Internship Report

Travel Time Estimation using Approximate Belief States on a Hidden Markov Model

California Center for Innovative Transportation

Walid Krichene
Contents

1 Context of the internship 3
  1.1 The Mobile Millennium Project . . . . . . . . . . . . . . . . . . . . . . . . 3
  1.2 Objectives . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2 Bayesian inference on a Hidden Markov Model 5
  2.1 Graphical Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
  2.2 Hidden Markov Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
  2.3 Parametrization of a HMM . . . . . . . . . . . . . . . . . . . . . . . . . . 6
  2.4 Inference . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
  2.5 Message passing algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . 8
  2.6 Parameter Estimation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
  2.7 The Expectation Maximization Algorithm . . . . . . . . . . . . . . . . . . 10
    2.7.1 Maximum Likelihood estimate in the general setting . . . . . . . . . 11
    2.7.2 Latent variables . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
    2.7.3 Maximizing a lower bound of the log likelihood . . . . . . . . . . . 12
  2.8 Derivation of the EM algorithm in the case of a HMM . . . . . . . . . . . . 13

3 Modeling Framework 17
  3.1 Arterial network . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
  3.2 HMM . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
    3.2.1 Transition model . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
    3.2.2 Observation model . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
  3.3 Belief state . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
  3.4 Travel Time estimation . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
    3.4.1 Bayesian tracking (E step) . . . . . . . . . . . . . . . . . . . . . . . 20
    3.4.2 Updating the transition and observation models (M step) . . . . . . 20
  3.5 Prediction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
  3.6 Computational intractability and approximate belief states . . . . . . . . . 21

4 Boyen-Koller approximate inference 23
  4.1 Approximate belief state . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
  4.2 Transition model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
  4.3 Inference under the BK approximation . . . . . . . . . . . . . . . . . . . . 23
  4.4 Time complexity of the approximate inference . . . . . . . . . . . . . . . . 24
  4.5 Error induced by the approximate belief state . . . . . . . . . . . . . . . . 24
    4.5.1 Bounding the error in the case of a single process . . . . . . . . . . 24
    4.5.2 Extension to compound processes . . . . . . . . . . . . . . . . . . . 25

5 Partitioning the network into weakly dependent clusters 26
  5.1 Weighted graph of traffic interactions . . . . . . . . . . . . . . . . . . . . 26
  5.2 Partitioning the weighted graph . . . . . . . . . . . . . . . . . . . . . . . 26
  5.3 Partitioning results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28

6 Discussion and summary 30
1 Context of the internship

During my internship at CCIT (California Center for Innovative Transportation), I was part of the Arterial Team, who works, within the Mobile Millennium project, on estimating travel times and traffic indicators on the San Francisco Bay Area Arterial network, using different sources of data, including GPS coordinates of probe vehicles provided by a San Francisco cab network.

1.1 The Mobile Millennium Project

The Mobile Millennium project is about developing a phone application that collects anonymous GPS data from the users and provides them with real time traffic information, and estimates of travel times on the Highways and the Arterial road network. Both the Highway and Arterial teams work on similar problematics, using different approaches due to the inherently different setting: the Highway setting is suited for using a fine grained modeling, using fluid dynamics for instance. Such an approach would be computationally intractable in the setting of the Arterial network due to the large size of the network and to its more complex architecture (presence of intersections and traffic lights). The Arterial team rather uses Statistical Learning techniques to estimate travel times. A traffic model is used to describe the dependency of travel times on a set of traffic parameters, then those parameters are estimated using Bayesian inference on the observed data.

![GPS coordinates](image)

Figure 1: GPS coordinates sent by the probe vehicles during a full day, on the San Francisco arterial network.

The observations are provided by a group of probe vehicles that send their GPS locations periodically, which gives us access to a set of paths on the network, and the travel times along those paths (where a path is a sequence of connected links of the network).Inferring the paths from the raw observations (GPS coordinates) is not a
straightforward task due to the fact the raw GPS coordinates are noisy. The Arterial team has developed an algorithm that infers the paths from raw GPS coordinates. Thus we will assume that the input data of our system are a set of paths across the network, and the travel times along those paths.

1.2 Objectives

The main objective of my internship was to improve an existing traffic model and its corresponding inference algorithm, by improving its computational complexity or the quality of the resulting estimation.

The existing model uses a HMM (Hidden Markov Model) to describe the system by choosing a set of state variables and defining conditional dependencies between those variables. The HMM setting provides a powerful machinery to do Bayesian inference, for example to estimate the parameters of the model using the observed data. However, due to the size of the Arterial road network, exact inference is not computationally tractable. To address this problem, the model was simplified by assuming conditional independence between all links in the network. Thus inference was made independently on every link of the network. This assumption solves the problem of computational complexity, but over-simplifies the model, such that the structure of the network is neglected. Thus we needed to find the right trade-off between exact inference on an over-simplified model, and approximate inference on a complex model.

After some time spent on documentation, we decided to use the Boyen-Koller algorithm, which provides a theoretical method for approximate inference on large HMMs. My work consisted in adapting the Boyen-Koller algorithm to the traffic model at hand, and present a theoretical study of the computational complexity and quality of estimation.

We will first describe the general setting of Hidden Markov Models and summarize the important inference algorithms, such as the message passing algorithms and the Expectation Maximization algorithm. Then we will describe the modeling framework of our system and the exact inference algorithm, as well as the computational problems that arise and render exact inference impractical. Finally we will describe the approximate inference algorithm (adapted from the Boyen-Koller algorithm) and present some theoretical results. As we will explain in the corresponding section, using the Boyen-Koller approximation requires to define a partition of the network that satisfies a set of conditions. We will address the partitioning problem and present results of the proposed partitioning algorithm.
2 Bayesian inference on a Hidden Markov Model

In this section, we will describe the general setting of Hidden Markov Models and summarize the important inference algorithms. This section is largely inspired from the excellent course notes of Pr. Jordan on Statistical Machine Learning.

A Hidden Markov Model is a Graphical Model which dynamics are assumed to be a Markov process, and that has a set of hidden, or unobserved variables. Let us first describe Graphical Models in the general setting.

2.1 Graphical Model

A Graphical Model is a family of probability distributions defined in terms of a graph. The nodes in the graph are identified with random variables, and joint probability distributions are defined by taking products over functions defined on connected subsets of nodes. This formalism provides general algorithms for computing marginal and conditional probabilities. One fundamental observation we should make is that when conditioning on a set of nodes \( \{n^1, \ldots, n^K\} \) separates two variables \( x \) and \( y \), meaning every path that connects \( x \) to \( y \) in the graph contains one of the nodes \( n^k \), then \( x \) and \( y \) are conditionally independent given \( \{n^1, \ldots, n^K\} \).

Figure 2: A simple graphical model. Variables \( x \) and \( y \) are independent conditionally on \( (n_1, n_2) \), but are not independent conditionally on \( n_1 \).

2.2 Hidden Markov Model

We represent the state at time \( t \) as a random variable \( s_t \in (s^1_t, \ldots, s^N_t) \). We assume that state variables are not observed. They are called hidden or latent variables. Instead, we observe variables \( y_t \) that we assume only depend on the state variable \( s_t \).

**Conditional independence**  Let us first observe that conditioning on a state \( s_t \) yields the following conditional independence: variables \( s_u \) and \( s_v \) are conditionally independent for all \( u < t < v \). This means that the past and future are independent given the present. The same result goes for observed responses, \( y_u \) and \( y_v \) are independent, conditionally on \( s_t \), for all \( u < t < v \).
Figure 3: The representation of a HMM as a graphical model. Each vertical slice represents a time step. In each time slice, the top node represents the observed variable $y_t$, and the bottom node represents the hidden (unobserved) state variable $s_t$. Conditioning on state variable $s_t$ separates variables $y_{t-1}$ and $y_{t+1}$.

