ENHANCING RELIABILITY IN SEMANTIC COMMUNICATION: A STOCHASTIC APPROACH TO SEMANTIC-GRAPH MODELING

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

By

Sadık Yağız Yetim

September 2023

Enhancing Reliability in Semantic Communication: A Stochastic Approach to Semantic-Graph Modeling
By Sadık Yağız Yetim
September 2023

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Orhan Arıkan (Advisor)

Tolga Mete Duman

Umut Orguner

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

ENHANCING RELIABILITY IN SEMANTIC COMMUNICATION: A STOCHASTIC APPROACH TO SEMANTIC-GRAPH MODELING

Sadık Yağız Yetim
M.S. in Electrical and Electronics Engineering
Advisor: Orhan Arıkan
September 2023

Semantic communication is expected to play a critical role in reducing traffic load in future intelligent large-scale sensor networks. With advances in Machine Learning (ML) and Deep Learning (DL) techniques, design of semantically-aware systems has become feasible in recent years. This thesis focuses on improving the reliability of the semantic information represented in a graph-based language that was previously developed. Inaccuracies in the representation of the semantic information can arise due to multiple factors, such as algorithmic shortcomings or sensory errors, deteriorating the performance of the semantic extractor. This thesis aims to model the temporal evolution of semantic information, represented using the graph language, to enhance its reliability. Each unique graph configuration is treated as a distinct state, leading to a Hidden Semi-Markov Model (HSMM) defined over the state space of the graph configurations. The HSMM formulation enables the integration of prior knowledge on the semantic signal into the graph sequences, enhancing the accuracy in identifying semantic innovations. Within the HSMM framework, algorithms designed for graph smoothing, semantic information fusion, and model learning are introduced. The efficacy of these algorithms in improving the reliability of the extracted semantic-graphs is demonstrated through simulations and video streams generated in the CARLA simulation environment.

Keywords: Semantic Communications, Semantic Signal Processing, Semantic Graph Signals, Hidden Semi-Markov Models, Viterbi Algorithm.

ÖZET

ANLAMSAL İLETİŞİMDE GÜVENİLİRLİĞİ ARTTIRMA: ANLAMSAL-GRAFİK MODELLEMESİNE STOKASTİK YAKLAŞIM

Sadık Yağız Yetim
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Orhan Arıkan
Eylül 2023

Anlamsal iletişimin geleceğin büyük ölçekli akıllı sensör ağlarında trafik yükünü hafifletmede kritik bir rol oynaması beklenmektedir. Makine Öğrenmesi ve Derin Öğrenme alanlarındaki ilerlemeler sayesinde son yıllarda iletilen bilginin anlamını kavrayabilen sistemlerin tasarımı mümkün hale gelmiştir. Bu tez grafik tabanlı bir dil aracılığıyla temsil edilen anlamsal bilginin doğruluğunu artırmayı amaçlamaktadır. Algoritmik sorunlar ve sensör hataları anlamsal çıkarıcının gözlemlediği sinyalin grafik temsilinde hatalara yol açabilmektedir. Grafik dili kullanarak temsil edilen anlamsal bilginin zaman içindeki değişimini matematiksel olarak modellemeye ve anlamsal çıkarıcının başarısını artırmaya odaklanılmıştır. Sınırlı sayıdaki olası grafik yapıları üzerinde bir durum uzayı tanımlanmaktadır. Gözlemlenen grafik yapısının zaman içindeki değişimi bu uzayda bir Saklı Yarı-Markov Modeli olarak formüle edilmektedir. Bu yaklaşım öncül bilgileri anlamsal çıkarıcı tarafından üretilen grafik temsillerle birleştirmeyi sağlamaktadır. Bu sayede anlamsal çıkarıcının ürettiği grafik temsillerin başarısı artırılmakta ve gözlenen sinyaldeki anlamsal yenilikler daha isabetli bir şekilde tespit edilebilmektedir. Saklı Yarı-Markov Modeli çerçevesinde, hata düzeltme, anlamsal bilgi birleştirme ve model öğrenme için algoritmalar sunulmaktadır. Algoritmaların etkinliği, simülasyonlar ve CARLA platformunda üretilen video sinyalleri aracılığıyla doğrulanmaktadır.

Anahtar sözcükler: Anlamsal İletişim, Anlamsal Sinyal İşleme, Anlamsal Grafik Sinyalleri, Saklı Yarı-Markov Modeli, Viterbi Algoritması.

Acknowledgement

First and foremost, my profound gratitude goes to Prof. Orhan Arıkan. His unwavering support throughout my M.S. journey has been invaluable, and it was a privilege to study under his supervision. His invaluable feedback was instrumental in shaping this thesis.

I am deeply thankful to Prof. Tolga Mete Duman. Our regular meetings kept me grounded and focused. I am grateful for his patience and support in the last two years.

Heartfelt thanks to Prof. Umut Orguner for generously dedicating his time to serve on my thesis committee.

From the project group I want to thank Mehmetcan Gök, Arda Atalık, Dr. Mert Kalfa and Büşra Tegin (soon-to-be Dr.). Collaborating with such gifted individuals has been an invaluable experience.

I count myself truly privileged to have been educated at Bilkent University, benefiting from the wisdom of its remarkable faculty. Among these outstanding figures, I wish to extend special gratitude to Prof. Levent Onural, Prof. Sinan Gezici, and Prof. Serdar Yüksel. Their enlightening courses have been a wellspring of inspiration for me.

During my M.S studies I was also partially supported by TÜBİTAK under BİDEB 2210-A, which I am thankful for.

A heartfelt thanks to my friends for all the great memories. Being surrounded by a group of fun, honest and supporting people is another thing that I do not take for granted. Reflecting on my M.S. years, my gratitude goes out to Derin Erkan, who

expanded my horizons daily with his vast knowledge on subjects as diverse as aircraft carrier designs to RBMK reactors. A special mention to Elif Aygün for consistently challenging my right-wing perspectives on societal matters.

To my parents, words cannot capture the depth of my gratitude for the sacrifices and love you have showered upon me. Yasemin, my sister, has been a beacon of support, and her presence helped me get through the most challenging chapters of my life. Her brief residency in Ankara this past year has been a blessing for me.

Lastly, but by no means least, my feline and canine family members have brought immense joy and comfort. Efe, Rüzgar, Maya, (Old Man) Puik, Bamya, Bade, Haydut, Coco, and in loving memory of Pamuk, I am forever grateful for their unconditional love.

Contents

1 Introduction **1**

1.1 Thesis Contribution 5

1.2 Thesis Outline 6

2 Semantic and Goal-Oriented Communications **7**

2.1 Propositional Logic 8

2.2 Natural Languages 17

2.3 Graph-Based Representation of Semantic Information 19

2.3.1 Graph-Based Semantic Language 19

2.3.2 Goal-Oriented Signal Processing and Communication Framework 21

2.4 Goal-Oriented Communications 23

2.5 Chapter Summary 24

3 Hidden Semi-Markov Model for Semantic-Graphs 26

3.1 State Space Representation for Temporal Evolution of Semantic-Graphs 27

3.2 Hidden Semi-Markov Models for Graph Evolution 29

3.3 Notations and Definitions in Hidden Semi-Markov Models 31

3.4 HSMM-Based Methods for Improving Graph Accuracy 33

3.4.1 Error Correction and State Estimation in Graph Sequences: Viterbi Algorithm 34

3.4.2 Forward-Backward Algorithm for Graph Smoothing 40

3.4.3 Approximate Smoothing Algorithms for Graphs with Large State Space and Long Duration 46

3.4.4 Multi-Sensor Data Fusion for Semantic-Graphs 55

3.4.5 Learning Model Parameters from Observed Data 60

3.5 Chapter Summary 63

4 Empirical Validation of HSMM Algorithms: CARLA Experiments and MATLAB Simulations 66

4.1 MATLAB Simulations for HSMM Algorithms 67

4.1.1 Smoothing Performance of HMM and HSMM Algorithms . . . 70

4.1.2 Viterbi and Forward-Backward Algorithms for Semantic-Graph Smoothing 72

4.1.3 Reduced Time Complexity Smoothing Algorithms 75

4.1.4 Semantic Fusion 83

4.1.5 Model Learning 85

4.2 CARLA Experiments: Assessing HSMM Effectiveness in Realistic Video Scenarios 91

4.3 Chapter Summary 102

5 Conclusions and Future Directions **105**

List of Figures

1.1 Three levels of communication. The lowest level corresponds to the setting proposed by Shannon in [1]. The intermediate level depicts the semantic problem. Here, the receiver and transmitter must possess a shared knowledge base to ensure the accurate transmission of the intended meaning. The shared knowledge base becomes particularly significant in overcoming ambiguities that might be introduced due to semantic noise. Meanwhile, the highest level concentrates on the effectiveness of communication. The evaluation metric for this level takes into account how much the communication process helps in achieving the desired goals for both ends involved in communication. 2

2.1 Semantic Communication Model proposed by [2]. The sender and receiver are characterized as tuples (W_s, K_s, I_s, M_s) and (W_r, K_r, I_r, M_r) respectively. It's worth noting that the physical layer and encoding/decoding processes are ignored in this model, with the primary focus being on the semantic aspects of communication. 11

2.2 Graph language. Consider an example based on two component classes: cars and people ($C = C_1, C_2$). The graph language also includes two predicate classes: moving and stationary ($P = P_1, P_2$). The signal modality used for generating graph signals can be a video stream, and the associated attribute vectors of component nodes can include features such as position and velocity. 20

2.3 Atomic sub-graphs. The generated graph consists of multiple atomic sub-graphs. Car-1 and Person-1 components do not interact with any other component. Person-2 is driving Car-2 and only the relation among these two components is important. These three sub-graphs can be processed independently, thanks to their lack of interaction. . 21

