Masters Thesis: Constraint-Aware Meta-Learning, with Applications to Traffic Flow Prediction

Vincent Filardi
Data Science
Worcester Polytechnic Institute
vfilardi@wpi.edu

Advisor: **Oren Mangoubi**
Mathematical Sciences
Data Science
Worcester Polytechnic Institute
omangoubi@wpi.edu

Data Science
Worcester Polytechnic Institute
5/6/2022

APPROVED: _____

Abstract

We consider the general problem of training a machine learning model via a meta-learning approach, on a set of tasks $T_1, \ldots, T_n$ where the data $D_1, \ldots, D_n$ in each task satisfies a different set of inequality constraints $C_1, \ldots, C_n$. For each task T_i, we are given a dataset D_i and a set of inequality constraints C_i, which the data is known to satisfy. The goal is to train the meta-learning model in such a way that it has a high prediction accuracy on each task without over-fitting the data.

In the traditional meta-learning framework, task-specific constraint information C_i is not taken into account, and the learner must adapt the model to each task T_i using only the data D_i. In contrast, we propose a "constraint-aware" meta-learning framework, where we empower the meta-learner and adaptive learner to take into account both the data D_i as well as the constraints C_i which the data satisfies for any given task. This approach can potentially allow the trained model to achieve a much higher accuracy without over-fitting on tasks T_i where the dataset D_i is very small (or in the zero-shot learning scenario where there is no task-specific data available for the task T_i), provided the task-specific constraints C_i are known.

We apply our general framework to the problem of predicting the traffic flow of vehicles on different road networks. Changes to a road network present challenges to urban space and its mobility patterns. Here, the task T_i corresponds to the problem of predicting traffic flow on a given road network G_i, and the topology G_i of the road network imposes constraints C_i on the vehicle drivers. Our model learns how drivers respond to the constraints C_i imposed by the topology of the road network, by comparing traffic flow data from different road networks with different topologies G_i. We apply our model to a synthetic dataset generated by a traffic model with drivers that use shortest-path decision rules. When applied to this dataset, we observe that our model achieves a lower prediction error when compared to baseline models which do not take into account the constraints on the given tasks.

1 Introduction

As humans, when we need to learn something new quickly, we do not learn from scratch. If we are lucky, we may have a similar previously solved task and transfer that knowledge to the new task. When we are not as lucky, we consult our vast body of prior experience and knowledge of all prior tasks to solve the new task quickly. This strategy for learning the new task is under the assumption that our past experiences somehow relate to the new task we wish to learn. Still, when presented with many tasks, we want to take all tasks into account and learn quickly or more proficiently than solely on a new task. We can cover this task structure effectively by aggregating the past data across tasks to prime a learning agent to adapt to a new task.

Many popular machine learning frameworks, such as meta-learning [1] and transfer learning [2], aim to solve such multi-task learning problems. One popular meta-learning framework is model-agnostic meta learning (MAML) [3]. In the model-agnostic meta-learning algorithm, information is stored in trainable parameters updated by two agents: the meta-learning agent and adaptive learning agents. Starting from an initial set of trainable parameters θ, the adaptive learning agent chooses a set of task-specific adaptive parameters $\theta + \phi_i$, minimizing a loss function over the task-specific dataset D_i associated with each individual training task T_i. The meta-learning agent chooses the parameters θ to minimize the average loss over the entire collection of tasks T_i after the adaptive learner makes its update $\theta + \phi_i$.

Within the few-shot setting of the meta-learning framework, we sometimes do not have a sufficient amount of data to generalize to a new task. Often different task data satisfies a different set of constraints. If we know which constraint is associated with each task, we can incorporate this information during training to aid in generalizing to new tasks. This is especially useful for supplementing the training data when data is either expensive or unavailable. For instance, if we

wish to predict traffic for some town, cars must travel along a road network. The topology of the road network determines the constraints. Using this additional information about the network topology, we could potentially more quickly adapt new predictions to a new road network with a small amount of data.

In this paper, we consider the general problem of training a machine learning model via a meta-learning approach on a set of tasks $T_1, \ldots, T_n$ where the data $D_1, \ldots, D_n$ in each task satisfies a different set of inequality constraints $C_1, \ldots, C_n$. For each task T_i, we are given a dataset D_i and a set of inequality constraints C_i, which the data is known to satisfy. The goal is to train the meta-learning model in such a way that it has a high prediction accuracy on each task without over-fitting the data.

To the best of our knowledge, in previous meta-learning frameworks, task-specific constraint information C_i is not explicitly considered when training the model, and the learner must adapt the model to each task T_i using only the data D_i. In contrast, we propose a "constraint-aware" meta-learning framework, where we empower the meta-learner and adaptive learner to consider both the data D_i and the constraints C_i which the data satisfies for any given task. This can potentially allow the trained model to achieve a much higher accuracy without over-fitting on tasks T_i where the dataset D_i is very small (or no task-specific data available for the task T_i), provided the task-specific constraints C_i are known.

We apply our general framework to the problem of predicting the traffic flow of vehicles on different road networks. Here, the task T_i corresponds to the problem of predicting traffic flow on a given road network G_i, and the topology G_i of the road network imposes constraints C_i on the vehicle drivers. The constraints C_i arise from the fact that drivers must follow the laws of the road but also the rules resulting from the topology G_i of a road network. Providing C_i as input to a model frees up parameters from having to learn this innate feature from the data, and the model can use the data to learn other valuable features without over-fitting instead. More specifically, the constraints in the traffic flow model are given by a set of flow equations, which require that the net flow on each vertex of the network be equal to zero.

We propose two methods of incorporating C_i as a task descriptor when training our model. In the first method, we provide a low-dimensional representation of C_i as one of the inputs to a machine learning model and the data D_i and train this machine learning model on the collection of tasks T_i using the model-agnostic meta-learning algorithm. One approach we consider to obtain the low-dimensional representation of C_i is to represent C_i by a (much smaller) set of features B_i obtained via feature engineering provided by a domain expert. Another approach we consider is to train an auto-encoder to learn a low-dimensional representation of the different constraint sets C_i. As an alternative method of incorporating the information about the constraint set C_i when training our model, we generate "fake" data which satisfies the constraint information C_i and use this data to augment our model's training data. In both methods, we also require the model's output to satisfy the constraints C_i by projecting the model's prediction onto the constraint set C_i.