On the other hand, conditioning on an output node does not separate nodes in the graph and does not yield any conditional independence. In particular, conditioning on all of the output nodes fails to separate any of the hidden state nodes. That is, given the observable data, we cannot expect any independencies to be induced between the state nodes. Thus we should expect that the inference algorithm must take into account possible dependencies between states at arbitrary locations along the chain. In particular, learning something about the final node in the chain $s_T$, by observing $y_T$, can change the posterior probability distribution for all nodes in the chain, $(s_t)_{0 \leq t \leq T}$. Therefore, we expect the inference algorithms to propagate information from one end of the chain to the other.

2.3 Parametrization of a HMM

We parametrize the HMM by assigning local conditional probabilities to each of the nodes. The first state node $s_0$ in the sequence has no parents, thus we assign it with an unconditional distribution $\pi$, where $\pi^i = P(s^i_0)$ and $s^i_0$ is a shorthand notation for the event $s_0 = s^i$. Each successive state node has the previous state node in the chain as its only parent, thus we need a $N \times N$ matrix to specify its local conditional probability. We define a state transition matrix $T$, defined as $T_{i,j} = P(s^j_{t+1} | s^i_t)$. Here we assume that the transition probability is independent of $t$, that is, we assume a heterogeneous HMM.

Each of the output nodes has a single state node as a parent, thus we require a probability distribution $P(y_t | s_t)$. We again assume this distribution to be independent of $t$. We make no further assumptions regarding the form of $P(y_t | s_t)$.

The joint probability is obtained by taking the product over the local conditional probabilities. Thus, for a particular configuration $(s, y) = (s_0, \ldots, s_T, y_0, \ldots, y_T)$, we
obtain the following joint probability

\[ P(s, y) = P(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t) \prod_{t=0}^{T} P(y_t|s_t) \]  

(2.1)

Let \( T_{s_t,s_{t+1}} \) denote the entry \( T_{i,j} \) of the transition matrix when \( s_t = s^i \) and \( s_{t+1} = s^j \). Similarly, let \( \pi_{s_t} \) denote the entry \( \pi_i \) of the initial state distribution when \( s_0 = s^i \). Using these notations, we can rewrite the joint probability formula (2.1) using the transition matrix and the initial state distribution

\[ P(s, y) = \pi_{s_0} \prod_{t=0}^{T-1} T_{s_{t+1},s_t} \prod_{t=0}^{T} P(y_t|s_t) \]  

(2.2)

This is the parametrized probability distribution in which we wish to do inference.

2.4 Inference

There are several inference problems that are of interest in the HMM setting. The general inference problem is to compute the probability of a hidden state sequence \( s \) given an observed output sequence \( y \). It is also interesting to compute the probability of a state \( s \) conditioned on a partial output sequence \( y_{0:t} \).

- The **filtering problem** is the online problem in which we have a sequence of outputs \( y_{0:t} \) and we would like to compute the probability of the state \( P(s_t|y_{0:t}) \) at time \( t \) without waiting for future data.

- The **prediction problem** is to compute the probability of a future state \( P(s_u|y_{0:t}) \) for \( u > t \), given the partial output sequence up to time \( t \).

- The **smoothing problem** is the problem of calculating the probability of a past state \( P(s_v|y_{0:t}) \) for \( v < t \) given the partial output sequence up to time \( t \).

Let us consider the problem of computing the posterior probability \( P(s|y) \) where \( y = y_{0:T} \) is the entire output sequence at our disposal. Let \( s \) be an arbitrary fixed state whose probability we wish to compute. We have \( P(s|y) = P(s, y)/P(y) \). The numerator \( P(s, y) \) is computed using the joint probability formula (2.2), but calculating the denominator \( P(y) \) involves taking a sum across all possible values of the hidden states

\[ P(y) = \sum_{s_0,\ldots,s_T} \pi_{s_0} \prod_{t=0}^{T-1} T_{s_{t+1},s_t} \prod_{t=0}^{T} P(y_t|s_t) \]  

(2.3)

Computing this sum over all possible states \( (s_0, \ldots, s_T) \) is wildly intractable for reasonable values of \( N \) and \( T \) (the number of terms being \( N^T \)). In order to do efficient inference, we need to use the factorized form (2.2) of the joint probability distribution. Each factor involves only one or two of the state variables, and the factors form a neatly organized chain. This suggests that we could switch the sums over states with the products in a systematic way.

7
2.5 Message passing algorithms

To reveal the recursion behind the HMM inference problem, let us focus on a particular state node $s_t$, and compute the posterior probability $P(s_t|y)$. In order to simplify computation, we take advantage of the conditional independencies of the graphical model by conditioning on $s_t$

$$P(s_t|y) = \frac{P(y|s_t)P(s_t)}{P(y)}$$

$$= \frac{P(y_0, \ldots, y_t|s_t)P(y_{t+1}, \ldots, y_T|s_t)P(s_t)}{P(y)}$$

$$= \frac{P(y_0, \ldots, y_t, s_t)P(y_{t+1}, \ldots, y_T|s_t)}{P(y)}$$

$$= \frac{\alpha(s_t)\beta(s_t)}{P(y)} \quad (2.4)$$

where

$$\alpha(s_t) \triangleq P(y_0, \ldots, y_t, s_t) \quad (2.5)$$

is the probability of emitting a partial sequence of outputs $(y_0, \ldots, y_t)$ and ending up in a state $s_t$, and

$$\beta(s_t) \triangleq P(y_{t+1}, \ldots, y_T|s_t) \quad (2.6)$$

is the probability of emitting a partial sequence of outputs $(y_{t+1}, \ldots, y_T)$ given that the system starts in state $s_t$.

Notice that

$$\sum_{s_t} \frac{\alpha(s_t)\beta(s_t)}{P(y)} = \sum_{s_t} P(s_t|y)$$

$$= 1$$

thus

$$P(y) = \sum_{s_t} \alpha(s_t)\beta(s_t) \quad (2.7)$$

That is, we can obtain the likelihood $P(y)$ by calculating $\alpha(s_t)$ and $\beta(s_t)$ and summing their product. We have now reduced the problem to computing the $\alpha$ and $\beta$ variables.
The $\alpha$ forward recursion  Let us first derive a recursive algorithm for computing the $\alpha$ variables. We make use of the conditional independencies of the graphical model

\[
\alpha(s_{t+1}) = P(y_0, \ldots, y_{t+1}, s_{t+1}) \\
= P(y_0, \ldots, y_{t+1}|s_{t+1})P(s_{t+1}) \\
= P(y_0, \ldots, y_t|s_{t+1})P(y_{t+1}|s_{t+1})P(s_{t+1}) \\
= P(y_0, \ldots, y_t, s_{t+1})P(y_{t+1}|s_{t+1})
\]

Marginalizing over $s_t$

\[
= \sum_{s_t} P(y_0, \ldots, y_t, s_t, s_{t+1})P(y_{t+1}|s_{t+1}) \\
= \sum_{s_t} P(y_0, \ldots, y_t, s_{t+1}|s_t)P(s_t)P(y_{t+1}|s_{t+1}) \\
= \sum_{s_t} P(y_0, \ldots, y_t|s_t)P(s_{t+1}|s_t)P(s_t)P(y_{t+1}|s_{t+1}) \\
= \sum_{s_t} P(y_0, \ldots, y_t, s_t)P(s_{t+1}|s_t)P(y_{t+1}|s_{t+1}) \\
= \sum_{s_t} \alpha(s_t)T_{s_t,s_{t+1}}P(y_{t+1}|s_{t+1}) \tag{2.8}
\]

The $\alpha$ variables can thus be computed recursively starting with $\alpha(s_0)$

\[
\alpha(s_0) = P(y_0, s_0) \\
= P(y_0|s_0)P(s_0) \\
= P(y_0|s_0)\pi_{s_0}
\]

The computational complexity of each step of the $\alpha$ recursion is $O(N^2)$: for each one of the $N$ possible values of $s_{t+1}$, we require $N$ multiplications to compute $\alpha(s_{t+1})$. Computing all of the $\alpha$ variables from $t = 0$ to $t = T$ thus requires $O(N^2T)$ time.