2.4 Goal-Oriented Semantic Signal Processing Framework. The raw signal undergoes a pre-processing operation, which may include domain transformation and noise filtering, before its graph representation is obtained. The generated graph signal is then semantically filtered and post-processed to increase its reliability, while discarding information that is irrelevant to the specified goals. 22

3.1 State space representation for semantic-graphs. A graph consisting of a single component (person) and two predicates (standing, walking) is considered. In this example, semantic extractor generates either the Person-1 $\rightarrow$ Standing graph or the Person-1 $\rightarrow$ Walking graph. These two graph configurations are named as the first and second states of the related stochastic process. As long as the person keeps walking or keeps standing the graph representation stays the same hence there is not any graph-level semantic innovation. 28

3.2 Hidden Markov Model for the graph given in Figure 3.1. The underlying Markov Chain transits over the two states according to the state transition matrix **A** and initial probability distribution π. $a_{ij} = P(Q_{t+1} = s_j | Q_t = s_i)$ gives the probability of moving to state j from state i and $\pi_i = P(Q_1 = s_i)$ gives the probability of first state being i. At each time step, a graph representation (state) is observed according to the observation matrix **B**. $B_{ij} = P(O_t = s_j | Q_t = s_i)$ is equal to the probability of observing state j when the actual state is i. Underlying states and the observed states are defined over the same space consisting of graph configurations. 29

3.3 ED-HMM representing a graph with three states. At time t, State-3 is left and a state transition depending only on the transition probabilities $a_{ij} = P(Q_{t+1} = s_j | Q_t = s_i)$ occurs. After moving to the new state (State-2), the state duration is drawn from the related distribution $p_j(d)$ which gives the probability of occupying state j for d time steps. In this example, d is equal to four and at each of these four time steps an observation is made depending only on the correct state at that time instance according to the probability $B_{ij} = P(O_t = s_j | Q_t = s_i)$. . 30

3.4 M-Algorithm for ED-HMM with four states. D' is equal to three in this example and M value is chosen as two for the Explicit-Duration HMM M-Algorithm, therefore only the transitions from the two states with largest δ_t^* values at the last three steps is considered. To calculate $\delta_t^*(1)$, the two most likely states at time steps $t-3$, $t-2$, and $t-1$ are taken into account, which are $S'_{t-3} = \{S_1, S_3\}$, $S'_{t-2} = \{S_2, S_3\}$, and $S'_{t-1} = \{S_2, S_4\}$, respectively. Since the self-transition probability is zero, transition from State-1 at time $t-3$ is ignored. 48

3.5 Quantization of an exponentially decaying (Laplacian) distribution centered at 101. The CDF is partitioned into ten intervals of equal probability. The boundary points are further away at the edges as the probability decreases and they are closer around the middle, this enables to quantize the more likely values finer and less likely values in a more coarse way. 52

3.6 Multi-camera setup for the sensor fusion problem. The two cameras are placed at the opposite ends of a road. A tree blocks their view, preventing an accurate view of half the road. In this scene, there are two cars present, yet each camera can only detect one of them independently. By fusing the extracted graph sequences from both cameras to estimate the underlying semantic graph, instead of relying on two individually smoothed graphs that fail to leverage the correlation between the extracted sequences, a more accurate representation of the scene can be obtained. 57

4.1 Sub-graph used for MATLAB simulations. The sub-graph consists of two components: person and car. The P_0 predicate represents the situation where the component is out of the scene. The P_1 predicate indicates that the component is in the scene but not interacting with the other one. In the context of the person and car components, this corresponds to the person walking and the car being parked respectively. There is also a third predicate, P_2, which represents the scenario where the person is driving the car. This predicate indicates the interaction between the two components. The components can enter or leave the scene independently of each other, leading to four possible states represented by the combinations of connections to predicates P_0 and P_1. The fifth state of the sub-graph is when the person is driving the car, represented by the P_2 predicate. 68

4.2 Embedded Markov Chain of the HSMM corresponding to the sub-graph. The states of the sub-graph are expressed as tuples with 0, 1, or 2, representing the predicates to which the components are connected. In this representation, each state transition corresponds to a change in the component-predicate connections. It is important to recall that the embedded Markov Chain does not include self-transitions, meaning that a state cannot transition to itself. 69

4.3 State waiting time distributions for HMM and HSMM formulations. The graph (state) sequences are generated according to the Laplace waiting time distribution, represented by the blue curve, with parameters $\mu = 50$ and $b = 1$. The HSMM-Viterbi Algorithm employs this distribution for smoothing observed sequences. In contrast, the waiting times in the Markov setting must follow a geometric distribution. The self-transition probability for each state is chosen such that the mean of this distribution matches the mean of the Laplace distribution. The state waiting time in the HMM formulation is represented by the red curve, which differs significantly from the correct one. . . . 71

4.4 Waiting time distribution of each state. Duration is defined between 1 and 200. The location parameter (μ) is chosen as 100 and scale parameter (b) is set to 10. 73

4.5 Waiting time distribution. Duration is defined between 1 and 40. The location parameter (μ) is chosen as 20 and scale parameter (b) is set to 3. The same distribution is used for each state in the M-Algorithm simulations. 76

4.6 Waiting time distribution. Duration is defined between 1 and 600. The location parameter (μ) is chosen as 300 and scale parameter (b) is set to 25. The same distribution is used for each state in Quantized-Duration Algoritm simulations. 78

4.7 Waiting time distribution. Duration is defined between 1 and 250. The location parameter (μ) is chosen as 125 and scale parameter (b) is set to 15. The same distribution is used for every state in Quantized M-Algorithm simulations. 80

4.8 Waiting time distribution of each state. Duration is defined between 1 and 50, it is a Laplacian centered at 25 with scale parameter equal to 5. 83

4.9 Estimated waiting time durations for P_e value of 0.25. The blue curves represent the true distributions, red and yellow curves correspond to parametric and non-parametric estimations respectively. As anticipated, the parametric estimation yields smoother results for every state compared to the non-parametric method. Notably, the duration distribution for the fifth state shows oscillations in the non-parametric estimation. Below each graph, the true and estimated location (μ) and scale (b) parameters are given. 90

4.10 Waiting time distribution of three states used in CARLA simulations. The duration of the parked state is uniformly distributed between 10 and 19. For the slow and fast moving states, the waiting time is uniformly distributed between 5 and 9. These distributions are actually defined over integers (as pmfs), so the rising and falling edges in the plots can be ignored. 92

4.11 Snapshots from CARLA simulation. The sequence starts with the car standing still at the road's left end. Its speed changes based on the underlying state, and at times, it becomes stationary. As illustrated in (c), when the car is parked (or passing) under a tree, it goes undetected, leading to an incorrect graph representation. However, once the car resumes its movement, it becomes visible again, and the sequence terminates when the car exits the road on the right side. . . 94

4.12 Segmented versions of the images given in Figure 4.11. The bounding box of the car in the segmented images allows to detect its speed and assign the corresponding state (semantic-graph) to the observed scene. In instances like (c), where the car is not detected and thus does not have a bounding box, the observed state is labelled as DNE. 95

4.13 State sequences in CARLA simulations. True, observed, and smoothed sequences are illustrated using blue, red, and green curves, respectively, for five different simulations. Due to the shifts in the observed sequence, caused by the extractor, the smoothed sequence also experiences shifts. Apart from these shifts in transition instances, the smoothed sequence largely aligns with the true sequence. As a result, the smoothing algorithm provides a significantly more accurate representation of the underlying semantic information. 98

4.14 Confusion matrices of observed and smoothed states. Since the DNE state appears in the observed sequence, it is included in the confusion matrix for the extracted states. However, the confusion matrix for the smoothed states is a 3×3 matrix, as the DNE state is not present in either the smoothed or true sequences. The ratio of correctly identified states improves notably with smoothing, especially for the idle/parked state. However, the actual improvement is even greater than what is visible in the confusion matrix, due to the inconsistent delays present in the observed sequence. 99

4.15 Estimated waiting time duration distributions. The blue and red curves represent the true and estimated state duration distributions, respectively. 102

4.16 Confusion matrices of observed and smoothed states. The extracted sequences are smoothed with estimated parameters from 1028 other simulations. Slow and fast states are often confused with the idle/parked state. This is likely due to the delay in assigning speed to the car and estimation of DNE states as idle. 104

List of Tables

4.1 Error rate for HMM and HSMM Viterbi Algorithms. Each row presents the ratio of incorrect states to the total sequence length, $T = 20000$, for the corresponding sequence. These ratios, excluding the P_e values in the first row, represent averages calculated from 50 independent iterations for each unique error rate P_e. In each run, the same observed sequence is smoothed using the two algorithms. While the two algorithms perform similarly at low P_e values, the limitations of HMM become more apparent as P_e increases. 72

4.2 Error rate for MAP and ML estimators. Each row provides the ratio of incorrect states to the sequence length, $T = 10000$, for the corresponding sequence. These ratios, except the P_e values in the first row, represent the averages obtained from 100 independent iterations for each unique error rate (P_e). Second ratio is the average error of observed sequence which is almost identical to the P_e as expected. The third and fourth rows provide the ratio of time steps where the sequences smoothed by the Forward-Backward (MAP) and Viterbi (ML) algorithms deviate from the correct sequence. The final row gives the ratio of steps at which the sequences from the MAP and ML estimators differ. 74

4.3 Error rate and elapsed time for M-Algorithm simulations. Each row shows the ratio of incorrect states to the sequence length of $T = 10000$, except for the last column which indicates the time taken by the algorithm to smooth the observed sequence. The error ratios are averages obtained from 25 independent iterations for each distinct error rate (P_e). The smoothing performance of the M-Algorithm, even at $M = 5$, matches that of the Viterbi Algorithm while the M-Algorithm at $M = 5$ is over 72 times faster. Given that the elapsed time is not dependent on P_e, the elapsed time column represents the average from 250 runs for each algorithm. 77