We apply our general framework to the problem of predicting the traffic flow of vehicles on different road networks. Changes to a road network present challenges to urban space and its mobility patterns. Here, the task T_i corresponds to the problem of predicting traffic flow on a given road network G_i, and the topology G_i of the road network imposes constraints C_i on the vehicle drivers. When drivers plan a route, the topology of the road network implicitly constrains the space of possible destination paths. Viewing our road network as a graph, we can explicitly use a representation for our constraint set, the incidence matrix of G_i. The flow on the intersections of the roadway must also sum to zero, assuming every car which enters an intersection must also exit. To provide data to train our models, we generate synthetic data using a graph and rule 184 of cellular automata [4].

The graph cellular automata models the road network as a discrete dynamical system where the edges of the road network correspond to streets and vertices correspond to intersections. At each timestep, we update the states of the automata by keeping track of which roads the cars are currently traveling on. Cars in our simulation interact with the cars around them and follow the shortest path to a destination node. We alter the road network based on weather conditions when generating the data. In addition, the destination of each is determined by sampling from a probability distribution parameterized by the date and time.

In our experiments, the network G_i is comprised of two sub-networks connected by a set of edges representing "bridges" B_i. The edges B_i vary depending on the task T_i while the vertices and other edges in the network are kept fixed across the set of tasks T_i. Each task T_i is parametrized by the constraint set C_i determined by the network G_i, with a set of predictor variables specifying the weather, date, and time. In our first proposed method, we make our model constraint-aware by using B_i as input and constraining our solutions to the null space of the incidence matrix. Our experiments for this method successfully converged to higher test accuracy on fewer data points than our baseline models. In addition, this method showed better performance in the zero-shot setting when compared to a baseline trained in a traditional non-zero-shot training scenario. The model overcame negative transfer and overfitting challenges in both experiments by explicitly enriching the model with constraint information. Adding B_i as input strengthened the relationship between the tasks, minimizing the cross-task interference, allowing the model to leverage the task structure and train more effectively. The same can be said for the projection of the model's output which effectively limited the feasible solution space and simplified the model.

To handle situations where the set of edges B_i which vary across tasks is not explicitly provided by a domain expert, we explored instead providing the edge betweenness centrality of the networks G_i as input to our model. The edge betweenness centrality of a network is defined as the number of shortest paths of the graph which go through an edge for each edge of the graph [5]. Unfortunately, the edge betweenness centrality of C_i increased the input dimension of our model considerably. This increase in input size led to an increase in model parameters and impacted the model's generalization ability. The models using the edge betweenness centrality did not perform as well as the models feature engineered with B_i, likely because the high dimension of the input caused our model to overfit the data. We trained an autoencoder to lower the input dimension while still leveraging the edge betweenness centrality of C_i to learn a latent representation of the edge betweenness centrality. This was unsuccessful in learning a meaningful latent representation and requires further work. Another method we tried is to augment the training data set with synthetically generated "fake" data, generated from a distribution parameterized only by the constraint set C_i, to train our model; however, this did not lead to an improvement in the model's performance.

2 Related Work

2.1 Meta Learning and Transfer Learning

Many deep learning approaches rely on large-scale properly-labeled datasets. This sheer amount of data may not be available and may not be practical to clean. Problems with limited data require different frameworks to alleviate data-hungry methods, including few-shot learning. Few-shot learning has shown success in learning quickly from a small number of samples [3,6–9]. Having the perspective of "learning to learn," the adaptive learner learns from specific data belonging to a labeled task, T_i, while an meta learner learns across a collection of tasks. Each task is split up into a support and query set, used by the meta and adaptive learners. With a small number of samples, each task follows an N-way K-shot format, N is the number of classes, and K is the number of

examples. Tasks must share the same structure and have some degree of similarity.

Metric learning approaches allow for improved model training on multi-task datasets by exploiting the similarity between tasks. Metric learning learns an embedding that minimizes intra-class differences while encouraging divergence among different classes. Several contributions in this area have been made in work such as Matching Networks [6], Prototypical Networks [7] and AM3 [10]. Due to optimization challenges, few-shot learning methods may also suffer from negative transfer between tasks.

Another approach to the few-shot problem settings is to use transfer learning. In transfer learning, the number of tasks is not at the scale of meta-learning but still shares the same structure. With a new task T_b, a transfer learning model can leverage the insight from a solved task T_a to fine-tune the new task. Transferable Meta-Learning (TML) leverages sufficiently labeled source domain data and tasks to generalize well to a shifted target domain with insufficient labels [11].

In contrast to image classification, task structure is not apparent in many applications, such as urban traffic prediction. The sequential time series data of urban traffic prediction settings makes it challenging to segment the data into separate tasks, and previous works have addressed this challenging problem. For instance, the recent work Domain Adaptable Continuous Meta-Learning takes a Bayesian meta-learning approach to successfully capture the spatial and temporal correlations of urban traffic data without requiring explicit spatial or temporal task segmentation of the dataset [12].

2.2 MAML

Model-Agnostic Meta-Learning (MAML) is another meta-learning framework that tackles the few-shot framework [3]. MAML creates an optimization scheme to facilitate learning and adapting to new tasks, although this may require computing the second derivative of an objective function when training the model. Recent works have proposed other methods to solve computational bottlenecks arising when training MAML models, such as Reptile [13], a first-order optimization algorithm for training MAML models, and CAVIA, which partitions parameters to facilitate fast context adaption [14]. As one application of MAML, [15] uses MAML for weight initializations combined with feature extraction with GNN, followed by a metric learning module for meta fine-tuning of labeled support and unlabeled query sets [15].

2.3 Cellular Automata and Traffic

Cellular Automata is a collection of cells with a finite number of states for each cell. An individual cell has a neighborhood of cells around it interacting with its states. At the start of a simulation $t = 0$, each cell's initial state is assigned. Any initial state configuration may lead to complex behavior with the addition of randomness. Moving to the next timestep, $t + 1$, cells interact with their neighborhood of cells, and their states change according to a programmed set of fixed rules. These fixed rules are static and applied to the entire grid of cells in parallel. This simple set of programmed rules can result in complex patterns resulting in areas such as physics, theoretical biology, complexity theory, and traffic simulation. One famous example of cellular automata includes Conway's Game of Life [16].