The $\beta$ backward recursion  For the $\beta$ variables, we obtain a backward recursion in which $\beta(s_t)$ is expressed in terms of $\beta(s_{t+1})$. The derivation makes use of the conditional independencies that result from the graphical model

\[
\beta(s_t) = P(y_{t+1}, \ldots, y_T|s_t) \\
= \sum_{s_{t+1}} P(y_{t+1}, \ldots, y_T, s_{t+1}|s_t) \\
= \sum_{s_{t+1}} P(y_{t+1}, \ldots, y_{t+1}|s_{t+1}, s_t)P(s_{t+1}|s_t) \\
= \sum_{s_{t+1}} P(y_{t+2}, \ldots, y_T|s_{t+1}, s_t)P(y_{t+1}|s_{t+1})P(s_{t+1}|s_t) \\
= \sum_{s_{t+1}} \beta(s_{t+1})P(y_{t+1}|s_{t+1})T_{s_t,s_{t+1}} \tag{2.9}
\]
In order to initialize the $\beta$ backward recursion, we need to first define $\beta(s_T)$. Notice that the definition of $\beta(s_t) \triangleq P(y_{t+1}, \ldots, y_T|s_t)$ is only valid for $t < T$. Thus, we need to define $\beta(s_T)$ in a way that is consistent with the recursion. Let $\beta(s_T) = 1 \forall s_T$. This definition is consistent with the recursion (2.9) since it yields the correct value of $\beta(s_{T-1})$:

$$
\beta(s_{T-1}) = \sum_{s_T} \beta(s_T) P(y_T|s_T) T_{s_{T-1}, s_T}
$$

$$
= \sum_{s_T} P(y_T|s_T) T_{s_{T-1}, s_T}
$$

$$
= P(y_T|s_{T-1})
$$

2.6 Parameter Estimation

The parameters of the HMM are the transition matrix $T$, the initial probability distribution $\pi$, and the parameters $\eta$ of the Observation model (the Observation model describes the probability distribution of $y_t$ given $s_t$). We wish to estimate the value of these parameters that yields the maximum log likelihood of the observed data.

Let $\theta \triangleq (\pi, T, \eta)$ represent all of the parameters of the HMM model, where the Observation model can be described by the distribution $P(y_t|s_t, \eta)$. The likelihood of an observed output sequence is given by $P(y|\theta)$. Taking the logarithm of (2.2) we have the following log likelihood

$$
P(y|\theta) = \ln \sum_{s_0} \sum_{s_1} \cdots \sum_{s_T} \prod_{t=0}^{T-1} \pi_{s_0} T_{s_t, s_{t+1}} \prod_{t=0}^{T} P(y_t|s_t, \eta)
$$

Our goal is to maximize this expression with respect to $\theta$. The EM algorithm is particularly suited to solving this optimization problem.

2.7 The Expectation Maximization Algorithm

The EM algorithm is an iterative algorithm to compute the maximum likelihood (ML) estimate of a set of model parameters $\theta$, i.e. the value of the parameters for which the observed data are the most likely. It provides a general approach to the problem of ML parameter estimation in statistical models with latent variables.

Each iteration of the EM algorithm consists of two subroutines: The E (Expectation) step, and the M (Maximization) step. In the E step, the hidden (unobserved, latent) variables are estimated given the observed variables and the current estimate of the model parameters. This is achieved using conditional expectation. In the M step, the likelihood function is maximized under the assumption that the latent variables are known (the hidden variables are “filled-in” with their expected value calculated in the E step). Each iteration of the algorithm is guaranteed to increase the likelihood function. In order to derive the EM algorithm, we first need to recall Jensen’s Inequality for convex functions.
Jensen’s Inequality Let $f$ be a convex function and $\lambda_1, \ldots, \lambda_n \geq 0$ that satisfy $\sum_{i=1}^{n} \lambda_i = 1$. Then $\forall x_1, \ldots, x_n$

$$f \left( \sum_{i=1}^{n} \lambda_i x_i \right) \leq \sum_{i=1}^{n} \lambda_i f(x_i)$$

This result can be proved by induction

$$f \left( \sum_{i=1}^{n+1} \lambda_i x_i \right) = f \left( \sum_{i=1}^{n} (\lambda_i x_i) + \lambda_{n+1} x_{n+1} \right)$$

$$= f \left( (1 - \lambda_{n+1}) \sum_{i=1}^{n} \left( \frac{\lambda_i}{1 - \lambda_{n+1}} x_i \right) + \lambda_{n+1} x_{n+1} \right)$$

using the convexity of $f$

$$\leq (1 - \lambda_{n+1}) f \left( \sum_{i=1}^{n} \left( \frac{\lambda_i}{1 - \lambda_{n+1}} x_i \right) \right) + \lambda_{n+1} f(x_{n+1})$$

using the induction hypothesis

$$\leq (1 - \lambda_{n+1}) \sum_{i=1}^{n} \left( \frac{\lambda_i}{1 - \lambda_{n+1}} f(x_i) \right) + \lambda_{n+1} f(x_{n+1})$$

$$= \sum_{i=1}^{n+1} (\lambda_i f(x_i))$$

2.7.1 Maximum Likelihood estimate in the general setting

Let $Y = (Y^1, \ldots, Y^N)$ be a random vector of observed variables with realization $y = (y^1, \ldots, y^N)$, that results from a parametrized probability model $P(Y|\theta)$. Given a realization $y$, we wish to find the maximum likelihood estimate for $\theta$, i.e. the value of $\theta$ that maximizes $P(Y = y|\theta)$. The ML estimate of $\theta$ also maximizes the log likelihood function defined as

$$L(\theta) = \ln P(Y = y|\theta) \quad (2.10)$$

The EM algorithm is an iterative procedure for maximizing $L(\theta)$. Let $\theta_n$ be the estimate of $\theta$ after the $n^{th}$ iteration. At iteration $n + 1$, we wish to update the estimate of $\theta$ such that we maximize

$$L(\theta) - L(\theta_n) = \ln P(Y|\theta) - \ln P(Y|\theta_n) \quad (2.11)$$

2.7.2 Latent variables

Latent or hidden variables are generally introduced into the model in order to simplify it in some way. We may observe a complex pattern of dependency among variables $(Y^1, \ldots, Y^N)$. Rather than modeling this dependency directly via edges linking these variables, we may find it simpler to account for their dependency via top-down dependency on a latent variable $S$. In the simplest case, we may assume that the $Y^i$ are conditionally independent given $S$, and thus restrict our model to edges between the
node $S$ and the nodes $Y^i$.