4.4 Error rate and elapsed time for simulations using the Quantized-Duration ED-HMM Viterbi Algorithm. Each row gives the ratio of incorrect states to the sequence length ($T = 100000$). These error rates are the averages of 50 independent iterations for each P_e value. The last column is the average smoothing time in seconds for different n values, computed over 500 runs. As expected, smoothing performance declines with decreasing n. Notably, for very small P_e values, the smoothing error might surpass P_e. However, for larger P_e values (P_e greater than 0.0316), the Quantized Viterbi reduces the incorrect state ratio considerably and does it faster than the standard Viterbi Algorithm. 79

4.5 Error rate and elapsed time for ED-HMM Quantized M-Algorithm. . 82

4.6 Error rate for Semantic Fusion. Each row provides the ratio of incorrect states to the sequence length for the corresponding sequence. These ratios represent the averages obtained from 100 independent iterations for each P_e value. 85

4.7 Error rate for smoothing with estimated model parameters. Each row provides the ratio of incorrect states to the sequence length for the corresponding sequence. These ratios represent the averages obtained from 50 independent sequences for each unique error rate. The observation sequence is smoothed using both the perfect model and estimated models for parametric and non-parametric duration estimations. The error rate of smoothing with estimated models closely mirrors that of smoothing with the ideal model parameters. 88

Chapter 1

Introduction

Ever since Claude Shannon's seminal work in 1948 [1], Classical Information Theory (CIT) has provided the theoretical foundation for development of communication techniques. Taking inspiration from Shannon's mathematical formulation, Weaver proposed a three level categorization of communication in [3], with each level corresponding to a distinct degree of abstraction. The lowest level addresses the technical problem, which seeks to perfectly reproduce the source data, regardless of its content. The intermediary level is concerned with the semantic problem, emphasizing the accurate transmission of the meaning of the source data. The final and the most abstract level focuses on the effect of the source data, denoted as the effectiveness problem. Figure 1.1 illustrates the components of each level.

While the classical theory, the technical problem, has been extensively studied, the exploration of higher levels, such as Semantic and Goal-Oriented Communications, has gained attraction only recently. This newfound interest has been significantly fueled by the successful application and widespread adoption of Machine Learning (ML) and Deep Learning (DL) techniques across a range of fields. Demonstrated

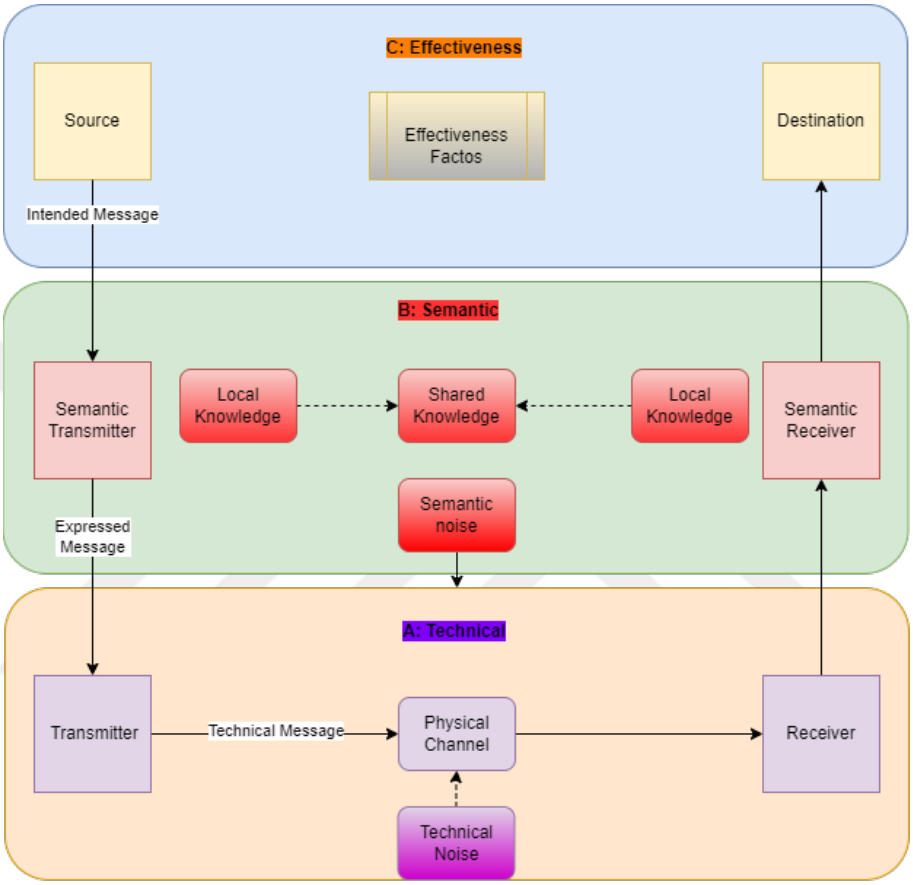


Figure 1.1: Three levels of communication. The lowest level corresponds to the setting proposed by Shannon in [1]. The intermediate level depicts the semantic problem. Here, the receiver and transmitter must possess a shared knowledge base to ensure the accurate transmission of the intended meaning. The shared knowledge base becomes particularly significant in overcoming ambiguities that might be introduced due to semantic noise. Meanwhile, the highest level concentrates on the effectiveness of communication. The evaluation metric for this level takes into account how much the communication process helps in achieving the desired goals for both ends involved in communication.

through various signal domains, such as speech [4], [5], [6], text [7], [8], [9], image [10], [11], [12] and video [13], [14], [15], ML/DL algorithms have achieved impressive results in the extraction of semantic information. This development enables the construction of communication systems that are equipped with semantic-aware capabilities. These systems can extract, filter, and transmit information based not on the traditional content-agnostic metrics, but on the meaning of the source data or its intended impact.

The concept of Semantic Communications, despite its current popularity driven by the adaptation of various ML/DL techniques, actually is as old as the classical theory. Earlier works on Semantic Information Theory (SIT) concentrate on logical statements to construct a common knowledge base and define information quantities based on logical probabilities [16]. However, recent developments are mostly focused on utilization of ML/DL techniques for building semantically-aware and goal-oriented communication systems.

Variety of alternative approaches are proposed to extract and transmit semantic information in various signal modalities in recent years. DeepSC, introduced in [17], is one such innovative system. Inspired from the recent advances in Natural Language Processing (NLP), DeepSC aims to create a semantically-aware communication system specifically designed for the accurate transmission of text-based semantic information. A similar system, DeepSC-S, is proposed for the transmission of speech signals as described in [18]. DeepSC-S, just like its text-based version, prioritizes transmitting the intended meaning accurately, rather than constructing an exact copy of the speech signal at the destination.

Originally introduced in [19], Scene-Graphs have emerged as a popular method for representing the semantic content in an image or a frame in a video stream. They allow for the identification of objects within the scene and describing the interactions among them. A more detailed exploration of the literature on Semantic

Communications is provided in Chapter 2.

The growing interest in Goal-Oriented and Semantic aspects of communication can be attributed to the anticipated bottlenecks and privacy concerns in future sensor networks, especially those consisting of interacting intelligent agents. These networks are expected to generate an enormous volume of data that cannot be transmitted using traditional content-agnostic communication techniques. Several works, including [20], [21], and [22], suggest that transitioning to communication systems with abstraction capabilities that can extract semantic information could provide a solution to these challenges.

With the aim of reducing the traffic load in future intelligent Machine-to-Machine (M2M) communication settings, a Goal-Oriented Semantic Signal Processing framework based on a hierarchical-graph language has been proposed in [23]. In this framework, component and predicate nodes of the graph language are used to describe the semantic information. Each component node belongs to a specific class, and predicate nodes express the relations among different components. Each node also has an accompanying attribute vector describing its features. The component, predicate nodes, and attributes can be defined according to the signal modality and application domain. Goals can be expressed in the graph language, which enables discarding redundant information and focusing on the essential nodes or specific graph structures. A semantic extractor is used for graph generation; this extractor can be an ML algorithm that detects and classifies the components in the sensor data and explains their relationships through predicate nodes. Thanks to the simplicity of the graph language, graph generation is expected to require moderate computational power.

1.1 Thesis Contribution

Generation of graphs in this framework is subject to errors arising from various factors, including algorithmic or sensing errors. For instance, issues with lighting in a video stream may cause misclassification of objects when the objects are positioned at a distance from the camera. Such mistakes in the graph representation can lead to false detections of innovations.

In this thesis, the focus is on developing algorithms to enhance the reliability of the temporal structure of the graph signals, which help to capture graph-level semantic innovations more accurately. To achieve this, the temporal evolution of the graph is modeled as a stochastic process, more specifically, as a Hidden Semi-Markov Model (HSMM), over the state space of graph configurations. This approach enables the incorporation of prior knowledge about temporal evolution of graphs and utilization of the continuity of the semantic-graph to correct errors in the extracted semantic information. This is particularly useful in addressing a shortcoming of many commonly used semantic extractors. These extractors typically operate on snapshots, focusing on information derived only from the most recent frame, while disregarding the correlation between consecutive frames and any prior semantic information on the input signal.

This thesis introduces HSMM-based algorithms aimed at correcting errors in the extracted semantic information expressed in a hierarchical-graph sequence. In addition to graph smoothing and model learning, an algorithm is proposed for fusing the semantic information generated by distinct sensors observing overlapping scenes. This offers a comprehensive approach to enhance the reliability of semantic extraction. Considering practical constraints, reduced complexity smoothing algorithms specifically designed for scenarios with computational limitations are also explored. Effectiveness of these algorithms in improving the accuracy of generated

semantic-graphs is demonstrated through a series of simulations and video streams. As anticipated, graph smoothing significantly reduces the proportion of incorrect semantic-graphs. Moreover, the reduced-complexity algorithms are found to perform comparably to the original smoothing algorithm, but with a notable reduction in computational complexity.