The parallelization and low computational cost make cellular automata excellent candidates for modeling vehicular traffic data. We can think of streets as a linear array of cells whose states depend on each cell's two neighboring cells. Nagel and Schreckenberg popularized the application of cellular automata to traffic in their theoretical model for freeway traffic jams [17]. The Nagel-Schreckenberg freeway traffic model follows five rules. The first rule of acceleration: all cars not

traveling at max velocity increase by one velocity unit until the max is reached. The second rule controls how cars slow down: If a car is within fewer cell units than the velocity units of the car in front, the car slows down the number of velocity units needed to avoid a collision. The model also incorporates stochastic behavior: cars have a small probability of reducing their velocity by 1 unit. The final rule moves all cars forward with the number of cells equal to their velocity. The Nagel-Schreckenberg model has been further expanded and improved to model traffic phenomena such as traffic lights, multiple intersections, and highway travel [18, 19], The Nagel-Schreckenberg model has also inspired work on analyzing best driving practices, city traffic forecasting, and cellular automata on graphs [20–22]. We use a graph cellular automata to model traffic flow between 2 cities connected by a set of bridges.

3 Meta-Learning Frameworks

We consider the general problem of training a machine learning model on a set of tasks $T_1, \ldots, T_n$ where the data $D_1, \ldots, D_n \in \mathcal{D}$ in each task satisfies a different set of inequality constraints $C_1, \ldots, C_n$. For each task T_i, we are given a dataset D_i and a set of inequality constraints C_i, which the data is known to satisfy. The goal is to train the meta-learning model so that it has a high prediction accuracy on each task without over-fitting the data.

Training in the supervised learning framework: In many supervised learning problems, one is given a training dataset $D = \{(x_i, y_i)\}_{i=1}^m$, with predictor variables x_i and labels y_i, sampled from some "population" distribution $\mathcal{D}$ [23]. To train the parameters θ of the model $f(\theta, x)$, one can solve the minimization problem

$$\min_\theta \sum_{j \in D} \ell(f(\theta; x_j), y_j) \tag{1}$$

where the loss function ℓ gives the error of the model's prediction.

Training in the traditional Meta-learning framework: In meta-learning problems, one is instead given a collection of datasets $D_1, \ldots, D_n$ [24]. Each dataset D_i is assumed to be sampled from some population distribution $\mathcal{D}_i$. The distributions $\mathcal{D}_1, \ldots, D_n$ are parameterized by different tasks $T_1, \ldots, T_n$. Here, the tasks T_i represents the setting in which the dataset D_i was collected; for instance, in the application to traffic flow prediction, the tasks T_i may encode the day of the week or year in which the data was collected as well as the road network on which the data was collected. The tasks $T_1, \ldots, T_n$ are assumed to be sampled independently from some distribution $\mathcal{T}$. The goal is to train a machine learning model in such a way as to minimize the prediction error of the model on a test dataset D_τ corresponding to a given "test" task T_τ sampled from the distribution $\mathcal{T}$.

Towards this end, one can consider a multi-agent optimization problem, which is the approach taken, e.g., by the MAML algorithm [3]. Here one considers models $f(\theta, \phi; x)$ with two sets of parameters θ and ϕ chosen by different agents. The first agent–the meta-learning agent–selects a single set of parameters θ, and the model uses this same set of parameters when making predictions on every task T_i. And, for each task T_i, a second agent– the adaptive agent– chooses "task-specific" parameters ϕ_i, which are only used by the model when making predictions on the task T_i. To train

the model, one finds parameters θ which solve the following multi-agent optimization problem:

$$\min_{\theta} \sum_{i=1}^{n} \min_{\phi_i} \sum_{(x,y)\in D_i} \ell(f(\theta, \phi_i; x), y) \tag{2}$$

$$\min_{\phi_i} \sum_{(x,y)\in D_i} \ell(f(\theta, \phi_i; x), y) \tag{3}$$

for some loss function ℓ. Here, we note that the adaptive parameters ϕ_i are chosen separately for each task T_i the model trains on. When applying the model to a new "test" task T_τ, one uses the meta-learning parameters θ obtained by solving (2) on the training Tasks $T_1, \dots T_n$, and finds task-specific parameters T_τ by solving (3).

Compared to the basic supervised learning training framework (1), the meta-learning training framework of Equations (2), (3) can oftentimes lead to a lower prediction error when the data D_i is sampled from a probability distribution $\mathcal{D}_i$ which is different for each task T_i since each set of parameters ϕ_i can be chosen separately to minimize the prediction error on the given task T_i. However, as each task-specific dataset D_i may contain a very small number of data points, which can be a much smaller amount of data than the entire training dataset $\cup_{i=1}^{n} D_i$, one must be careful to avoid over-fitting the task-specific parameters ϕ_i. To avoid over-fitting, some restriction is imposed on the adaptive learner when choosing the adaptive parameters ϕ_i; for instance, ϕ_i may be chosen from a space of dimension much smaller than the dimension of the parameters θ, or there may be a computational restriction on the adaptive learner when it computes the parameters ϕ_i.

The meta-learning framework of Equations (2), (3) includes many existing approaches to meta-learning. In particular, one recovers the model-agnostic meta-learning method of [3] if $f(\theta, \phi_i; x) = g(\theta + \phi_i; x)$ for some function g, and the adaptive learner is computationally restricted to finding the parameters ϕ_i by taking a small number of gradient descent steps.

Constraint-aware meta-learning Framework, with constraints as model inputs:

While placing restrictions on the parameters ϕ_i may be necessary to avoid over-fitting the task-specific dataset D_i, placing too many restrictions on the choice of parameters ϕ_i can lead to a larger training error. In the setting where we know that the labels in the dataset D_i come from a distribution with support on some known constraint set C_i, we can incorporate this additional task-specific information when training the model's parameters. This may, in principle, allow us to train the model with fewer restrictions on the task-specific parameters ϕ_i without over-fitting. However, we still need a way to incorporate the information about the constraints C_i into the model training.