Let $S$ be the random vector of latent variables and $s$ a given realization of $S$. The probability model becomes $P(Y, S|\theta)$. The probability that we wish to maximize, $P(Y|\theta)$, can be written as a marginal probability

$$L(\theta) = P(Y|\theta) = \sum_s P(Y|s, \theta) P(s|\theta)$$

### 2.7.3 Maximizing a lower bound of the log likelihood

We can then rewrite (2.11) by marginalizing over the latent variables $S$

$$L(\theta) - L(\theta_n) = \ln \left( \sum_s P(Y|s, \theta) P(s|\theta) \right) - \ln P(Y|\theta_n)$$

$$= \ln \left( \sum_s P(s|Y, \theta_n) \frac{P(Y|s, \theta) P(s|\theta)}{P(s|Y, \theta_n)} \right) - \ln P(Y|\theta_n)$$

using Jensen’s Inequality

$$\geq \sum_s P(s|Y, \theta_n) \ln \left( \frac{P(Y|s, \theta) P(s|\theta)}{P(s|Y, \theta_n)} \right) - \ln P(Y|\theta_n)$$

$$= \sum_s P(s|Y, \theta_n) \ln \left( \frac{P(Y|s, \theta) P(s|\theta)}{P(s|Y, \theta_n) P(Y|\theta_n)} \right)$$

$$\Delta = \Delta(\theta, \theta_n)$$

which can be written

$$L(\theta) \geq \Delta(\theta, \theta_n) + L(\theta_n) \Delta \equiv l(\theta, \theta_n)$$

(2.12)

with equality if $\theta = \theta_n$ since $l(\theta_n, \theta_n) = \Delta(\theta_n, \theta_n) + L(\theta_n) = L(\theta_n)$. Our objective is to maximize the log likelihood function $L(\theta)$ which is bounded by $l(\theta, \theta_n)$. At iteration $n + 1$, we update the estimate of $\theta$ by maximizing $l(\theta, \theta_n)$ instead of $L(\theta)$. Doing so ensures that $L(\theta_{n+1}) > L(\theta_n)$, since $L(\theta_{n+1}) \geq l(\theta_{n+1}, \theta_n) > l(\theta_n, \theta_n) = L(\theta_n)$. We can now derive the expression of $\theta_{n+1}$

$$\theta_{n+1} = \arg \max_\theta \left\{ l(\theta, \theta_n) \right\}$$

$$= \arg \max_\theta \left\{ L(\theta_n) + \Delta(\theta, \theta_n) \right\}$$

$$= \arg \max_\theta \left\{ \sum_s P(s|Y, \theta_n) \ln \left( \frac{P(Y|s, \theta) P(s|\theta)}{P(s|Y, \theta_n) P(Y|\theta_n)} \right) \right\}$$

12
Dropping the constant terms with respect to \( \theta \) we obtain

\[
\theta_{n+1} = \arg \max_{\theta} \left\{ \sum_s P(s|Y, \theta_n) \ln \left( \frac{P(Y|s, \theta) P(s|\theta)}{P(s|\theta)} \right) \right\}
\]

\[
= \arg \max_{\theta} \left\{ \sum_s P(s|Y, \theta_n) \ln (P(Y|s)) \right\}
\]

\[
= \arg \max_{\theta} \left\{ E_{S|Y, \theta_n} (\ln (P(Y,s|\theta))) \right\}
\]

(2.13)

The updated estimate \( \theta_{n+1} \) can thus be computed by first computing the conditional expectation \( E_{S|Y, \theta_n} (\ln (P(Y,s|\theta))) \) of the hidden variables \( S \) given the current estimate of the model parameters \( \theta_n \) and the observed variables \( Y \) (E step), then maximizing that expression with respect to \( \theta \) (M step).

### 2.8 Derivation of the EM algorithm in the case of a HMM

Let \( \eta_{s_t, y_t} \triangleq P(y_t|s_t, \eta) \). In order to simplify the derivation, let us consider the following notations

\[
\eta_{s_t, y_t} = P(y_t|s_t, \eta) = \prod_{i,j=1}^{N} [\eta_{ij}]^{s_i^t y_j^t}
\]

(2.14)

\[
T_{s_t, s_{t+1}} = P(s_{t+1}|s_t) = \prod_{i,j=1}^{N} [T_{ij}]^{s_i^t s_j^{t+1}}
\]

(2.15)

\[
\pi_{s_0} = P(s_0) = \prod_{i,j=1}^{N} [\pi_i]^{s_0^i}
\]

(2.16)

where \( s_i^t = \delta_{s_t}^i \) and \( y_i^t = \delta_{y_t}^i \). We begin by writing the complete log likelihood to discover the form of the M step estimates, as well as the sufficient statistics that are needed for the E step. We have:

\[
\ln(P(s, y)) = \ln \left( \pi_{s_0} \prod_{t=0}^{T-1} T_{s_t, s_{t+1}} \prod_{t=0}^{T} P(y_t|s_t, \eta) \right)
\]

\[
= \ln \left( \prod_{i,j=1}^{N} [\pi_i]^{s_0^i} \prod_{t=0}^{T-1} \prod_{i,j=1}^{N} [T_{ij}]^{s_i^t s_j^{t+1}} \prod_{t=0}^{T} \prod_{i,j=1}^{N} [\eta_{ij}]^{s_i^t y_j^t} \right)
\]

\[
= \sum_{i,j=1}^{N} s_i^0 \ln \pi_i + \sum_{i,j=1}^{N} \left( \sum_{t=0}^{T-1} s_i^t s_j^{t+1} \right) \ln T_{ij} + \sum_{i,j=1}^{N} \left( \sum_{t=0}^{T} s_i^t y_j^t \right) \ln \eta_{ij}
\]

(2.17)
Maximizing the complete log likelihood (2.17) decouples into smaller problems of the form

$$\max_{x_1, \ldots, x_N} \sum_{k=1}^{N} a_k \ln x_k \text{ subject to } \sum_{k=1}^{N} x_k = 1$$

Using a Lagrange multiplier, the solution of this maximization problem is a stationary point of

$$\Lambda(x, \lambda) = \sum_{k=1}^{N} a_k \ln x_k - \lambda \left( \sum_{k=1}^{N} x_k - 1 \right)$$

Taking the partial derivatives with respect to $x_k$ yields $\lambda = a_k / x_k$, i.e., $x_k = a_k / \lambda$. Then $\sum_k x_k = 1$ yields $\lambda = \sum_k a_k$, hence the solution to this simple maximization problem

$$x_j = \frac{a_j}{\sum_{k=1}^{N} a_k}$$

Therefore, we can derive a simple expression of the maximum likelihood estimates of the parameters $\theta = (\pi,T,\eta)$ in the case of complete observation (assuming variables $s_t$ are observed)

$$\hat{T}_{ij} = \frac{m_{ij}}{\sum_{k=1}^{N} m_{ik}} \quad (2.18)$$

$$\hat{\eta}_{ij} = \frac{n_{ij}}{\sum_{k=1}^{N} n_{ik}} \quad (2.19)$$

$$\hat{s}_i = s_{i0}^{0} \quad (2.20)$$

where $m_{ij} \triangleq \sum_{t=0}^{T-1} s_t^i s_{t+1}^j$ and $n_{ij} \triangleq \sum_{t=0}^{T-1} s_t^i y_t^j$ (these are called the sufficient statistics for $T_{ij}$ and $\eta_{ij}$, respectively).

However, since we do not have direct access to the values of $m_{ij}$, $n_{ij}$, and $s_{i0}^{0}$ due to the latent variables, we need to compute their expected value given the observed variables $y$ and the current estimate of the model parameters $\theta_n$.