1.2 Thesis Outline

The rest of the thesis is structured as follows: The second chapter provides an overview of Semantic and Goal-Oriented Communications, focusing particularly on recent advances, and outlines the framework and graph language upon which the HSMM algorithms are constructed. The third chapter delves into the fundamentals of the HSMM and provides the derivation of algorithms used for smoothing, learning, and fusion. The fourth chapter showcases the results obtained from MATLAB simulations and video signals generated within the controlled CARLA environment. Finally, the fifth chapter concludes the thesis, summarizing the contributions made and discussing potential future directions.

Chapter 2

Semantic and Goal-Oriented Communications

The concepts behind Semantic and Goal-Oriented Communications, along with their potential benefits, were discovered as early as the CIT. However, the ability to define and extract information based on its meaning and impact only became attainable with recent advances in ML/DL techniques. For the transmission of information in a semantic context, a shared knowledge base, language, between the source and the destination is necessary.

This chapter offers a literature review on SIT, structured by the type of shared knowledge base employed. Additionally, it explores recent advancements in Semantic and Goal-Oriented Communications, focusing particularly on the Goal-Oriented Semantic Signal Processing framework and the graph-based language upon which the HSMM framework is constructed.

2.1 Propositional Logic

An alternative to the content-agnostic approach of the classical theory is proposed by Rudolf Carnap and Yehoshua Bar-Hillel in 1952 [16], [24]. They suggested utilization of a language system built on logical statements as a means to integrate the meaning of messages into the study of communication.

The proposed language system, denoted as $\mathcal{L}_n^\pi$, consists of n individuals (objects, people, etc.) labeled $a_1, ..., a_n$ and π one-place predicates $P_1, ..., P_\pi$ that characterize the properties of these individuals. The basic component in this language is called an atomic statement, it employs predicates to describe the properties of the individuals. The expression P_1a_1 is used to indicate that the individual a_1 has property P_1. More complex expressions called molecular statements are constructed by linking these basic statements using the following logical operators:

- not $\sim$ (negation),
- or $\vee$ (disjunction),
- and . (conjuction),
- if then $\supset$ (implication),
- if and only if $\equiv$ (equivalence).

Any statement within this language can be classified as either L-true (analytic, e.g. $P_1a_1 \vee \sim P_1a_1$), L-false (self-contradictory, e.g. $P_1a_1. \sim P_1a_1$), or logically indeterminate (factual, e.g. $P_1a_1 \vee P_1a_2$).

The universe described by $\mathcal{L}_n^\pi$ can assume $2^{n\pi}$ configurations depending on the properties assigned to each individual. This universe is characterized through logical

expressions called state-descriptions. A state-description is a conjunction containing every possible atomic statement or its negation (but not both) to provide a comprehensive description of the universe. For any logical statement j in $\mathcal{L}_n^\pi$, the set of state-descriptions where j applies is defined as the range of j.

A disjunction containing each $n\pi$ atomic statement or its negation (but not both) is referred to as a content-element. Each content-element is a negation of a state-description. A state-description is formulated as a conjunction of statements, hence all properties of each individual are known. In contrast, a content-element only L-implies itself, therefore it carries significantly less information compared to a state-description.

Semantic information carried by statement i in $\mathcal{L}_n^\pi$ is defined in terms of the content-elements. $Cont(i)$ is defined as the content of statement i and is equal to the class of content-elements L-implied by this statement. $Cont(i)$ is equal to the class of the negations of the state-descriptions contained in the range of $\sim i$. Relative information is also defined in terms of content-elements. The relative content of j relative to i is defined as $Cont(j/i) = Cont(i.j) - Cont(i)$.

The measurement of the amount of information is also crucial. Several potential candidates, called m-functions, are defined for the range of statements in [25]. One of the most plausible candidates among these m-functions is denoted as m_D. With this function, each content-element is assigned the same value, simplifying calculations. For instance, consider two independent coin flips, represented by individuals a_1 and a_2, with P_1 used to denote the result of landing on heads. In this case, both $m_D(P_1a_1)$ and $m_D(P_1a_2)$ would be equal to 1/2. Meanwhile, $m_D(P_1a_1 \vee P_1a_2)$ would be 3/4, and $m_D(P_1a_1.P_1a_2)$ would be 1/4.

Although the m-functions are developed for the range of statements, hence state-descriptions, they can be easily applied to content-elements thanks to the properties of logical negation. The information conveyed by statement i, in $\mathcal{L}_n^\pi$ is called the content-measure, denoted as $cont(i)$. The content-measure is defined as $cont(i) = m_D(\sim i)$, which is equal to $1 - m_D(i)$. The same concept can be used to measure relative information as well. Content-measure of j relative to i is defined as $cont(j/i) = cont(i.j) - cont(i)$. Which gives the amount of information obtained from statement j when statement i was already known. $cont(i)$ satisfies the following properties:

- $0 \leq cont(i) \leq 1$,
- $cont(i.j) = cont(i) + cont(j) - cont(i \vee j)$,
- $cont(i.j) \leq cont(i) + cont(j)$.

A second definition for the amount of information, which fulfills a different set of requirements, is given as $inf(i) = -log_2(m_D(i))$. This definition bears a resemblance to Shannon entropy in the classical theory. The key distinction from the classical definition is that this definition relies on the logical probabilities of statements, rather than their frequencies. The relative amount of information is defined as $inf(j/i) = -log_2(c(j,i))$, where $c(j,i)$ represents the degree of confirmation of hypothesis j based on evidence i. The confirmation degree $c(j,i)$ is defined as $m_D(i.j)/m_D(i)$ in [25].

In a more recent study, based on Carnap and Bar-Hillel's ideas, a semantic communication model based on propositional logic has been developed to define and analyze the semantic analogues of key concepts in classical theory such as entropy and channel capacity [2]. In this model, probabilities are assigned to logical sentences rather than the frequencies of symbols.

The proposed semantic communication model is given in Figure 2.1. The model

defines the source and receiver as tuples with four elements (W, K, I, M), where

- W represents the world model,
- K is the background knowledge base,
- I is the inference procedure,
- M is the semantic encoder/decoder.

Separate tuples are defined for the source and receiver, distinguished by subscripts s and r, respectively.

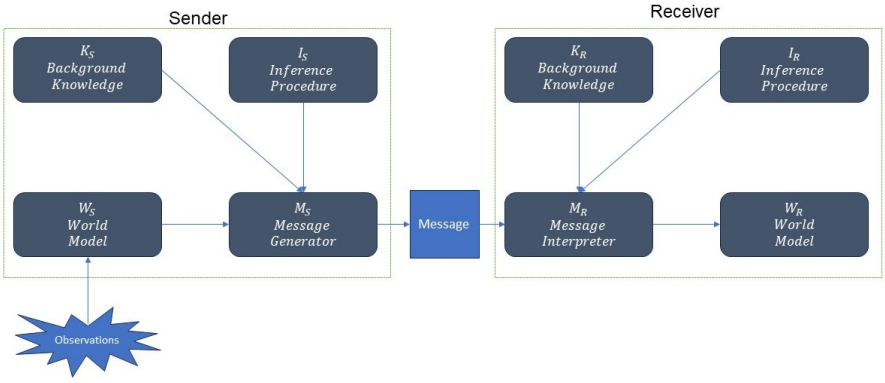


Figure 2.1: Semantic Communication Model proposed by [2]. The sender and receiver are characterized as tuples (W_s, K_s, I_s, M_s) and (W_r, K_r, I_r, M_r) respectively. It's worth noting that the physical layer and encoding/decoding processes are ignored in this model, with the primary focus being on the semantic aspects of communication.

In the proposed semantic communication model, a semantic source is defined as an entity capable of emitting messages using a fixed syntax. These messages are true according to the source's current state and inference capabilities. In this work, the source is assumed to be making factual statements in terms of logical statements. The source observes the world (denoted as w) and interprets these observations using its inference mechanism (I_s), in the context of its background knowledge (K_s). The

message generator (m_s), equivalent to a semantic encoder in this model, generates the message to be transmitted based on this interpretation. Various encoding techniques can be applied at this stage to reduce redundancy or improve reliability, similar to the classical communication setups. Upon receipt of the message, the receiver does not merely decode the syntax of the message in a mechanical way. Instead, it draws conclusions from the message using its own background knowledge. The physical channel over which the message is transmitted is omitted in this model.

In [2], there is no specific assumption or detailed discussion regarding the world models or the inference procedures in the model. It is stated that several factors such as loss in the semantic encoding, physical noise in the physical channel or a mismatch between the knowledge bases might cause an error in terms of semantic communication where a statement perceived as true by the sender (W_s, K_s, I_s, M_s) can be false at the receiver (W_r, K_r, I_r, M_r).

The proposed model presents a specific world model W, defined as a set of positive propositions that have a probability distribution μ. The semantic encoder, denoted by M_s, is responsible for generating a message x in such a way that this message is true in the observed world model, verified by the inference procedure I_s. This truthfulness is represented as $w \models x$. For any given message x, the set of worlds in which x is true is denoted by $W_x = \{w \in W | w \models x\}$. Then, the logical probability of message x is defined as:

$$m(x) = \frac{\mu(W_x)}{\mu(W)} = \frac{\sum_{w \in W, w \models x} \mu(w)}{\sum_{v \in W} \mu(v)}. \tag{2.1}$$

Without a restriction on the background knowledge, $\sum_{v \in W} \mu(v)$ is equal to 1.