As a first approach, we design the machine learning model f to take as input some representation $g(C_i)$ of the constraints C_i. For instance, in the traffic flow application, the constraints C_i on the traffic flow can be represented by the Laplacian matrix of the road network G_i. And we can project the output of the model onto the constraint set via a projection map P_{C_i}. To train the model, one would therefore like to find parameters θ and ϕ_i, which solve the following multi-agent optimization problem:

$$\min_{\theta} \sum_{i=1}^{n} \min_{\phi_i} \sum_{(x,y)\in D_i} \ell(P_{C_i} f(\theta, \phi_i;\ x, g(C_i)),\ y) \tag{4}$$

$$\min_{\phi_i} \sum_{(x,y)\in D_i} \ell(P_{C_i} f(\theta, \phi_i;\ x, g(C_i)), y) \tag{5}$$

Unfortunately, if the number of constraints is large, the input $g(C_i)$ to the model may have a large dimension. Thus, if, e.g., the model is a neural network with weights θ, ϕ_i, this may cause

the number of trainable parameters in the neural network to be large as well, which can lead to over-fitting. One way around this problem is to choose a low-dimensional representation $g(C_i)$ of the parameters, where the map g is not invertible. For instance, if C_i are the flow constraints for a given network G_i, we can choose $g(C_i)$ to be a function only of the edges connecting the most important vertices in the network G_i. Alternatively, we also consider training an auto-encoder to learn a low-dimensional representation $g(C_i)$ of the constraints C_i of any given task T_i.

In particular, if we remove the parameters ϕ_i from the model f, then one can apply the trained model to new tasks T_τ where no data is available for training, and only the constraints C_τ associated with the task are known. When applied to the traffic flow prediction problem, a model trained with this optimization framework would allow for predictions on a road network where no real data is yet available. This would allow an urban planner to predict the traffic flow patterns of a proposed road network before it is constructed.

Constraint-aware meta-learning via data augmentation:

As an alternative way to incorporate information about the constraints C_i into the model training without introducing additional inputs into the model f, we can use the constraints C_i to generate synthetic data $\hat{D}_i$ specific to the task T_i in order to augment the "real" task-specific dataset D_i. The synthetic data $\hat{D}_i$ should lie inside the constraint set C_i, For instance, in the traffic flow prediction problem, the synthetic dataset $\hat{D}_i$ might be obtained by running a random walk on the road network or by computing the edge betweenness centrality [25] of the road network. We can train this model by solving the following minimization problem

$$\min_{\theta} \sum_{i=1}^{n} \sum_{(x,y)\in D_i \cup \hat{D}_i} \ell(P_{C_i} f(\theta, \phi_i(\theta);\ x),\ y) \tag{6}$$

$$\phi_i(\theta) \in \operatorname{argmin}_z \sum_{(x,y)\in \hat{D}_i} \ell(P_{C_i} f(\theta, z;\ x, y) \tag{7}$$

Here the adaptive learning agent is only allowed to use the synthetic dataset $\hat{D}_i$ when training the task-specific model parameters ϕ_i (Equation (7)). On the other hand, the Meta-learning agent can use both the real data as well as the synthetic data to train the global parameters θ (Equation (6)) and chooses its parameters θ in such a way that the adaptive learner's choice of parameters $\phi_i(\theta)$ causes the model to have a low prediction error on both the real *and* the synthetic data. Thus, once the model parameters θ are found by solving the optimization problem in (6) and (7), the adaptive learning agent can adapt the model to any test task T_τ by solving the optimization problem in (7), using only synthetic data generated using the constraints C_τ associated with the task τ. Thus, the learning framework in (6) and (7) also allows one to adapt the model to new test tasks where no data is yet available.

4 Data Generation and ML Methodologies

This section will detail how we generated the data and the task structure. In all experiments, we consider a road network $G = (E, V)$, where $G_s = (E_s, V_s)$ and $G_w = (E_w, V_w)$ are the road networks of two disconnected graphs of Stamford, CT and Worcester, MA respectively. Next, we normalize the physical latitude and longitude of the vertices of each graph to lie in [0,1]. We enclose both graphs into a bounding circle centered at (.5, .5) and divide the circle into ten equal sectors. Letting $\{v_{s1}, ..., v_{s10}\}$ be 30 vertices of G_s corresponding to 30 partitioned sectors of the circle and similarly $\{v_{w1}, ..., v_{w30}\}$ of G_w to the same partition, we connect each of the disconnected graphs by edges between $\{(v_{w1}, v_{s1}), ..., (v_{w30}, , v_{s30})\}$ to form the graph

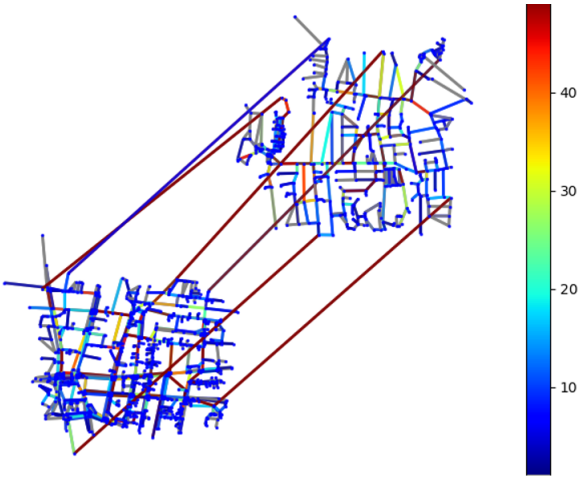

Figure 1: A visualization of a sample road network G_i used to generate an example of our synthetic dataset. We can visually see the "bridges" B_i connecting the two sub-networks on the bottom left and top right. The color of each edge represents the total edge flow on that edge in our simulation.

$G = (G_s, G_w) = (E_s \cup E_w \cup \{(v_{w1}, v_{s1}), ..., (v_{w30}, , v_{s30})\}, V_s \cup V_w)$ (Figure 1). For convenience, we denote these connecting edges as "bridges". In a particular graph G_i we sometimes may not use all of the edges in $\{(v_{w1}, v_{s1}), ..., (v_{w30}, , v_{s30})\}$ to connect G_s and G_w. To denote which bridges are included in a particular graph G_i we use a bit-wise vector $B_i \in \{0, 1\}^{30}$.

4.1 Datasets

There are various ways to capture real-life traffic patterns. Often these datasets do not include real-time GPS data due to privacy concerns. The real-time GPS data can also be highly inaccurate due to noise and poor sampling at small scales. All data in our work was synthetically generated using cellular automata on a graph structure similar to the traffic model of Nagel and Schreckenberg [17] to avoid the pitfalls of real GPS data and provide a controlled testing environment for new algorithms.