Let us first derive the expected value of $n_{ij}$
\[ E(n_{ij}|y, \theta_n) = \sum_{t=0}^{T} E(s_i^t|y, \theta_n) y_j^t \]
\[ = \sum_{t=0}^{T} p(s_i^t|y, \theta_n) y_j^t \]
\[ = \sum_{t=0}^{T} p(s_i^t|y) y_j^t \text{ since } s_t \text{ is independent of the model parameters} \]
\[ \text{using the definition (2.4)} \]
\[ \Delta = \sum_{t=0}^{T} \gamma(s_i^t) y_j^t \] (2.21)

where we define \( \gamma(s_i^t) \triangleq \frac{\alpha(s_i^t) \beta(s_i^t)}{P(y)} \). The expectation of the \( n_{ij} \) variables can therefore be computed from the values of the \( \alpha \) and \( \beta \) variables. Similarly, the expectation of \( m_{ij} \) can be expressed as

\[ E(m_{ij}|y, \theta_n) = \sum_{t=0}^{T-1} E(s_i^t s_{t+1}^j|y, \theta_n) \]
\[ = \sum_{t=0}^{T-1} P(s_i^t s_{t+1}^j|y, \theta_n) \]
\[ = \sum_{t=0}^{T-1} P(s_i^t s_{t+1}^j|y) \]
\[ \Delta = \sum_{t=0}^{T-1} \xi(s_i^t, s_{t+1}^j) \] (2.22)

where we define \( \xi(s_i^t, s_{t+1}^j) \triangleq P(s_i^t, s_{t+1}^j|y) \), which can be expressed in terms of \( \alpha \), \( \beta \), and \( T \):

\[ P(s_t, s_{t+1}|y) = \frac{P(y|s_t, s_{t+1})P(s_{t+1}|s_t)P(s_t)}{P(y)} \]
\[ = \frac{P(y_0, \ldots, y_t|s_t)P(y_{t+1}|s_{t+1})P(y_{t+2}, \ldots, y_T|s_{t+1})P(s_{t+1}|s_t)P(s_t)}{P(y)} \]
\[ = \alpha(s_t)P(y_{t+1}|s_{t+1})\beta(s_{t+1}) T_{s_t, s_{t+1}} \] (2.23)
In summary, the sufficient statistics are calculated via the recursive forward-backward procedure. We can substitute the expression of the estimated sufficient statistics, (2.22) and (2.21), into the maximum likelihood formulas, (2.18) and (2.19), to obtain the M step of the EM algorithm in the case of a HMM. We obtain

$$
\hat{T}_{ij} = \frac{E(m_{ij}|y, \theta_n)}{\sum_{k=1}^{N} E(m_{ik}|y, \theta_n)}
= \frac{\sum_{t=0}^{T-1} \xi(s_t^i, s_{t+1}^j)}{\sum_{t=0}^{T-1} \sum_{k=1}^{N} \xi(s_t^i, s_{t+1}^k)}
= \frac{\sum_{t=0}^{T-1} \xi(s_t^i, s_{t+1}^j)}{\sum_{t=0}^{T-1} \gamma(s_t^i)}
$$

(2.24)

where we use the fact that $\sum_{k=1}^{N} \xi(s_t^i, s_{t+1}^k) = \gamma(s_t^i)$, and

$$
\hat{\eta}_{ij} = \frac{E(n_{ij}|y, \theta_n)}{\sum_{k=1}^{N} E(n_{ik}|y, \theta_n)}
= \frac{\sum_{t=0}^{T} \gamma(s_t^j) y_t^j}{\sum_{t=0}^{T} \sum_{k=1}^{N} \gamma(s_t^i) y_t^k}
= \frac{\sum_{t=0}^{T} \gamma(s_t^j) y_t^j}{\sum_{t=0}^{T} \gamma(s_t^i)}
$$

(2.25)

where $\sum_{k=1}^{N} y_t^k = 1$. The EM algorithm iterates between the forward-backward recursion (E step) and the updates (2.24), (2.25) and (2.20) (M step). The updates require $O(T)$ operations, and the forward-backward recursion requires $O(N^2 T)$ operations. Therefore, the EM algorithm has a time complexity of $O(N^2 T)$. 
3 Modeling Framework

In this section, we will describe the modeling framework of our system, and we will discuss the computational problems that arise and render the exact inference intractable.

3.1 Arterial network

We are interested in estimating travel times on an arterial network consisting of $L$ links indexed by $l \in \{1, \ldots, L\}$. We describe the state of link $l$ at time $t$ using a discrete variable $S_{l,t} \in \{1, \ldots, K\}$ (for instance with $K = 2$ we can use a binary state to describe the state of congestion of each link, $S_{l,t} = 0$ if link $l$ is congested, and $S_{l,t} = 1$ if the link is in free flow state). The state of the entire network at time $t$ is described using variable $S_t = (S_{1,t}, \ldots, S_{L,t})$. Let $\mathcal{S} = \{s^1, \ldots, s^N\}$ be the state space of the network (where $N = K^L$ is the total number of possible network states).

The evolution of traffic states is assumed to be a Markov process (the state at time $t + 1$ only depends on the state at time $t$: $P(S_{t+1}|S_0, \ldots, S_t) = P(S_{t+1}|S_t)$), and the system is modeled using a Hidden Markov Model. 

3.2 HMM

We assume that the discrete state of the network is hidden, meaning not directly observed. Instead, we observe a set of continuous random variables that result from a distribution depending on the discrete hidden state. In our case, we do not have direct access to the state of congestion of the link, we only observe a set of travel times along paths on the network, and we assume that the observed travel times result form a distribution whose parameters depend on the hidden state of congestion.

The Markov process can then be modeled using a transition model that describes the probability to transition to a state at time $t + 1$ given the state at time $t$, and an observation model that describes the probability that a response is observed at time $t$ given the state at time $t$. 

17
Figure 4: The HMM representation of the process. Each vertical slice represents a time step. Each horizontal slice represents a link. In each time slice, the square nodes represent the observed variables $y^l_t$, and the circle nodes represent the hidden state variables $s^l_t$.

3.2.1 Transition model

The transition process is described using a transition matrix $T_t$ defined as

$$T_t(s^i \rightarrow s^j) \triangleq P(s^j_{t+1}|s^i_t)$$

where $s^i_t$ is a shorthand notation for $S_t = s^i$.

3.2.2 Observation model

During observation, we only have access to a set of $H(t)$ random observable variables $Y_t = \{Y^1_t, \ldots, Y^{H(t)}_t\}$ (the number of observations $H(t)$ can vary depending on the time $t$). In practice, we use probe vehicle data where each observation consists of a path (a sequence of connected links) on the network, the start and end offset on the path, and the travel time along that path. We allocate the travel time across the links of the path to obtain a set of tuples $(l, x_i, x_f, \delta)$, each of which corresponds to the following event: travel time on link $l$ from position $x_i$ to position $x_f$ is $\delta$. Travel time allocation across the links is done by solving a simple optimization problem. See [4] for further details.

The observation process can be described by an observation model

$$O_t(s \rightarrow y) \triangleq P(y_t|s_t)$$
where $y_t$ is a shorthand notation for $Y_t = y$. We model the observation process as follows: we consider the travel time on link $l$ in state $s$ from $x_i$ to $x_f$ to be a random variable that has distribution $f_t^{l,s,x_i,x_f}$. We then decompose the travel time distribution

$$f_t^{l,s,x_i,x_f}(\delta) = g_t^{l,s}(\delta) \times \int_{x_i}^{x_f} \rho_l^t(x)dx$$

where $\rho_l^t$ is the probability distribution of vehicle locations on link $l$ at time $t$. $g_t^{l,s}$ is a distribution describing the total travel time on link $l$ in state $s$, and is chosen to be a Gaussian distribution. This decomposition assumes that the partial travel time on link $l$ is a fraction of the total travel time on the same link, and that the fraction is given by the probability distribution of vehicle locations: more time is spent on the parts of the link that have more vehicles.