Entropy definitions in this semantic communication model are similar those found in CIT, as well as the second definition of semantic entropy presented by Carnap and Bar-Hillel in [16]. The entropy associated with the model of the semantic source,

represented as $H_s(W)$, is defined as:

$$H_s(W) = -\sum_{w \in W} \mu(w) log_2(\mu(w)). \tag{2.2}$$

Next, the semantic entropy of a message x is given by:

$$H_s(x) = -\log_2(m(x)). \tag{2.3}$$

The effect of the background knowledge base is incorporated into the model mathematically through conditional probabilities. Under a certain knowledge base K, the set of possible world models is restricted to those where K is true. This results in a new distribution defined over this reduced set of world models. The conditional distribution of world models given the knowledge base K is defined as:

$$\mu'(w|K) = \frac{\mu(w)}{\sum_{v \in W, v \models K} \mu(v)}. \tag{2.4}$$

This equation adjusts the original distribution $\mu(w)$ by normalizing it with respect to the sum of probabilities of all world models where K is true. The conditional logical probability of a statement x given the knowledge base K is defined as:

$$m(x|K) = \frac{\sum_{w \in W, w \models K, x} \mu(w)}{\sum_{v \in W, v \models K} \mu(v)}. \tag{2.5}$$

Conditional semantic entropy is defined for the source and message using the conditional probabilities as follows:

$$H_s(W|K) = -\sum_{w \in W, w \models K} \mu'(w) log_2(\mu'(w)), \tag{2.6}$$

$$H_s(x|K) = -log_2(m(x|K)). \tag{2.7}$$

A semantic coding strategy is required for choosing the message x that will be used to represent the observed world model $w \in W$. It is assumed that the set

of allowed messages X is finite. The semantic coding strategy is characterized by the conditional distribution $P(X|W)$. A deterministic coding strategy is a specific type of semantic coding strategy. The distribution of the message x with the chosen coding strategy can be formulated as:

$$P(x) = \sum_{w \in W} \mu(w) P(x|w). \tag{2.8}$$

Furthermore, the entropy associated with messages X under this strategy is expressed as:

$$H(X) = -\sum_{x \in X} P(x) log_2(P(x)). \tag{2.9}$$

Given the definitions of entropy and conditional entropy, the following relationship between the model entropy and message entropy of the source can be derived:

$$H(X) = H(W) + H(X|W) - H(W|X). \tag{2.10}$$

In this equation $H(W|X)$ is a measure of semantic ambiguity which is the uncertainty about the world given a particular message and $H(X|W)$ is a measure of semantic redundancy which is the uncertainty about the message given a particular world state. The theorem suggests that the message entropy can be either greater or smaller than the model entropy, depending on the relation between ambiguity and redundancy in the semantic coding strategy.

The second theorem presented in [2], establishes the bound for lossless compression within the proposed semantic communication model. For a source with a message set X and message distribution $P(X)$, let $\overline{X}$ be defined as the smallest subset of X such that for every message x, a unique corresponding message $\overline{x}$ exists:

$$\forall x \in X,\ \exists \overline{x} \in \overline{X}\ s.t.\ x \leftrightarrow \overline{x}. \tag{2.11}$$

Furthermore, the distribution of message $\overline{x} \in \overline{X}$ can be expressed in terms of $P(x)$ as follows:

$$\overline{P}(\overline{x}) = \sum_{x\ s.t\ x \leftrightarrow \overline{x}} P(x). \tag{2.12}$$

The theorem states that for a semantic source with the message set X, there exists a coding strategy which generates semantically equivalent expressions using messages X' with message entropy $H(X') \geq H(\overline{X})$ and no such X' exists with message entropy less than $H(\overline{X})$. The difference $H(X) - H(\overline{X})$ determines the reduction in lossless compression.

When communicating over a channel, the received message may not always be accurate. In semantic communication, this inaccuracy can arise due to disturbances in the physical layer or from semantic noise, such as ambiguities in the message itself. For an input message x chosen by the source to describe the world state w, the message that the destination receives is denoted as y. The semantic error investigated in [2] is concerned with situations where a true message x is transmitted by the source ($w \models x$) yet the received message is false ($w \not\models y$). This discrepancy occurs over the set of world models shared by the sender and the receiver.

The objective of semantic channel coding is to minimize the semantic error described above. Mathematically, this is expressed as:

$$\max \sum_{w \models y} p(w, x, y). \tag{2.13}$$

This is equivalent to:

$$\max \sum_{w \models y} p(y|x)p(x|w)\mu(w). \tag{2.14}$$

Here, $p(y|x)$ is determined by the semantic channel and it is uncontrollable. Similarly, $\mu(w)$ represents the distribution of world model of the source and is thus fixed. Consequently, semantic channel coding primarily concentrates on maximizing the coding scheme modeled by $p(x|w)$. The ultimate goal is to select a message that is best equipped to tolerate noise for explaining the given world state w.

Analogous to the classical theory, the capacity for a noisy semantic channel is defined in [2]. This capacity provides a bound on the transmission rate that can be

achieved with an arbitrarily small semantic error. For a discrete memoryless semantic channel, the capacity is defined as:

$$C_s = \sup_{P(X|W)} \{I(X;Y) - H(W|X) + \overline{H_s(Y)}\}. \tag{2.15}$$

Here, the maximization is performed over the semantic coding strategy, $P(X|W)$. The term $I(X;Y)$ represents the mutual information between messages X and Y capturing the classical aspects of the channel. Meanwhile, $H(W|X)$ quantifies the semantic ambiguity. The expression $\overline{H_s(Y)}$ is defined as follows:

$$\overline{H_s(Y)} = \sum_y p(y) H_s(y). \tag{2.16}$$

It represents the average logical information of the received messages and serves as an indicator of the receiver's ability of interpreting these messages.

The semantic channel capacity can either exceed or fall below the technical capacity. This difference depends on the relation between the semantic ambiguity introduced by the semantic encoder ($H(W|X)$) and the inference capabilities of the semantic decoder ($\overline{H_s(Y)}$). The Semantic Channel Coding Theorem suggests that, even with a channel with low technical capacity, high-rate semantic communication can be achieved provided that there is minimal semantic ambiguity and strong inference capabilities.

The logic-based languages discussed so far might not sufficiently capture the semantics in human communication. Natural Language (NL) serves as a more fitting knowledge base for modeling human communication. The following section discusses some recent studies that utilize NLs to construct semantics-aware communication models.

2.2 Natural Languages

Recent studies utilizing NLs in the development of semantically-aware communication systems largely focus on joint semantic-channel coding and decoding. These systems are capable of extracting and encoding the meaning in text and speech signals. Most of these works integrate the successful DL-based NLP techniques into their communication paradigms. This section offers a summary of several recent publications focusing on the development of end-to-end (E2E) systems that are built around a Transformer Network for this purpose.

A semantic communication system named DeepSC is introduced in [17]. DeepSC is designed to accurately transmit text-based semantic information, specifically the meaning of sentences, over a noisy channel. Unlike traditional approaches, it emphasizes semantic distortion as the performance metric, rather than bit (symbol) level errors, with a focus on achieving efficient transmission. The transmitter and receiver are jointly designed as DNNs. The physical channels considered include AWGN and Rayleigh fading channels.

The cross-entropy (CE) loss is used to minimize the difference between transmitted and received sentences. In addition to preserving the meaning, DeepSC also aims to maximize the data transmission rate. This is accomplished by maximizing the mutual information between the transmitted symbols and the received symbols. The network is trained to minimize the following loss function to concurrently achieve both objectives:

$$\mathcal{L}_{total} = \mathcal{L}_{CE}(\mathbf{s}, \hat{\mathbf{s}}) - \lambda \, \mathcal{L}_{MI}(\mathbf{X}, \mathbf{Y}). \tag{2.17}$$

Evaluating accuracy regarding the preservation of meaning is challenging. In this study, two different metrics are employed to measure the similarity between the

original and reconstructed sentences. The first metric used is the BLEU score, which quantifies the differences in n-grams. A more sophisticated metric, which accounts for polysemy, is called Sentence Similarity. This metric computes the cosine distance between $\mathbf{s}$ and $\hat{\mathbf{s}}$ based on their BERT [26] representations. The presented simulation results indicate that the proposed DeepSC system outperforms traditional methods, which ignore the meaning of messages, especially under low signal-to-noise (SNR) conditions.

While it is plausible to use such DL-based techniques to transmit text-based information semantically, these systems are computationally demanding and not well-suited for devices with computational constraints. The authors address this challenge in a subsequent work. In [27], a simplified version of DeepSC is introduced. By employing techniques like sparsification and quantization, the network's complexity is reduced. This new system is named L-DeepSC. Besides DeepSC and L-DeepSC, which center on text-based information, the authors propose DeepSC-S for the transmission of speech signals, in [18]. This system differs slightly from other two. While DeepSC-S maintains the E2E framework, it replaces the Transformer network with another attention-based Neural Network, specifically SE-ResNet [28], suited for semantic encoding and decoding of the speech signals.

Using an NL as a shared base for semantic communication is intuitive. With recent advancements in NLP, DL-based methods have been successfully integrated into semantically-aware systems. However, successful NLP applications often require training complex networks with large amounts of data. To overcome these issues related to model complexity and the need for large-scale training data, a more straightforward knowledge base might be preferable. In such scenarios, graph-based languages provide an alternative for conveying semantic information.

2.3 Graph-Based Representation of Semantic Information

Utilizing a graph-based knowledge base eliminates the need for training with large datasets, which is a common challenge with NLs. Given the limited number of unique nodes and edges, the extraction and interpretation of graph representations is more straightforward compared to NLs.