Our simulation is a function of a graph G_i with bridge configuation vector B_i and our input variables x. Each simulation is associated with a specific bridge configuration B_i and, therefore, a specific set of inequality constraints, C_i, which the traffic flow y generated by the simulation must satisfy. In addition to the bridge configuration B_i, the input to are simulation includes the variable $x =$ (time, day, month, weather, temperature) where each input variable has the following integer ranges:

- time $\in \{0, \ldots, 23\}$
- day $\in \{0, \ldots, 6\}$
- month $\in \{ 0, \ldots, 11\}$
- weather $\in \{0, , 3\}$
- temperature $\in \{0, \ldots, 4\}$

The output of a simulation, y, represents the total edge flow of automaton vehicles for each edge on the input graph G_i after 4000 time steps. Our simulation function, S, simulates 3000 cars moving along the road network of the input graph. The destination of each driver is sampled at random

from a probability distribution represented by a vector $d \in [0,1]^{|V|}$. The temporal variables, time of day, the month of the year, and day of the week influence the driver destination probability vector d. The temporal input features partition the vertices of G_i. This partition receives an increase in probability in d to capture different temporal demands of a road network. To facilitate cars moving between G_w and G_s, each graph component of G_i receives an equal increase in probabilities. The weather influences the capacity of each edge on the road network; this is to capture the additional space needed between vehicles to drive safely in inclement weather. The temperature variable influences the max speed limit on the edges of G_i.

To start a simulation, all cars are initialized and given destinations randomly according to d. Every car travels to its assigned destination vertex on G following the shortest path. The shortest path for each car is weighed by edge weights proportional to the length of a road and assigned speed limit. Cars may also only travel on roads at less than full capacity. If a car arrives at a full capacity road, it calculates a new shortest path to its destination. In addition, cars have a small probability of giving up on their current destination and being assigned a new destination. To capture the randomness of an actual driver and enforce travel times along edges, once each car reaches a new edge, the car must wait on this edge according to a Poisson process as a function of the travel time along said edge.

4.2 Task Structure

In each simulation, drivers must adhere to the constraints C_i imposed by the topology of the road network G_i. We use this relationship to design our tasks T_i. For every task T_i, we generate an associated dataset D_i. Each dataset for a task is comprised of 45 datapoints $D_i = (((x_1, y_1), ..., (x_{45}, y_{45}))$ with x the input feature and y the vector of flows on the edges of the graph. When training our models, each task T_i is sampled i.i.d. from a distribution $p(\mathcal{T})$.

4.3 Enforcing constraints

Each graph $G = (E, V)$ has a natural matrix representation as its incidence matrix M [26]. G can be represented as $M \in \mathbb{R}^{|E|} \times \mathbb{R}^{|V|}$, with 1 at $(i, j) \in M$ if an edge i is connected to vertex j and 0 otherwise. If we are trying to enforce the constraint of the net flow on each edge to be 0, then we should look for solutions y satisfying $My = 0$. Another way to think of this constraint is that y must be in the null space of M, null(M). We can obtain the null space of M via many matrix decomposition methods on M such as Graham-Schmidt, QR decomposition, and SVD [27]. By projecting the output of our model into the null space of the incidence matrix of G_i for each task T_i, we can enforce the constraint of a net flow of zero on each vertex.

5 Methodologies

This section provides details of the machine learning models we consider in our experiments, including the baseline models provided for comparison, and the "constraint-aware" models we introduce. In all experiments, we use root mean square error (RMSE) to evaluate our edge flow predictions.

5.1 Baseline models

The following sections describe the baseline machine learning models we consider.

5.1.1 Sample Mean Model

The simplest model we consider uses the sample mean of the training data to predict the network flows, denoted as "MEAN." If a model could not beat the "MEAN" model's performance, we do not include the result in Section 6 and accompanying plots. By taking an average of the y's of the training set, the "MEAN" model does not capture the modes of the data associated with the input variables such as time, day, month, weather, and temperature.

5.1.2 Decision Tree

The second baseline model we consider is a basic decision tree model. With relatively few parameters compared to other models we tried, the decision tree was able to pick up on the non-linearity of the data and generalized well. We tuned the decision tree's parameters to have the lowest RMSE on a validation set. A more detailed explanation of the results can be seen in Section 6.

5.1.3 MAML

Another baseline model we considered was a neural network trained using MAML. We use a fully connected neural network trained using MAML for the optimization scheme in our experiments. The tasks for MAML are the same as discussed in Section 4.2. When training the model we used a first-order approximation for MAML, with one way, five shots, and five updates of the adaptive learner. In each experiment, we predict the flow y over each edge of the graph G_i as a function of the predictor variables x. The MAML baseline was less successful than the decision tree. We believe this was due to the model over-fitting to the small amount of task data. The baseline MAML model was was not able to beat the baseline decision tree with an RMSE of 251.22 after ample hyper-parameter tuning. For this reason, we did not include its results in Section 6.

5.2 Constraint-aware Meta-Learning, via Feature Engineered Constraint Representation

To improve the performance of both the baseline decision tree and the baseline MAML model, we designed versions of these models which take as input a representation of the constraints C_i obtained via feature engineering. Specifically, we provided the neural network model with the additional information of the bridges B_i which were included in each network G_i, as well as the values of the predictor variables x. Unfortunately, the decision tree's performance did not improve with the additional information of B_i as input.

We then introduced a constraint-aware version of MAML, which takes into account the constraints of a given task C_i (Algorithm 1). Specifically, the neural network model f in Algorithm 1 takes as input a representation $g(C_i)$ of the constraints C_i (in addition to the predictor variables x from the data). To ensure that the traffic flow predicted by the trained model satisfies the constraints C_i, Algorithm 1 projects the output of the neural network model onto the constraints C_i. To represent the constraints C_i, we first used the indicators B_i for the bridges, that is, we set $g(C_i) = B_i$. In the next section we consider an alternative choice of representation g for the constraint set, which bypasses the need for feature engineering by instead using the network edge betweenness centrality measure to encode each constraint set C_i.

To ensure that the output of our model satisfies the constraints C_i, we projected the output onto the null space of the incidence matrix for the network where all 30 bridges connecting G_s and G_w are included. This method of using all 30 bridges avoided output sizing issues. This projection enforced the net flow on each vertex to sum to zero. This is a reasonable assumption since every

car which moves onto an edge must eventually leave this edge, resulting in a net flow of 0 on each vertex. By adding the constraint information to the input and output of the model, we aimed to test if the model could learn efficiently with lower root mean squared error (RMSE) on a validation set.