**Vehicle location probability distribution** In order to estimate $\rho_l^t(x)$, the probability distribution of vehicle locations on link $l$ at time $t$, we use the model described in [4]. The probability distribution of vehicle locations is assumed to be a piecewise linear function

$$\rho_l^t = \begin{cases} \bar{\rho}_a & x \in [0, L - (l_r + l_{max})] \\ \bar{\rho}_a + \bar{\rho}_b \frac{x - (l_r + l_{max})}{l_{max}} & x \in [L - (l_r + l_{max}), L - l_r] \\ \bar{\rho}_a + \bar{\rho}_b & x \in [L - l_r, L] \end{cases}$$

where $L$ is the total length of the link, $l_r$, $l_{max}$ and $\bar{\rho}_a$ are parameters of the model to be evaluated ($\bar{\rho}_b$ is a function of the rest of the parameters: $\bar{\rho}_b = 2\frac{1 - \bar{\rho}_a L}{l_{max} + 2l_r}$).

Assuming the observations are independent, the observation model becomes

$$O_t(s \rightarrow y) = \prod_{(l,x_i,x_f,\delta) \in y} f_t^{l,s,x_i,x_f}(\delta) = \prod_{(l,x_i,x_f,\delta) \in y} g_t^{l,s}(\delta) \times \int_{x_i}^{x_f} \rho_l^t(x)dx$$

The observation model is thus described using a set of the following parameters: the mean and variance of the total travel time distribution $g_t^{l,s}$ for each link $l$ and state $s$ (a total of $2KL$ parameters), and the parameters $\bar{\rho}_a$, $l_r$ and $l_{max}$ of the probability distribution of vehicle locations $\rho_l^t$ for each link $l$ (a total of $3L$ parameters).

We assume that the processes are time-invariant during each 1-hour time slice: $T_t(s^i \rightarrow s^j)$ and $g_t^{l,s}$ do not depend on $t$, and that observations $Y_t$ only depend on the state $S_t$.

### 3.3 Belief state

To describe the knowledge we have about the model at time $t$ given a history of observed responses $y_{0:t} \triangleq (y_0, \ldots, y_t)$, we use a prior belief state $q_t$ and a posterior belief state $p_t$.
We are interested in estimating travel times on the network. This can be achieved by tracking the state of the system, i.e. maintaining the posterior belief state $p_t$ as new responses are observed (Bayesian tracking), and updating the parameters of the transition and observation models $T$ and $O$. These two steps correspond to the E step and M step of the Expectation Maximization algorithm.

### 3.4.1 Bayesian tracking (E step)

Tracking can be achieved through propagating the belief state $p_t$ using the transition model $T$ to obtain the prior belief state $q_{t+1}$

$$q_{t+1}(s^j) \triangleq P(s^j_{t+1} \mid y_{0:t})$$

$$= \sum_i P(s^i_{t+1} \mid s^j_t)P(s^j_t \mid y_{0:t})$$

$$= \sum_i T(s^i \rightarrow s^j)p_t(s^i) \Delta = T[p_t](s^j)$$

then conditioning on the current observed response $Y_{t+1} = y$ to update the posterior belief state

$$p_{t+1}(s^j) = P(s^j_{t+1} \mid y_{0:t+1})$$

$$= \frac{q_{t+1}(s^j)O(s^j \rightarrow y)}{\sum_i q_{t+1}(s^i)O(s^i \rightarrow y)} \Delta = O^y[q_{t+1}](s^j)$$

The tracking process (which corresponds to the E step in the Expectation Maximization algorithm) can be summarized as

$$p_t \xrightarrow{T[.]} q_{t+1} \xrightarrow{O^y[.]} p_{t+1}$$

where $T[.]$ and $O^y[.]$ are shorthand notations for the propagation and conditioning phases, respectively.

### 3.4.2 Updating the transition and observation models (M step)

First we start by updating the parameters of the probability distribution of the vehicle locations ($\rho^j_l$, $l^l_t$ and $l^l_{max}$ for each link $l$), by maximizing, for each link, the log likelihood
of the observed locations at time $t + 1$. Let $X^l_t$ be the set of observed vehicle locations on link $l$ at time $t$. The optimization problem is the following:

$$\max_{\bar{\rho}_l, \rho_{l, \max}} \sum_{x \in X^l_t} \ln \rho_l(x)$$

Then the transition matrix $T$ and the parameters of the observation model $O$ ($\mu^{l,s}$ and $\sigma^{l,s}$ for each link $l$ and state $s$) are updated by maximizing the log likelihood of the observed responses $y_{0:t+1}$ given the belief states $p_0, ..., p_{t+1}$. The optimization problem is the following:

$$\max_{\mu^{l,s}, \sigma^{l,s}} L(y_{0:t+1} | p_{0:t+1}) \triangleq \max_{\mu^{l,s}, \sigma^{l,s}} \sum_l \sum_s \sum_{0 \leq \tau \leq t+1} p_\tau(s) \left( \sum_{(l,x_i,x_f,\delta) \in y_\tau} \ln (g^{l,s}(\delta)) \right)$$

The problem is solved using the EM algorithm derived in section 2, which has a time complexity of $O(N^2T)$ where $N$ is the size of the network and $T$ is the time step.

### 3.5 Prediction

For short term prediction, we can use the transition matrix learned up to time $t_0$ and propagate the belief state $p_{t_0}$ up to time $t_0 + T$ (without the conditioning phase since we do not have access to any more observations starting at time $t_0$). Let $\mu_{t_0}^{t_0}(s^i) = P(s_{t_0+t}^i | y_{0:t_0})$. The propagation process is summarized as

$$p_{t_0} = \mu_{t_0} \xrightarrow{T} \mu_{t_1} \xrightarrow{T} \mu_{t_2} \xrightarrow{T} \mu_{t_T}$$

### 3.6 Computational intractability and approximate belief states

It is in general intractable to compute the exact belief state due to the size of the transition matrix ($N^2 = K^{2L}$). Besides, the EM algorithm requires $O(N^2T) = O(K^{2LT})$
operations, which is exponential in the size $L$ of the network. Thus we are interested in using an approximate belief state $\tilde{p}_t \in \mathcal{P}$ where $\mathcal{P}$ is a family of distributions over the state space $S$. Introducing an approximate belief state makes it possible to limit the size of the operations in each step of the EM algorithm as we will explain in the following section. To evaluate the quality of the approximate belief state $\tilde{p}_t$, it is frequent to use the Kullback-Leibler divergence (relative entropy)

$$D[p_t, \tilde{p}_t] = \sum_i p_t(s^i) \ln \frac{p_t(s^i)}{\tilde{p}_t(s^i)}$$
4 Boyen-Koller approximate inference

Boyen and Koller have proposed in [1] and [2] a general method for approximate inference on Hidden Markov Models and Dynamic Bayesian Networks. The idea is to use a family of approximate belief states that have a compact representation which takes advantage of the structure of the HMM. In this section, we adapt the BK algorithm to our specific problem, and describe the steps of the EM algorithm under the BK approximation. We also analyze the trade-off between the time complexity and the error induced by the approximation.