An increasingly popular form of graph-based semantic representation is the scene-graph, originally introduced in [19]. Scene-graphs are used to summarize the semantic content of images and video frames. They describe the properties of individual objects as well as the relationships between pairs of objects within a given scene. In these graphs, nodes represent the objects, complete with their classes and attributes, edges capture the pairwise relations between objects.

The graph-based language emphasized in this thesis, introduced in [23], closely aligns with scene-graphs. The distinguishing feature is that relationships among objects are described through nodes as well. The state space in which the HSMMs operate corresponds to the set of possible configurations within this graph language.

2.3.1 Graph-Based Semantic Language

In the language proposed in [23], signal components are classified into classes and predicate nodes are utilized to express the states of component nodes as well as the relationships among different components. Each component node belongs to a class, and a signal's graph representation may contain multiple instances from the same component class. Similarly, predicate nodes also come from a predetermined class. It is possible to define an hierarchical attribute vector for each node to express

its features fully. Since the focus of this thesis is on the graph structure detailed discussion at the level of attributes is omitted. A simple example of semantic-graph can be found in Figure 2.2.

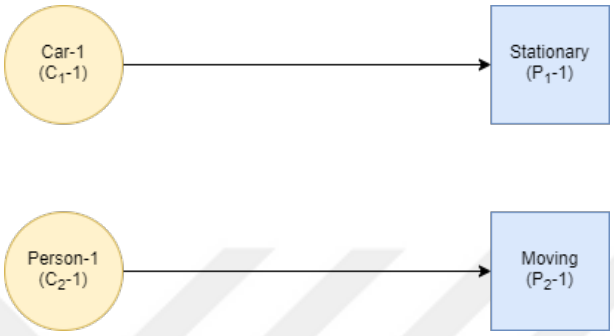


Figure 2.2: Graph language. Consider an example based on two component classes: cars and people ($C = C_1, C_2$). The graph language also includes two predicate classes: moving and stationary ($P = P_1, P_2$). The signal modality used for generating graph signals can be a video stream, and the associated attribute vectors of component nodes can include features such as position and velocity.

This language is defined to have two distinct levels of graph representation: class and instance. Instance level graph contains every component detected in the signal. In contrast, the class level graph offers a concise summary of the signal by representing different instances of the same class as a single component. The classes of component, predicate nodes and their corresponding attributes can be defined according to the signal modality, application and provided goals. The language consisting of node classes and the attributes can be defined by the user or it can be learned online. As a result of its flexibility, graphs in this language are expected to be simple, allowing for easy generation, filtering, and processing.

The generated graph in this language comprises multiple atomic sub-graphs, where each sub-graph contains only the components that interact with each other. This further simplifies graph processing, as only a few sub-graphs are expected to be present at the output of goal-oriented filtering and each sub-graph can be processed independently in parallel. Figure 2.3, provides an example of atomic sub-graphs.

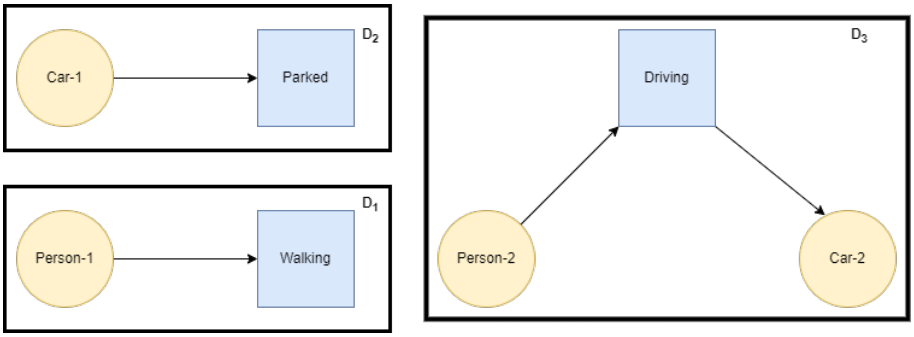


Figure 2.3: Atomic sub-graphs. The generated graph consists of multiple atomic sub-graphs. Car-1 and Person-1 components do not interact with any other component. Person-2 is driving Car-2 and only the relation among these two components is important. These three sub-graphs can be processed independently, thanks to their lack of interaction.

2.3.2 Goal-Oriented Signal Processing and Communication Framework

The goal-oriented semantic signal processing framework used for generating and processing graph signals is shown in Figure 2.4. This framework consists of a sequence of cascaded blocks. The first block is the pre-processing block, which helps obtaining a more robust input for the semantic extractor. This block may involve domain transformation and noise filtering of the raw input signal. Semantic extractor is the most crucial block in the framework. It generates the semantic description of the input signal in terms of the graph language. The semantic filtering block filters out the irrelevant information from the extracted semantic information. Semantic information typically consists of multiple disjoint graphs. Semantic filtering eliminates those graphs with no relevant information. Concerning the graphs that carry relevant information, semantic filtering functions as a masking operation, where only sub-graphs related to the goals are kept at the output. In addition to masking operations, semantic filtering can include searching for certain graph structures (motifs) or applying operations on the attribute vectors associated with nodes in the graph.

As the final step, filtered graph can be further processed before it is transmitted or stored in the post-processing step.

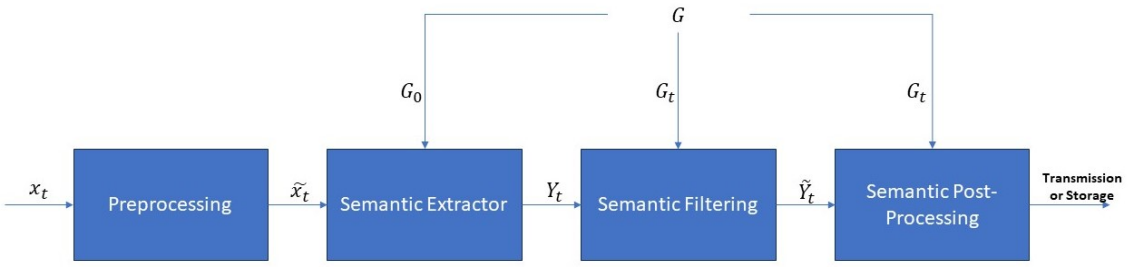


Figure 2.4: Goal-Oriented Semantic Signal Processing Framework. The raw signal undergoes a pre-processing operation, which may include domain transformation and noise filtering, before its graph representation is obtained. The generated graph signal is then semantically filtered and post-processed to increase its reliability, while discarding information that is irrelevant to the specified goals.

One of the primary advantages of this framework is that it allows to transmit or store semantic information related to the present goals only when a significant change affecting the goals occurs. These changes are referred to as semantic innovations in the graph-based language, which can occur at two levels: graph and attribute. Graph-level innovation corresponds to changes in the signal's graph representation such as arrival or departure of a component node, dissolution or emergence of edges connecting component and predicate nodes. Attribute-level innovation corresponds to significant changes in a node's feature that might affect the goal.

The accurate detection of semantic innovation is critical, as it directly influences decision-making based on the observed signals. However, the semantic-graph generated by the extractor can be flawed due to a range of factors such as sensory errors or limitations in the semantic extraction algorithm. This thesis aims to address these challenges by focusing on error correction in the generated graph sequences, thereby improving the identification of graph-level semantic innovations. For instance, consider a smart-city application where a decision-maker needs to know the number of

available parking spots. If the only source of information is a camera overlooking the parking lot, various issues like poor lighting conditions could result in vehicles at a distance not being detected. Consequently, the semantic graph may incorrectly suggest that a spot is available when it is not. By using prior information, such as the known presence of a stationary vehicle, the HSMM algorithms can correct this inaccurate representation, thereby enhancing the reliability of the semantic information used for decision-making.

2.4 Goal-Oriented Communications

Goal-Oriented Communications has been drawing attention alongside Semantic Communications. Although the two are closely related, Goal-Oriented Communications is more concerned with the effectiveness and impact of communication rather than the intrinsic meaning of the data. For instance, the framework introduced in [23] operates in the semantic domain, using semantic-graphs, but also incorporates goals during filtering, processing, and transmission. This section summarizes other recent works in Goal-Oriented Communications, particularly those addressing bottlenecks in sensor networks.

In [20], semantic content is evaluated based on its significance rather than its inherent meaning. The paper argues that both the characteristics of the source and the end goal that the transmitted message aims to achieve should be considered when designing communication systems. This is particularly emphasized in the context of future network systems consisting of intelligent actors. The authors advocate for a joint design of different blocks for sampling and transmission, taking the end goal into account. Various metrics are proposed to define and measure the semantics and utility of information, such as the Value of Information, Age of Information, and Relevance of Information. The Value of Information metric measures the utility of

a transmission in relation to the end goal, compared to the cost of the transmission. The Age of Information measures the timeliness or freshness of the information, while the Relevance of Information is concerned with the change in the observed process since the last received sample. The authors point to the importance of End-to-End systems in efficiently constructing semantics-aware networked systems.

Similar perspectives are explored in [22], where the authors emphasize the need for a joint approach that incorporates sampling, pre-processing, and transmission, all designed for semantics-aware communication. Semantic content is evaluated based on its impact rather than its inherent meaning. The authors define semantic value at three distinct levels: Source, Link, and System. At the Source level, events may be probabilistically equivalent but differ in significance. At the Link level, innate attributes are independent of usage, while contextual attributes depend on the goal. At the System level, End-to-End (E2E) differences are observed, particularly a mismatch between the generated and reconstructed information, with an emphasis on accurate and timely reconstruction aligned with the end goal. The authors emphasize the importance of developing semantic metrics that can effectively measure the qualitative attributes of information.