We evaluated all models on unseen data from tasks used in training. In addition, we explored a zero-shot setting where we trained on graphs with two bridge configurations and tested on configurations of graphs with three bridges. The performance of MAML and decision tree with additional feature engineering and constrained output can be seen in Figure 2.

Algorithm 1 Constraint-aware MAML via constraint representation

Require: Distribution over tasks $P(\mathcal{T})$
Require: Learning rates α_0, α_1, batch size b, number of adaptive steps k
Require: A function $g : \mathcal{C} \to \mathbb{R}^m$ which gives a representation $g(C_t)$ of each constraint set $C_t \in \mathcal{C}$.
Require: Projection P_{C_t} onto each constraint set $C_t \in \mathcal{C}$
Initialize θ_1, ϕ_t
while not done **do**
  Sample tasks $t = 0, \ldots, T$ from $P(\mathcal{T})$
  for $t = 0, \ldots, T$ **do**
    $\phi_t \leftarrow 0$
    for $j = 0, \ldots, k$ **do**
      Sample a batch D_t from the dataset for task t.
      $\phi_t \leftarrow \phi_t - \frac{\alpha_0}{b} \sum_{(x,y)\in D_t} \nabla \ell(P_{C_t} f(\theta + \phi_t; x, g(C_t)), y)$ *(Update adaptive-learner's parameters.)*
    end for
  end for
  for $t = 0, \ldots, T$ **do**
    Sample a batch D_t from the dataset for task t.
  end for
  $\theta \leftarrow \theta - \frac{\alpha_1}{bT} \sum_{t=0}^{T} \sum_{(x,y)\in D_t} \nabla \ell(P_{C_t} f(\theta + \phi_t; x, g(C_t)), y)$ *(Update meta-learner's parameters.)*
end while

5.3 Constraint-aware meta-learning, via constraint representation

The domain expertise required to engineer features such as B_i is often lacking in many real-world applications. Thus, in many cases, we may need to learn a low-dimensional representation of C_i using only the constraint sets $C_1, \ldots, C_T$. To obtain a representation of the constraints, we first computed the edge betweenness centrality of the network, which gives the number of shortest paths of the network which go through each edge of the network [5]. Using the edge betweenness centrality as our function $g(C_i)$, we modified the decision tree and CA-MAML models to take input x and $g(C_i)$ for each task. We see the results of this model in Table 1 with "CA-MAML BC."

With the CA-MAML model on the "30 bridge" dataset, we saw performance beat that of the "MEAN" model marginally, yet not to the same degree as the increase in performance of CA-MAML with B_i on the "10 bridge" dataset. We believe the change in performance of CA-MAML with this choice of $g(C_i)$ is likely due to the increase in input dimensionality. By including edge betweenness centrality as input, we increased the size of the input of the neural network by 1764, or the number of edges on the undirected graph G_i. These new parameters may have caused the model to overfit the trained data and suffer in performance on the holdout test set. The loss curve

of the experiment can be seen in Figure 3. To find a lower dimensional representation of $g(C_i)$, we attempted to use an autoencoder trained on $g(C_i)$ for all tasks. We did not find success in training the autoencoder; we saw the same performance with a latent space of rank two and a latent space of rank 32. We plan to investigate the autoencoder method further and look towards other ways of learning meaningful features the models can use in predicting traffic flows.

5.4 Constraint-aware meta-learning via data augmentation and multi-agent optimization

To implement the framework of constraint-aware meta-learning via data augmentation we introduce Algorithm 2. Here, we have three agents working in together to solve equations 6 and 7. Line 8 of Algorithm 2 updates the parameters ϕ_t of an adaptive learner using "fake" data $\hat{D}_t$ generated using only the constraints C_t. To capture equation 6 we present 2 meta-learners 1 and 2 in Algorithm 2 in lines 13 and 14 which update parameters θ_1 and θ_2.

Algorithm 2 Constraint-aware meta-learning via data augmentation

Require: Distribution over tasks $P(\mathcal{T})$
Require: Learning rates $\alpha_0, \alpha_1, \alpha_2$, batch size b, number of adaptive steps k
Require: A distribution $h(\cdot; C_t)$ for the fake data, parameterized by the constraint set C_t, and an oracle for sampling from this distribution.
1: Initialize θ, ϕ_t
2: **while** not done **do**
3: Sample tasks $t = 0, \ldots, T$ from $P(\mathcal{T})$
4: **for** $t = 0, \ldots, T$ **do**
5: $\phi_t \leftarrow 0$
6: **for** $j = 0, \ldots, k$ **do**
7: Sample a batch $\hat{D}_t$ of fake data from the distribution $h(\cdot, C_t)$.
8: $\phi_t \leftarrow \phi_t - \frac{\alpha_0}{b} \sum_{(x,y)\in\hat{D}_t} \nabla \ell(f(\theta_1 + \phi_t; x), y)$ *(Update adaptive-learner using fake data for task t.)*
9: **end for**
10: **end for**
11: **for** $t = 0, \ldots, T$ **do**
12: Sample a batch of real data D_t from the dataset for task t.
13: Sample a batch $\hat{D}_t$ of fake data from the distribution $h(\cdot, C_t)$.
14: **end for**
15: $\theta_1 \leftarrow \theta_1 - \frac{\alpha_1}{bT} \sum_{t=0}^{T} \sum_{(x,y)\in\hat{D}_t} \nabla \ell(f(\theta_1 + \phi_t; x), y)$ *(Update meta-learner #1 using fake data for all tasks.)*
16: $\theta_2 \leftarrow \theta_2 - \frac{\alpha_2}{bT} \sum_{t=0}^{T} \sum_{(x,y)\in D_t} \nabla \ell(f(\theta_2 + \phi_t; x), y)$ *(Update meta-learner #2 using real data for all tasks.)*
17: **end while**

In practice, we found that Algorithm 2 performed poorly in comparison to the other models we investigated. We suspect the poor performance may be due to a lack of regularization on the updates to θ_1 resulting in the overfitting on the fake data.