4.1 Approximate belief state

We assume that the state variables $S$ are partitioned into $C$ clusters of variables $S = \cup_{c=1}^{C} S_c$. The approximate belief state is defined to be a product of marginals $\tilde{p}_t^c$ over the clusters

$$p_t(s^i) = P(s^i | y_{0:t}) \approx \prod_{c=1}^{C} P(s_c^i | y_{0:t}) = \prod_{c=1}^{C} \tilde{p}_t^c(s_c^i) \Delta = \tilde{p}_t(s^i)$$

where $s_c^i$ is the canonical projection of state $s^i$ on the cluster $c$. Let $M = K^{\max_c |S^c|}$ be the maximum size of the marginal beliefs, where $|S^c|$ is the size of the cluster. Splitting the process into several subprocesses reduces the size of the belief states ($C$ belief states of maximum size $M$, instead of a single belief state of size $N = K^L$) as well as the transition matrices. Hence computation is more likely to be tractable. Besides, the error induced by the approximation (the distance between $p_t$ and $\tilde{p}_t$) remains bounded over time under certain assumptions that will be detailed below.

4.2 Transition model

Let us further describe the transition model under these assumptions. We assume that the state of cluster $c$ at time $t + 1$ only depends on the state of $N(c) \triangleq \cup_{l \in c} N(l)$ where $N(l)$ is the set of geographic neighbors of link $l$ (two road links are said to be geographic neighbors if they share a common intersection). Thus the transition model can be described using one transition matrix (of size $K^{ \mid N(c) \mid \times K^{ \mid S^c \mid} }$) per cluster.

4.3 Inference under the BK approximation

The approximate belief state is updated in the same way as the exact belief state, i.e. by propagating it through the transition model $\mathcal{T}[,]$ then conditioning on the observed response. However, the resulting posterior belief state $\hat{p}_{t+1}$ does not necessarily fall into the approximation family $\mathcal{P}$, thus we have to project it on $\mathcal{P}$ using a projection $\Pi[,]$

$$\tilde{p}_t \xrightarrow{\mathcal{T}[,]} \hat{q}_{t+1} \xrightarrow{\mathcal{O}p[,]} \hat{p}_{t+1} \xrightarrow{\Pi[,]} \tilde{p}_{t+1}$$

23
The new approximate belief state $\tilde{p}_{t+1}$ is computed as the product of marginal distributions over the subprocesses.

4.4 Time complexity of the approximate inference

Updating the components of the approximate belief state is done in the same way as the exact belief state, and the parameter estimation of the models is done by solving $C$ parameter estimation problems using the EM algorithm on a graph of maximum size $M$, thus each subgraph requires $O(M^2 T)$ operations. Therefore, the total time complexity of the approximate inference is $O(CM^2 T)$, where $C$ is the number of clusters, $M = K_{\text{max,[S^c]}}$ is the maximum size of the subgraphs, and $T$ is the current time step. When $M$ is fixed (a maximum size of the clusters is chosen), the number of clusters increases as a linear function of the network size $L$, and the time complexity is $O(LM^2 T) = O(LT)$.

4.5 Error induced by the approximate belief state

It is interesting to study the error introduced by the approximation. Intuitively, one expects to observe a trade-off between the quality of the approximation and the size of the clusters (thus the time complexity of the inference). Smaller clusters will result in faster inference but poorer estimation of the travel times. The Kullback-Leibler divergence is a natural measure of the quality of the approximation, and is used by Boyen and Koller in [1] to study the bounds on the induced error of the approximation.

4.5.1 Bounding the error in the case of a single process

One interesting result presented in [1] is that

$$D[p_{t+1}, \hat{p}_{t+1}] \leq (1 - \gamma)D[p_t, \hat{p}]$$

where $\gamma$ is the mixing rate of process $T$ defined as $\gamma_T = \min_{i_1,i_2} \sum_j \min[T(s^{i_1} \rightarrow s^j), T(s^{i_2} \rightarrow s^j)]$. The update procedure (through $T[.]$ and $O_g[.]$) contracts the distance of the approximate belief state to the actual belief state.

It is then important to choose an appropriate projection $\Pi$ such that

$$D[p_{t+1}, \tilde{p}_{t+1}] \leq \epsilon + D[p_{t+1}, \hat{p}_{t+1}]$$

for some $\epsilon$. One sufficient condition is that

$$\max_i \ln \frac{\hat{p}_t(s^i)}{\tilde{p}_t(s^i)} \leq \epsilon$$

Under these conditions, the error remains bounded: let $d(t) = D[p_t, \tilde{p}_t]$. Since $d(t+1) \leq \epsilon + (1 - \gamma)d(t)$, the error at time $t$ remains bounded $d(t) \leq \sum_{i=0}^{t} \epsilon(1 - \gamma)^i \leq \frac{\epsilon}{\gamma}$.
4.5.2 Extension to compound processes

For a compound process $T$ of $C$ weakly interacting subprocesses \( \{T_1, \ldots, T_C\} \), where each subprocess $T_c$ depends on at most $r$ others, and affects at most $q$ others, the previous result becomes

\[
d(t) \leq \frac{\epsilon}{(\gamma/r)^q}
\]

where $\gamma = \min \{\gamma_T\}$. These results show that weakly interacting subprocesses result in better approximations, it is therefore very important to define an appropriate partitioning of the process, hence a partitioning of the road network into weakly interacting subgraphs.
5 Partitioning the network into weakly dependent clusters

Partitioning the network into clusters is an important step, as it will affect the algorithm in several ways: smaller clusters make computation more tractable but are likely to increase the error incurred by the approximation of assuming them to be independent. The clustering strategy has to take into account the size of the clusters and the interaction between them.

The proposed algorithm takes into account the information provided by the observations (specifically, the path of each observation) to partition the network into clusters such that links that appear in a same path should belong to the same cluster.

5.1 Weighted graph of traffic interactions

Given a set $P$ of observed paths, where each path $p \in P$ is a sequence of connected links $p = (l_{i_1}, \ldots, l_{i_l(p)})$ ($l(p)$ is the length of path $p$), define the binary relation $l_i \stackrel{p}{\rightarrow} l_j$ as: $l_i$ and $l_j$ are consecutive elements of the sequence $p$. Then define $G$ as the weighted directed graph where each vertex $l$ represents a link, and each pair of geographically neighboring links $l_i$ and $l_j$ is connected by an edge $e_{i,j} = (l_i, l_j, w_{i,j})$ with weight

$$w_{i,j} = \frac{\# \{ p \in P | l_i \stackrel{p}{\rightarrow} l_j \} }{\# \{ p \in P | l_i \in p \} }$$

The weight of an edge $e_{i,j}$ is the proportion of vehicles that went to link $l_j$ from link $l_i$. This definition makes the weights normalized such that for each link $l_i$, the sum of the weights of outgoing edges is 1.

$$\forall i, \sum_j w_{i,j} = 1$$

5.2 Partitioning the weighted graph

For a connected component $G_c$ of the graph, let $V_c$ be the set of vertices and $E_c$ be the set of edges of subgraph $G_c$. Our objective is to partition $G$ into clusters $(G_c)_{1 \leq c \leq C}$, such that the removed edges have minimal weights, and the size of each cluster is at most $M$ ($\forall c$, $|V_c| \leq M$). One way to formalize this is to find the partitioning that minimizes the following loss function

$$L((G_c)_{1 \leq c \leq C}) = \sum_{c,c'} Ncut(G_c, G_{c'})$$

where

$$Ncut(G_c, G_{c'}) = \frac{\text{cut}(G_c, G_{c'})}{\sum_{l_i \in G_c} w_{i,j} + \sum_{l_i \in G_{c'}} w_{i,j}}$$

and

$$\text{cut}(G_c, G_{c'}) = \sum_{l_i \in G_c, l_j \in G_{c'}} w_{i,j}$$

26
We use the normalized cuts as the function to minimize by the partitioning algorithm, rather than using regular cuts, in order to have clusters that are balanced in size. Minimizing the cut function alone yields unbalanced clusters, while normalizing the cut function favors balanced clusters.