2.5 Chapter Summary

This chapter offers a comprehensive overview of Semantic Communications, tracing its roots from early works in Semantic Information Theory (SIT) that utilizes logic-based languages. It discusses the role of logical probabilities in defining new concepts like semantic entropy for quantifying semantic information. The chapter also highlights the potential of Natural Languages (NL) as a shared knowledge base, fueled by recent developments in DL-based NLP algorithms. The focus then shifts to the graph-based language that forms the foundation of this thesis, along with the

Goal-Oriented Semantic Signal Processing framework built upon it. The chapter concludes with a review of recent studies in Goal-Oriented Communications, with applications to networked systems.

Chapter 3

Hidden Semi-Markov Model for Semantic-Graphs

The state-space representation of graph configurations in the semantic language, developed in [23], allows for the modeling of semantic information as a stochastic process. In this setting, semantic innovations at the graph level are identified as state transitions. However, these transitions are not always accurately detected in practice due to sensory errors or limitations in the semantic extractor. Most semantic extractors in use operate on isolated snapshots and do not incorporate prior information about the process under examination. To address this issue, a Hidden Semi-Markov Model (HSMM) is formulated over the state-space of these graph configurations. This model accounts for the discrepancies between the true and observed graphs. This stochastic approach utilizes both prior information and temporal continuity of the semantic information to correct errors in the extracted graph sequences, thereby enhancing the reliability of the semantic graph.

In this chapter, the derivation of error correction algorithms is presented, with a particular focus on the Viterbi Algorithm. Taking practical computational constraints into account, algorithms with reduced time complexity are introduced for smoothing complex graphs with extended state durations and large state spaces. Furthermore, an algorithm is proposed for the fusion of semantic graphs generated by different sensors observing a common scene or signal. This chapter also includes a discussion on a model learning algorithm within the HSMM framework, designed for estimating model parameters from real-world, noisy observations.

3.1 State Space Representation for Temporal Evolution of Semantic-Graphs

Explicit definition of graph-level semantic innovations is necessary for their accurate detection and effective transmission in the proposed framework. This can be achieved by utilizing a state space representation, where each unique graph configuration is associated with a distinct state. With this approach, state transitions correspond to semantic innovations at the graph level. This idea is based on the observation that as long as the graph generated by a sensor stays the same, the underlying high-level semantic information does not change. Therefore, any decision based on the semantic information generated by the sensor should not be affected. Figure 3.1 provides an example illustrating the state space representation for a simple graph. Finally, this state space formulation paves the way for modeling the temporal evolution of the graph as a stochastic process.

A plausible assumption in the temporal evolution of graphs is that a state transition is solely dependent on the current and subsequent states, and unaffected by past information. This is equivalent to the Markov Property, and leads to the formulation of graph evolution as a Markov Chain. However, the observed graph is

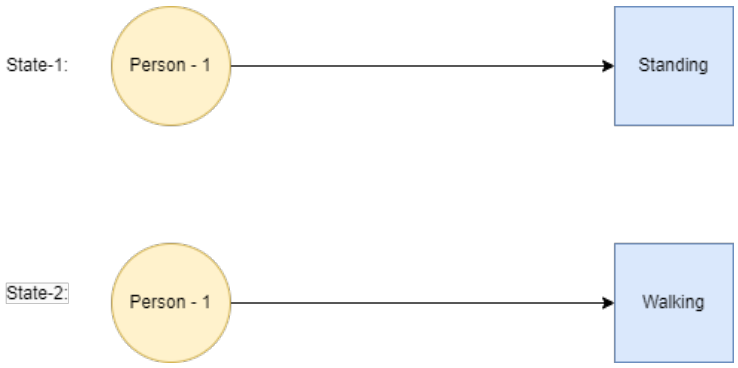


Figure 3.1: State space representation for semantic-graphs. A graph consisting of a single component (person) and two predicates (standing, walking) is considered. In this example, semantic extractor generates either the Person-1 $\rightarrow$ Standing graph or the Person-1 $\rightarrow$ Walking graph. These two graph configurations are named as the first and second states of the related stochastic process. As long as the person keeps walking or keeps standing the graph representation stays the same hence there is not any graph-level semantic innovation.

not always correct due to inaccuracies in the graph generation techniques or sensing errors, indicating that a Markov Chain may not be adequate for modelling the temporal evolution of the semantic-graph. For instance, a slow walking person in a video stream may be inaccurately represented as standing in the extracted graph. Given this discrepancy, the temporal evolution of the generated graph over the state space is more appropriately modeled as a Hidden Markov Model (HMM) rather than a Markov Chain. In HMM, the underlying (correct) graph states are assumed to evolve as a Markov Chain, and the observed graph state at each time step is considered a function of the correct state at that time instance [29]. Figure 3.2 illustrates an example of an HMM for a simple graph.

Moreover, in addition to the problem of discrepancies between the correct and observed graphs (states), the Markov Model presents another limitation when modeling the graph evolution in the proposed framework. The memorylessness property of the Markov Model results in state waiting times (sojourn times) following geometric distributions. As a consequence, this model struggles to accurately represent any state,

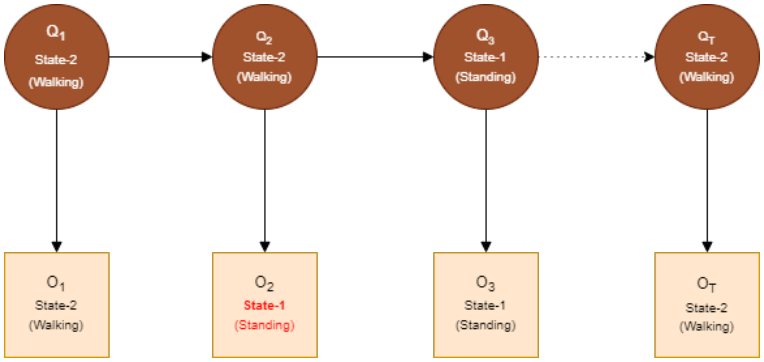


Figure 3.2: Hidden Markov Model for the graph given in Figure 3.1. The underlying Markov Chain transits over the two states according to the state transition matrix **A** and initial probability distribution π. $a_{ij} = P(Q_{t+1} = s_j | Q_t = s_i)$ gives the probability of moving to state j from state i and $\pi_i = P(Q_1 = s_i)$ gives the probability of first state being i. At each time step, a graph representation (state) is observed according to the observation matrix **B**. $B_{ij} = P(O_t = s_j | Q_t = s_i)$ is equal to the probability of observing state j when the actual state is i. Underlying states and the observed states are defined over the same space consisting of graph configurations.

graph configuration, which follows a waiting time distribution other than geometric. To more accurately represent a wider variety of waiting time distributions observed in practice, the underlying Markov Chain in the hidden model is replaced with a Semi-Markov Process. This updated model is known as the Hidden Semi-Markov Model (HSMM) in literature [30], [31]. Further details on the HSMM are provided in the following section.

3.2 Hidden Semi-Markov Models for Graph Evolution

A Hidden Semi-Markov Model (HSMM) is a generalization of the Hidden Markov Model (HMM) where the underlying process is Semi-Markov. This implies that state waiting times, or sojourn times, do not necessarily follow geometric distributions.

This thesis narrows its focus onto a straightforward, yet most commonly adopted, variant of HSMM, referred to as the Explicit-Duration Hidden Markov Model (ED-HMM). In the ED-HMM, the state transition probabilities depend only on the current state and the next state, and self-transition probability for any state is zero. The waiting time in each state is dependent only on the current state. At each time step, the observed output is determined by the underlying state at that specific time step [29], [32]. The waiting time for each state is assumed to be bounded to ensure practical applicability of the algorithms presented in this thesis. Figure 3.3 provides an example of an HSMM used for modeling the evolution of a semantic-graph.

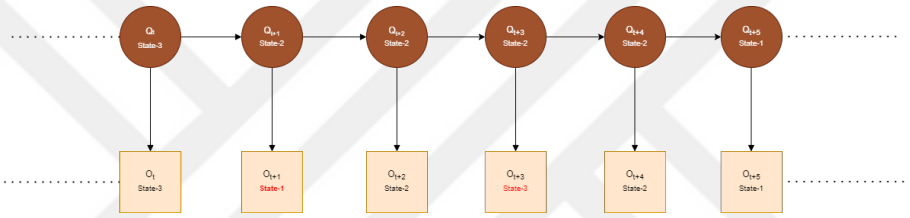


Figure 3.3: ED-HMM representing a graph with three states. At time t, State-3 is left and a state transition depending only on the transition probabilities $a_{ij} = P(Q_{t+1} = s_j | Q_t = s_i)$ occurs. After moving to the new state (State-2), the state duration is drawn from the related distribution $p_j(d)$ which gives the probability of occupying state j for d time steps. In this example, d is equal to four and at each of these four time steps an observation is made depending only on the correct state at that time instance according to the probability $B_{ij} = P(O_t = s_j | Q_t = s_i)$.

By representing the generated graph sequence over time as an HSMM, the framework benefits from exploiting the temporal continuity of semantic information and the incorporation of prior knowledge concerning the semantic signal. This utilization of the HSMM allows the accuracy of the extracted graph sequence to be enhanced through the application of algorithms generally employed for hidden models. The idea of employing hidden models for graph evolution and graph smoothing in the proposed framework was initially introduced using HMMs in [33].

The feasibility of the smoothing algorithms for use with extracted graphs is anticipated due to the adoption of goal-oriented filtering framework. The principle

behind goal-oriented filtering suggests that only few atomic sub-graphs in the original graph with smaller state spaces will survive goal-filtering. Additionally, the absence of interaction among components in distinct atomic sub-graphs allows for parallel processing, as illustrated in Figure 2.3.