Dataset	Learning Scenario	Model	Test RMSE
10 bridges	Basic Meta-Learning	MEAN	268
		Decision Tree	239.5
		CA-MAML B_i	**226**
	Zero-shot Learning	MEAN	275.2
		Decision Tree B_i	250
		CA-MAML B_i	**246**
30 bridges	Basic Meta-Learning	MEAN	556
		Decision Tree BC	**529.6**
		CA-MAML BC	549

Table 1: The above table gives a summary of the results of our experiments. In our experiments we trained different machine learning models on synthetic multi-task datasets: The baseline sample mean model (MEAN), a baseline decision tree model, our "constraint-aware" decision tree model (which takes a representation of the task-specific constraints as input), and our Constraint-Aware Model Agnostic Meta Learning model (CA-MAML) (which also takes a representation of the task-specific constraints as input). The "Dataset" column denotes the dataset used in each experiment. We considered two different meta-learning scenarios: In the first scenario, "Basic Meta-Learning," datapoints from each task were split between a test and train set. In the second scenario, "Zero-shot learning", the models were trained using a training set consisting only of a subset of the tasks in the dataset, and models were then tested on data from different tasks which were not included in the training dataset. To denote the representation of the constraints used as input to our models, we used B_i denotes the bridge indicator representation, and "BC" to denote the edge betweenness centrality representation. We observed that our CA-MAML model with the B_i representation as input had the lowest RMSE on the "10 bridges" dataset, while our decision tree model with BC representation as input had the lowest RMSE on the "30 bridge" dataset.

5.5 Neural Network Architecture and Hyperparameters

For all of the models above using a neural network, we used a general architecture for our deep learning methods. We use a fully connected neural network in both Sections 5.2 and 5.3. Our model architecture used a shallow, fully connected neural network consisting of 4 layers. The first input layer of the model is a linear layer with variable input depending on the experiment has an output of 128. We then apply a ReLU function to the output of the first layer. The second linear layer with an output of 2670. The output of the second layer is followed again by a ReLU function and a dropout layer with $p = 0.33$. The final layer is a linear layer with a varying output size depending on the experiment. We used SGD for the meta learners and ADAM with default bias parameters for the adaptive step when optimizing the neural networks.

6 Experiments

A summary of the results of our numerical experiments is shown in Table 1. For each of our experiments, we used RMSE to measure performance. The best performing model for each learning scenario can be seen in bold in Table 1. The RMSE of our models at different stages of training are shown in Figures 2 and 3.

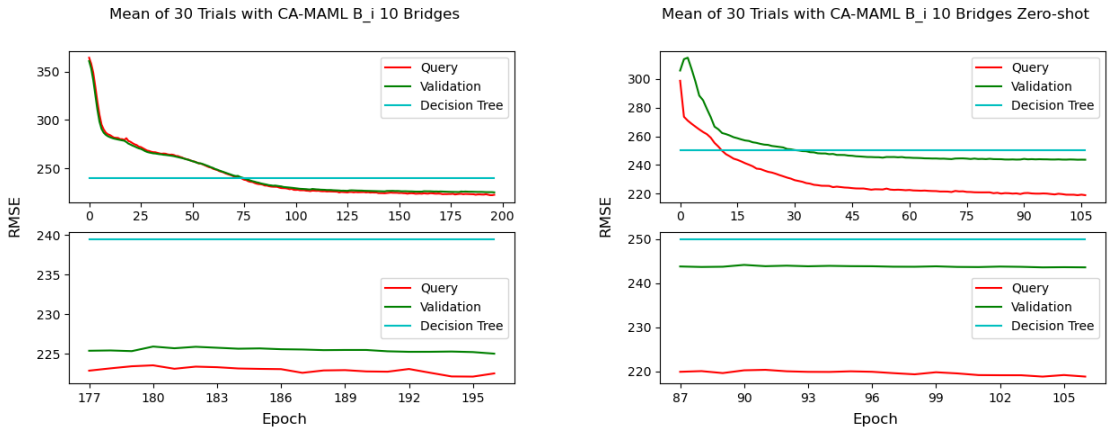

Figure 2: RMSE loss curves for constraint-aware MAML (CA-MAML) on the training "Query" set (red) and test "Validation" set (green). The RMSE for the baseline decision tree model is plotted as a constant line in cyan for comparison. CA-MAML was trained using the"feature engineered" representation B_i of the constraints C_i as model input. The plots on the left were obtained in the "Basic Meta Learning" scenario, while the plots on the right were obtained in the "Zero-shot learning" scenario. All loss curves were averaged over 30 trials on the"10 bridges" dataset. The top plots show the RMSE curves over all epochs of training, while the bottom plots show a close-up of the RMSE over the last 30 epochs.

6.1 Baseline Model

For a baseline model, we use the"MEAN" model of Table 1. The 'MEAN" model corresponds to using the sample mean to predict the testing labels. The RMSE score of each "MEAN" model can be seen in the last column of Table 1.

6.2 Decision Tree

We trained a decision tree as our second baseline model to capture the non-linearities present in our dataset in a small number of parameters. We tuned each decision tree for the best performance in each experiment. Each of the decision tree results of Table 1 are an average of 30 trial runs. We performed a T-test on $n = 30$ on the results of each decision tree experiment. We found statistical significance in each test with a p-value of 1.2×10^{-9}, 1.1×10^{-7}, 5.4×10^{-9} in order of appearance from top to bottom of Table 1. The decision tree with the additional edge betweenness centrality input for each task was the best performing model in the experiments using the "30 bridges" dataset in the "Basic Meta-Learning" scenario.

6.3 Constraint-Aware Model Agnostic Meta Learning via constraint representation

The results of Algorithm 1 can be seen in the Results Table 1 with models titled "CA-MAML". We performed a T-test on $n = 30$ on the results of each CA-MAML experiment noting statistical significance in each with a p-value of 1.6×10^{-11} and 0.0061 for the "Basic Meta Learning" and "zero-shot" learning scenarios, respectively. The loss curve for CA-MAML on the "10 bridges" dataset and "Basic Meta-Learning" scenario can be seen on the left of Figure 2. Similarly, The loss curve for CA-MAML on the "10 bridges" dataset and "zero-shot" scenario can be seen on the right of Figure 2.