Computing the exact solution of the minimum normalized cut is NP-hard. We propose to explore two approximate solutions: a recursive top-down solution that recursively splits the graph into two clusters until obtaining the desired size, and an iterative bottom-up solution that starts from a trivial clustering where each cluster consists of a single link, and propagates the clusters according to specific rules.

Recursive approach

One possible approximation is to compute the minimum cut recursively (resulting in hierarchical cuts): define $Ncut(G)$ as:

\[
\begin{align*}
\text{if } (\#(V) \leq M) & \text{ then} \\
& \quad \text{return } \{G\} \\
\text{else} & \\
& \quad (G_1, G_2) = N\text{mincut}(G) \\
& \quad \text{return } Ncut(G_1) \cup Ncut(G_2) \\
\end{align*}
\]

where $N\text{mincut}(G)$ computes the minimizing normalized cut of $G$ into two clusters. This approach reduces the problem to partitioning a graph into two clusters using the Normalized cut.

Iterative approach

This approach has been described in [3], and can be considered a particular Markov Cluster Algorithm [6]. It consists in starting with a trivial clustering of the graph, in which each cluster contains a single link, and iteratively updating the clusters as follows: each link $l_i$ is affected to the cluster $G_{c_0}$ that has the greatest cumulative weight among neighbors of $l_i$: $c_0 = \arg\max_c \sum_{l_j \in N(l_i)} w_{i,j} d_{j,c}$ where $d_{j,c} = 1$ if $l_j \in G_c$ and 0 otherwise.

Let $A$ be the adjacency matrix of graph $G$ ($A_{i,j} = w_{i,j}$) and $D$ be a $L \times L$ matrix representing the clustering of the graph: $D_{i,c} = 1$ if $l_i \in G_c$, 0 otherwise (each row of $D$ represents an element, and contains a single 1, and each column of $D$ represents a cluster). Starting with a trivial clustering where each cluster contains a single element, $(D = Id_L)$, iteratively update $D$ by reassigning each vertex $l_i$, and remove any clusters that have size $\geq M$. This can be done by the simple algorithm:

where $\text{maxrow}(D)$ replaces each row $D_i = [D_{i,1}, \ldots, D_{i,L}]$ with $[0, \ldots, 0, 1, 0, \ldots, 0]$ where $c_0 = \arg\max_c (D_{i,c})$
while \( (D \text{ did not converge} \& \ G \text{ is not empty}) \) do
\[
D := D \cdot A \ //\text{propagate the clustering}
\]
\[
D := \maxrow(D) \ //\text{project the resulting clustering matrix}
\]
for (each new cluster \( D_{i,c} \)) do
\[
\text{if (size of the cluster exceeds } M \ (\sum_i D_{i,c} \geq M)) \ \text{then}
\]
remove the cluster from \( D \)
\end{if}
end for
end while

METIS algorithm

The METIS algorithm is an iterative algorithm that provides a fast approximation of the normalized minimum cut solution. We decided to use the METIS software \cite{METIS} to partition the weighted graph since it provides an algorithm that scales well with the size of the graph (a few seconds for graphs of several thousand nodes) and that yields good quality approximations. However, some of the returned subgraphs happen to be non geographically connected, which is counter-intuitive since all clusters should be, by definition, geographically connected (edges in the graph only appear between nodes that are geographically connected). Therefore, we decided to post-process the output of the METIS software in order to have geographically connected clusters. We modify the output of METIS as follows: for each subgraph, we only keep the largest connected component. The remaining nodes are then reassigned (in a random order) to the cluster that has largest cumulative weight. This post-processing ensures that each cluster consists of a single connected component and seems to actually improve the value of the loss function to minimize.

5.3 Partitioning results

We tested the clustering algorithm using the METIS software with the described post-processing on historical data of observations aggregated on one hour time slices, for each day of week, over a period of 3 months. The results are satisfactory: every cluster consists of a single connected component and corresponds approximately to a different neighborhood in San Francisco. Major arteries and smaller roads in their immediate proximity appear in the same clusters, as shown in the figures below.
Normalizing the weights of the graph does not seem to have a substantial effect on the output of the clustering, although the obtained clusters are different.

The partitioning that we obtained seems to be consistent with traffic observations: for instance, all sections of highway 80 (Bay Bridge) as well the connected road segments
in its proximity in the SoMa neighborhood, all appear in the same cluster as shown in
the figure below.

Figure 7: A single cluster has been isolated in this figure. It shows that the sections
of Highway 80 (Bay Bridge) and roads in its proximity in the SoMa neighborhood, all
appear in the same cluster.

6 Discussion and summary

The Boyen-Koller approximate inference on the HMM makes the inference (updating
the belief states during the E step and the model parameters during the M step) computa-
tionally tractable by partitioning the graph into several subgraphs of limited size, and
approximating the belief state of the entire system as a set of localized beliefs about its
different parts. With this approximation, the time complexity of the inference algorithms
is polynomial in the size of the network, as opposed to the exponential complexity of
the exact inference. Besides, the error induced by this approximation remains bounded
in time.

The use of the Boyen-Koller approximation promises to be a very effective way to
find the appropriate trade-off between the quality of the travel time estimation and the
speed of computation. Besides, using the factored form of the belief state provides a
perfect framework for using concurrent processing, as the inference can be performed
separately on each subgraph. This possibility is being explored by the Arterial Team,
and I have started the implementation of the BK inference engine in Scala, which pro-
vides a very neat way of integrating concurrent processing. The implementation is still
at its early stage, but it will be carried by other members of the team.
During my internship, I have adapted the BK algorithm to the specific HMM of the traffic model, and have presented a detailed description of all the steps of the estimation algorithm. The model assumptions used remain the same as in the model described in [4], except for the independence of the links within the network (this approximation corresponds to a particular case of the BK approximation where the subgraphs all consist of a single link. In this sense, the algorithm proposed here is a generalization of the one described in [4]). Another slight difference is the stationarity assumption: I proposed to assume that the transition and observation models are stationary during one-hour time slices (in stead of plain stationarity of the models), and use a different partition for the network graph for each corresponding time slice to account for possible changes in drivers’ behavior at different times of the day.

I have also addressed the problem of graph partition and proposed a new method for partitioning the network graph, by defining a weighted graph using the historical observations of travel paths. The partitioning results are satisfactory and reflect real traffic observations.

Now that we have a detailed description of the model and the approximate inference algorithm, as well as the appropriate partition of the road network into weakly dependent subgraphs, it should be straightforward to test the algorithm on real data to validate the theoretical results presented here. The implementation of the BK approximate inference is in progress, and several tests are planned in order to validate the results: comparing the time complexity of the exact inference and the approximate inference for increasing network sizes, to observe the polynomial complexity of the approximate inference versus the exponential complexity of the exact inference, as well as measure the error induced by the approximation for different sizes of the subgraphs, and validate that the error remains bounded in time.

Working within the Arterial team at CCIT has been challenging and very interesting and rewarding. I came to learn a lot about Statistical Learning theory and work on a very specific and interesting problem. Working on a real system and real data was also interesting and presented its own set of challenges, due to the complexity of the mobile millennium system. I have had a very positive feedback from the team who was satisfied with my work, and will carry its implementation on the Mobile Millennium system.
References


