcal{G}_2 = (V_2, E_2)$ into $\mathcal{G}_\times = (V_\times, E_\times)$ where

$$V_\times := \{(u, v) : u \in V_1 \text{ and } v \in V_2\} \tag{1}$$

and

$$\begin{aligned} E_\times := \ &\{((u_i, u_j), (v_i, v_j)) : \\ &(u_i, u_j) \in E_1 \text{ and } (v_i, v_j) \in E_2\} \end{aligned} \tag{2}$$

Finally, the kernel between two graphs is computed as

$$\begin{aligned} K_\times^\infty(G_i, G_j) &= \sum_{p,q=1}^{|V_\times|} \Big[\sum_{l=0}^{\infty} \lambda^l A_\times^l\Big]_{pq} \\ &= e^T (I - \lambda A_\times)^{-1} e \end{aligned} \tag{3}$$

where e is the full-one vector, λ a positive real scalar and $A_\times$ is the adjacency matrix of the combined graph $\mathcal{G}$. We included the geometric random walk kernel in our experiments because it is one of the most well-studied graph kernel.

Shortest path The shortest path kernel operates by considering all pairs of nodes in a given graph $\mathcal{G} = (V, E)$ and calculating the shortest path between them. It uses the intuition that two graphs may be considered similar if they have many similar shortest paths.

Let $d_G(u, v)$ denote the shortest path between nodes u and v in graph $\mathcal{G}$. The shortest path kernel between two graphs $\mathcal{G}_1 = (V_1, E_1)$ and $\mathcal{G}_2 = (V_2, E_2)$ is then defined as:

$$K_{sp}(G_1, G_2) = \sum_{u,u' \in V_1} \sum_{v,v' \in V_2} \delta(d_{G_1}(u, u'), d_{G_2}(v, v')) \tag{4}$$

where $\delta(x, y)$ is the Kronecker delta function, which is 1 if $x = y$ and 0 otherwise. In other words, for each pair of nodes in $\mathcal{G}_1$, we sum the number of pairs of nodes in $\mathcal{G}_2$ that have the same shortest path length.

2.5 Models

Recall that we want to figure out whether one can use Graph Kernel methods to filter the most relevant data to train machine learning models. We want to study whether the results are model dependent, hence we consider multiple types of models. We consider both classical methods as well as GNNs. For average travel time regression, we consider XGBoost regressors [1], a Graph convolution network (GCN) and a Graph Attention Network (GAT) based network. For link prediction, we use XGBoost predictors [1] and the same two neural networks architectures. We tune each model's hyperparameters using random search. A priori, since we work with handcrafted features, tree-based models might work better than deep-learning methods [4]. We specialize the GNNs models for classification or regression in the loss and evaluation functions, that is, we use unnormalized outputs for both, and the classification model uses binary cross-entropy with logits. For XGBoost, the predictor is able to handle the data by default

Graph Convolution Network We implemented a basic GCN, which comprises three graph convolutional layers, each succeeded by an Exponential Linear Unit (ELU) activation function. This architecture allows the processing of node features while capturing the local structure of the graph through each convolutional layer. These processed node features are then rearranged and concatenated for each respective edge, thus transforming them into edge features. These edge features are then channeled through a linear classifier. This final classifier serves as a mapping function, transitioning the high-dimensional edge features to their respective target classes.

Graph Attention Network (GAT) We also implemented a more advanced neural network, GAT, followed the architecture from the original GAT paper[12]. The architecture comprises three graph attention layers, each equipped with distinct head configurations: the first two layers incorporate four attention heads each, while the final layer expands to utilise six attention heads. As recommended by the foundational GAT research, our model also leverages skip connections, a mechanism to facilitate the flow of information and mitigate the vanishing gradient problem. Post-processing, the output of the last graph attention layer is rearranged and concatenated, subsequently being channeled through a linear classifier for final output generation.

3 Experiments

In our research for average travel time regression, we focused on the cities of Berlin and Paris. We evaluated the performance of three different models — XGB, GCN, and GAT — and compared their predictive capabilities. Our comparison focused on two primary evaluation metrics: Mean Squared Error (MSE) and Mean Absolute Error (MAE).