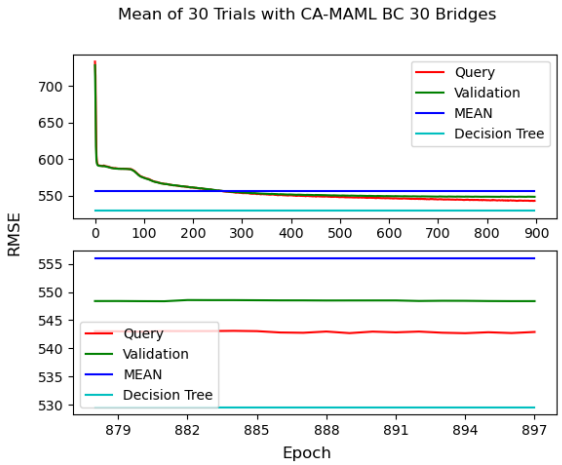

Figure 3: Above we present the loss curves of constraint-aware MAML with the edge betweenness centrality as our constraint representation, on the"30 bridge" dataset in the "Basic Meta Learning" scenario. RMSE for all epochs of training are shown in the top plot, the bottom plot provides a closer look at the last 30 epochs. The RMSE on the query set can be seen in red, and the RMSE on the holdout validation set is green. For comparison, we also plot the RMSE for the decision tree (cyan) and sample mean model (blue) on the validation set. All loss curves are averaged over 30 trials.

We observed that our CA-MAML model with the B_i representation as input had the lowest RMSE, while our decision tree model with BC representation as input had the lowest RMSE on the "30 bridge" dataset. We believe that that the decision tree model performend better under the much higher-dimensional BC representation since it was less likely to over-fit the data due to having fewer trainable parameters than the neural network in the CA-MAML model. On the other hand, we believe that the CA-MAML model had the best performance when the B_i representation was used, since the B_i representation had very low dimension (dimension = 10), preventing the model from over-fitting.

6.4 Constraint-Aware Meta-Learning via Data Augmentation

We observed in our experiments that the Constraint-aware meta-learning via data augmentation model produced an RMSE higher than even the sample mean baseline model. We therefore omit the experimental results for this method.

7 Conclusion

This work introduces various methods to incorporate information about the constraints associated with different tasks in a meta-learning framework. In our experiments, we observed that some of these methods were successful in the basic meta-learning and zero-shot learning setting when provided with a low-dimensional representation of the constraints associated with each task. In future work, we hope to extend our constraint-aware machine learning models to scenarios where domain expertise is not available, by using methods such as autoencoder neural networks to provide a low-dimensional representation of the constraint set as input to our model.

Acknowledgment

Vincent Filardi was supported in part by NSF grant CCF-2104528. He would like to thank Professor Oren Mangoubi, Professor Yanhua Li, and Xin Zhang for their guidance and stimulating conversations on this work.

References

[1] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *arXiv preprint arXiv:2004.05439*, 2020.

[2] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning. arxiv e-prints," *arXiv preprint arXiv:1911.02685*, 2019.

[3] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*, pp. 1126–1135, PMLR, 2017.

[4] K. Nagel, "Particle hopping models and traffic flow theory," *Physical review E*, vol. 53, no. 5, p. 4655, 1996.

[5] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[6] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, *et al.*, "Matching networks for one shot learning," *Advances in neural information processing systems*, vol. 29, 2016.

[7] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," *Advances in neural information processing systems*, vol. 30, 2017.

[8] Y. Guo, N. C. Codella, L. Karlinsky, J. V. Codella, J. R. Smith, K. Saenko, T. Rosing, and R. Feris, "A broader study of cross-domain few-shot learning," in *European Conference on Computer Vision*, pp. 124–141, Springer, 2020.

[9] M. Yin, G. Tucker, M. Zhou, S. Levine, and C. Finn, "Meta-learning without memorization," *arXiv preprint arXiv:1912.03820*, 2019.

[10] C. Xing, N. Rostamzadeh, B. Oreshkin, and P. O. O Pinheiro, "Adaptive cross-modal few-shot learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[11] B. Kang and J. Feng, "Transferable meta learning across domains.," in *UAI*, pp. 177–187, 2018.

[12] X. Zhang, Y. Li, X. Zhou, O. Mangoubi, Z. Zhang, V. Filardi, and J. Luo, "Dac-ml: Domain adaptable continuous meta-learning for urban dynamics prediction," in *2021 IEEE International Conference on Data Mining (ICDM)*, pp. 906–915, IEEE, 2021.

[13] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," *arXiv preprint arXiv:1803.02999*, 2018.

[14] L. Zintgraf, K. Shiarli, V. Kurin, K. Hofmann, and S. Whiteson, "Fast context adaptation via meta-learning," in *International Conference on Machine Learning*, pp. 7693–7702, PMLR, 2019.

[15] P. Rajpurkar, J. Irvin, K. Zhu, B. Yang, H. Mehta, T. Duan, D. Ding, A. Bagul, C. Langlotz, K. Shpanskaya, *et al.*, "Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning," *arXiv preprint arXiv:1711.05225*, 2017.

[16] M. Gardner, "Mathematical games," *Scientific american*, vol. 222, no. 6, pp. 132–140, 1970.

[17] K. Nagel and M. Schreckenberg, "A cellular automaton model for freeway traffic," *Journal de physique I*, vol. 2, no. 12, pp. 2221–2229, 1992.

[18] K. Małecki and S. Iwan, "Modeling traffic flow on two-lane roads with traffic lights and countdown timer," *Transportation Research Procedia*, vol. 39, pp. 300–308, 2019.

[19] W. Ning and W. Brilon, "Cellular automata for highway traffic flow simulation,"

[20] G. Abramson, V. Semeshenko, and J. R. Iglesias, "Cooperation and defection at the crossroads," *Plos one*, vol. 8, no. 4, p. e61876, 2013.

[21] K. Małecki, "Graph cellular automata with relation-based neighbourhoods of cells for complex systems modelling: A case of traffic simulation," *Symmetry*, vol. 9, no. 12, p. 322, 2017.

[22] Y. Hu, M. Li, H. Liu, X. Guo, X. Wang, and T. Li, "City traffic forecasting using taxi gps data: A coarse-grained cellular automata model," *arXiv preprint arXiv:1612.02540*, 2016.

[23] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*, vol. 112. Springer, 2013.

[24] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," *Artificial intelligence review*, vol. 18, no. 2, pp. 77–95, 2002.

[25] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, pp. 35–41, 1977.

[26] D. Spielman, "Spectral graph theory," *Combinatorial scientific computing*, vol. 18, 2012.

[27] L. N. Trefethen and D. Bau III, *Numerical linear algebra*, vol. 50. Siam, 1997.