A critical step in our methodology was the use of graph kernel measures to identify cities most similar to our test cases, Berlin and Paris. This similarity assessment served as the foundation for our model training strategy. Based on the graph kernel scores, we selected the five cities that demonstrated the highest similarity to the city under examination.

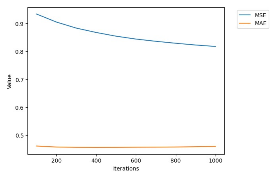
Subsequently, we partitioned this subset of cities further for model validation and hyperparameter tuning. The city deemed most similar was set aside for validation, while the remaining four were utilised in the hyperparameter optimisation process.

Once the hyperparameters were tuned on the most similar cities, we proceeded with the training phase. We developed baseline GCN and GAT models by training them on all cities, excluding the one under test (either Berlin or Paris). Following this, we compared the performance of these baseline models with models trained exclusively on the top five similar cities as determined by our earlier graph kernel analysis.

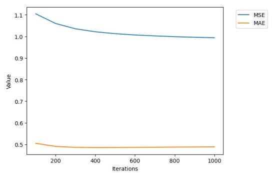
As explained earlier, we compare those GNNs with classical Machine Learning methods, namely gradient-boosted tree methods [1], a popular framework used for both regression and classification tasks.

Upon assessing our models, the findings—illustrated in Table 2 provided insightful revelations. Specifically, GNNs showed promising results, especially when trained on data from cities similar to the test city. Our GAT distinguished itself by registering the lowest MSE values. The other models demonstrated comparable performance, with values predominantly around the 0.8 mark. A similar trend was also observed for the MAE values.

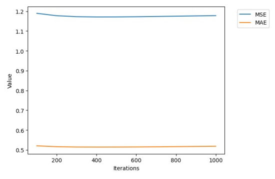
Regarding the task of link prediction in our research, the GNNs unfortunately underperformed. However, the XGB model also displayed disappointing F1-scores during testing as shown in Table 3, indicating a potential issue with the quality of the features we generated. This also suggests the effectiveness of using graph kernels for data selection may vary based on the specific task at hand.



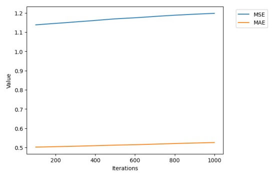
(a) GCN Ours



(b) GCN Baseline

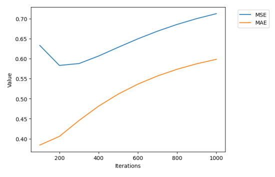


(c) GAT Ours

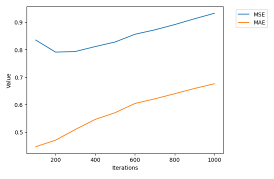


(d) GAT Baseline

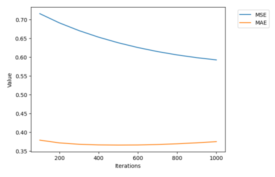
Figure 4: Evaluation metrics for Average Time Prediction for Paris



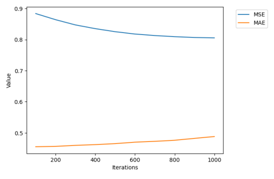
(a) GCN Ours



(b) GCN Baseline



(c) GAT Ours



(d) GAT Baseline

Figure 5: Evaluation metrics for Average Time Prediction for Berlin

It's also important to note that the method of

	Berlin			Paris		
Metrics	XGB	GCN	GAT	XGB	GCN	GAT
MSE (Baseline)	0.871	-	-	0.654	-	-
	0.890	0.817	0.801	1.163	1.276	1.197
MSE (Ours)	0.512	-	-	0.425	-	-
	0.845	0.821	0.576	1.240	0.993	1.178
MAE (Baseline)	0.448	-	-	0.395	-	-
	0.601	0.460	0.490	0.535	0.609	0.526
MAE (Ours)	0.307	-	-	0.354	-	-
	0.692	0.468	0.405	0.567	0.489	0.518

Table 2: This table presents a comparison of evaluation metrics used to predict average bus travel times. These predictions are made using both traditional ML approach and Graph Neural Networks (GNNs). The columns in the table correspond to different methods: "XGB" signifies the XGBoost tree regressor, "GCN" denotes the graph convolutional network, and "GAT" stands for the graph attention model. Row colors provide additional context: grey rows show the evaluation metrics derived from training data, while white rows represent metrics obtained from validation or test data.

Metrics	Berlin	Paris
Accuracy (Baseline)	0.931	0.931
	0.908	0.908
Accuracy (Ours)	0.959	0.957
	0.910	0.908
F1-score (Baseline)	0.444	0.444
	0.155	0.155
F1-score (Ours)	0.743	0.723
	0.097	0.134

Table 3: This table presents a comparison of evaluation metrics used for link prediction, we only listed results for classical ML XGB models since the GNNs did not work properly.

training on the five cities that are most similar to the test subject could potentially introduce overfitting. This potential pitfall becomes notably apparent for simpler networks, such as the basic GCN we implemented, as evidenced in Figure 5.

3.1 Further Directions & Discussion

Regarding the exploration, we did not find a suited network models. Given more time, we would have investigated the lattice model as it encompasses the fact that vertices only connect with local neighbours.

Regarding the exploitation part, although our study has achieved encouraging results in predicting average travel times, there is a need to further validate the applicability of our approach. Moreover, given that our models' performance appears to be task-dependent, we should apply the same approach to various other datasets to draw further conclusions.

The most obvious potential pitfall is the use of pre-computed features. Since our networks did not have structural features by default, we had to rely on handcrafted ones, such as the ones we saw in class. Other datasets, such as molecule ones (see [3]), might be more suited, as the features have physical meaning (e.g. electrostatic potential). Moreover, assuming we stuck with handcrafted features, one should investigate which features are most relevant before considering the graph comparison.

Finally, we only experiment with very few architectures and distance measures. A more thorough exploration of the literature might unveil interesting metrics that are more suited for data selection. As a hands-on class project, we remained on a high-level and did not study the dynamics of each kernel in depth. We however believe this is an interesting direction and we did not find a lot of other research in data selection based on graph similarities. We believe that this approach can improve the performance of existing models and uncover new insights.

References

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

[2] Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, Peter W. Battaglia, Vishal Gupta, Ang Li, Zhongwen Xu, Alvaro Sanchez-Gonzalez, Yujia Li, and Petar Velickovic. ETA prediction with graph neural networks in google maps. *CoRR*, abs/2108.11482, 2021.

[3] P. Gainza, F. Sverrisson, F. Monti, E. Rodolà, D. Boscaini, M. M. Bronstein, and B. E. Correia. Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning. *Nature Methods*, 17(2):184–192, December 2019.

[4] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on tabular data?, 2022.

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

[6] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabás Póczos, and Eric P. Xing. Neural architecture search with bayesian optimisation and optimal transport. *CoRR*, abs/1802.07191, 2018.

[7] Rainer Kujala, Christoffer Weckström, Richard K. Darst, Miloš N Mladenović, and Jari Saramäki. A collection of public transport network data sets for 25 cities. *Scientific Data*, 5(1), May 2018.

[8] Hermina Petric Maretic, Mireille El Gheche, Giovanni Chierchia, and Pascal Frossard. GOT: an optimal transport framework for graph comparison. *CoRR*, abs/1906.02085, 2019.

[9] Giannis Nikolentzos, Giannis Siglidis, and Michalis Vazirgiannis. Graph kernels: A survey. *Journal of Artificial Intelligence Research*, 72:943–1027, nov 2021.

[10] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In David van Dyk and Max Welling, editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 488–495, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR.

[11] Alex Tamkin, Mike Wu, and Noah Goodman. Viewmaker networks: Learning views for unsupervised representation learning. 10 2020.

[12] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[13] Peter Wills and François G. Meyer. Metrics for graph comparison: A practitioner's guide. *PLOS ONE*, 15(2):1–54, 02 2020.