Throughout this thesis, the terms Hidden Semi-Markov Model (HSMM) and Explicit-Duration Hidden Markov Model (ED-HMM) are used interchangeably. It is important to acknowledge that HSMMs involve modeling the entrance time to the initial state and the departure time from the final state. The algorithms and results presented in this thesis are based on the simplifying assumption that the initial state is entered at time step $t = 1$, and the final state is exited at $t = T$ which correspond to the first and the final time steps of the sequence [30], [34].

Throughout the remainder of this chapter, algorithms for graph smoothing, fusion, and model learning are provided within the framework of HSMMs for semantic-graphs. To avoid confusion, the mathematical notation is explained prior to the derivation of the algorithms.

3.3 Notations and Definitions in Hidden Semi-Markov Models

The graph-based language, developed in [23], guarantees a bounded number of graph configurations, since it only involves a finite number of component and predicate nodes in practice. This characteristic allows the state space to be defined as an enumeration of potential graph configurations, denoted as $S = \{S_1, S_2, \ldots, S_N\}$. In the HSMM setup, both the hidden states representing the underlying, correct graph sequences (Q_t) and the extracted sequences (O_t) are defined over the set S.

In the context of HMMs, due to Markov Property, the observation sequence and the underlying state sequence are analyzed in discrete time steps at a time. Even algorithms that operate on the whole sequence are designed by examining individual time steps. Conversely, with HSMMs, the Markov Property is not applicable in general. This necessitates the examination of the observation and the state sequences in terms of segments, rather than individual time steps.

Throughout this thesis, the notation employed in [30] and [34] is adopted to represent segments of state and observation. In this notation, the use of brackets implies state transitions. For instance, $Q_{[t_1:t_2]} = S_i$ indicates that the state remains S_i between time steps t_1 and t_2, and it differs from state i at $t_1 - 1$ and $t_2 + 1$. Conversely, in the absence of brackets, as in $Q_{t_1:t_2} = S_i$, it indicates that the state remains at i between t_1 and t_2, but it might be different in the preceding and succeeding time steps. The same concept is applicable to the observation sequence. For example, $O_{[t_1:t_2} = S_i$ suggests that the observation stays at S_i between t_1 and t_2, and while it might remain the same beyond t_2, it transitions from another state to S_i at time step $t_1 - 1$.

In an HMM, the hidden states (Q_t) evolve as a Markov Chain according to the state transition matrix $\mathbf{A}$ and initial probability distribution π. An observation is made at each time step in accordance with the observation matrix $\mathbf{B}$. These model parameters are also directly applicable to the HSMM, and are defined in (3.1) - (3.3).

HSMM differs from HMM due to the explicit definition of waiting time distributions for each state, as presented in (3.4). When the underlying process transitions to a new state S_j, the state duration is sampled from the distribution $p_j(d)$. For practicality, the waiting time distribution is assumed to be bounded above by a constant D'. The state duration distribution $p_j(d)$ is defined over the set $D = \{1, 2, \ldots, D'\}$, representing potential durations of a state.

In the HSMM framework, $u_t(j,d)$ denotes the likelihood of observing a sequence spanning d time steps until time t, provided the underlying state is S_j throughout this time interval. This thesis employs an ED-HMM as a specific variant of HSMM, which assumes conditional independence for the observation sequence [34]. This implies that individual observations within a segment are treated as conditionally independent when the underlying state is known. Consequently, $u_t(j,d)$ can be formulated as the product of individual observation probabilities, as defined in (3.5).

$$A_{ij} = P(Q_{[t+1} = s_j | Q_{t]} = s_i) \tag{3.1}$$

$$B_{ij} = b_i(j) = P(O_t = s_j | Q_t = s_i) \tag{3.2}$$

$$\pi_i = P(Q_1 = s_i) \tag{3.3}$$

$$p_j(d) = P(Q_{t+d]} = s_j | Q_{[t} = s_j) \tag{3.4}$$

$$u_t(j,d) = P(O_{t-d+1:t} | Q_{t-d+1:t} = s_j) = \prod_{\tau=t-d+1}^{t} b_j(o_\tau) \tag{3.5}$$

For in-depth information on definitions, algorithms, and variations of HMMs, please refer to [29]. [30] provides a detailed tutorial on HSMMs.

3.4 HSMM-Based Methods for Improving Graph Accuracy

The HSMM framework offers a solid mathematical foundation for incorporating prior knowledge about graph evolution and modelling the continuity of semantic-graphs. The usefulness of the HSMM framework lies in its flexibility; well-known algorithms utilized to address a range of problems, including state sequence estimation and

model learning, can be effectively applied to semantic-graphs. This section is dedicated to exploring these algorithms within the context of ED-HMMs, addressing challenges such as smoothing graph sequences, model learning, and semantic fusion. In addition, to ensure efficient computation in practical situations with computational constraints, reduced complexity variants of the Viterbi-based smoothing algorithm are presented.

3.4.1 Error Correction and State Estimation in Graph Sequences: Viterbi Algorithm

In the proposed framework, errors in semantic extraction can result in the creation of incorrect graphs. This causes a mismatch between the observed graph (state) sequence and the true underlying graph (state) sequence. However, prior statistical knowledge on the graph language can be leveraged to correct these errors and improve the accuracy of the extracted graph sequence.

Assuming that the required model parameters ($\mathbf{A}$, $\mathbf{B}$, $\boldsymbol{\pi}$, $p_j(d)$) are available, the most likely state sequence ($\hat{Q}$) that generates the observed sequence (O) can be estimated using the Viterbi Algorithm, which is similar to the case of HMMs [35]. This involves finding the state sequence that maximizes the probability of observing the given output sequence ($O_{1:T}$), as expressed in (3.6) and (3.7).

$$\hat{Q} = \arg\max_{Q_{1:T}} P(Q_{1:T}|O_{1:T}) = \frac{P(Q_{1:T}, O_{1:T})}{P(O_{1:T})} \tag{3.6}$$

$$\hat{Q} = \arg\max_{Q_{1:T}} P(Q_{1:T}, O_{1:T}) \tag{3.7}$$

The Viterbi Algorithm, in the context of a general HSMM, is presented in [30]. In this thesis, however, we focus specificially on ED-HMMs, for which a less complex

version can be derived by exploiting the ED-HMM's assumptions. This algorithm requires to find the probability of the most probable state sequence for a given output sequence. The first step is to define the probability of the most probable state sequence that ends at time t, leaving the j^{th} state, and yielding the output sequence $O_{1:t}$ denoted as $\delta_t^*(j)$, as follows:

$$\delta_t^*(j) = \max_{Q_{1:t-1}} P(Q_{t]} = s_j, Q_{1:t-1}, O_{1:t}). \tag{3.8}$$

In the Viterbi Algorithm, as described in [29], the most critical step is the recursive maximization of the intermediate variable $\delta_t^*(j)$. In the ED-HMM setup, this maximization needs to be performed over $Q_{1:t-1}$, as the Markov Property is not necessarily true. Maximizing over $Q_{1:t-1}$ is effectively the same as maximizing in relation to both the last state's duration d and the preceding state sequence $Q_{1:t-d}$:

$$\delta_t^*(j) = \max_{d \in D} \max_{Q_{1:t-d}} P(Q_{[t-d+1:t]} = s_j, Q_{1:t-d}, O_{1:t}). \tag{3.9}$$

In (3.9), the state preceding s_j is known to be left at $t-d$. Therefore, the maximization over $Q_{1:t-d}$ can be further divided into maximization over the previous state (s_i) and $Q_{1:t-d-1}$ as:

$$\delta_t^*(j) = \max_{s_i \in S \setminus \{s_j\}, d \in D, Q_{1:t-d-1}} P(Q_{[t-d+1:t]} = s_j, O_{t-d+1:t}, Q_{t-d]} = s_i, Q_{1:t-d-1}, O_{1:t-d}). \tag{3.10}$$

By expressing the joint probability in (3.10) in terms of conditional probabilities, the desired recursive expression can be obtained in terms of previous δ^* values and model parameters as follows:

$$P(Q_{[t-d+1:t]} = s_j, O_{t-d+1:t}, Q_{t-d]} = s_i, Q_{1:t-d-1}, O_{1:t-d}), \tag{3.11}$$

$$= P(O_{t-d+1:t}|Q_{[t-d+1:t]} = s_j)P(Q_{[t-d+1:t]} = s_j|Q_{t-d]} = s_i)P(Q_{t-d]} = s_i, Q_{1:t-d-1}, O_{1:t-d}). \tag{3.12}$$

The first expression in (3.12) represents the probability of observing the provided output section between $t-d+1$ and t when the underlying state is j. As specified in (3.5), it is denoted as $u_t(j,d)$.

In the context of ED-HMM, it is important to note that the state duration distribution is solely dependent on the current state, whereas the transition probability is influenced by the states themselves, not by their durations. Therefore, the second expression in (3.12) can be expressed in terms of waiting time distribution and the state transition probabilities, as given in (3.13) and (3.14).

$$P(Q_{[t-d+1:t]} = s_j|Q_{t-d]} = s_i) = P(Q_{t]} = s_j|Q_{[t-d+1} = s_j)P(Q_{[t-d+1} = s_j|Q_{t-d]} = s_i) \tag{3.13}$$

$$P(Q_{[t-d+1:t]} = s_j|Q_{t-d]} = s_i) = p_j(d)a_{ij}. \tag{3.14}$$

The final term in (3.12), when maximized over $Q_{1:t-d-1}$, equals $\delta^*_{t-d}(i)$:

$$\max_{Q_{1:t-d-1}} P(Q_{t-d]} = s_i, Q_{1:t-d-1}, O_{1:t-d}) = \delta^*_{t-d}(i). \tag{3.15}$$

Therefore, the desired recursive expression for $\delta^*_t(j)$ can be formulated as follows:

$$\delta_t^*(j) = \max_{s_i \in S \setminus \{s_j\}} \max_{d \in D} u_t(j,d) a_{ij} p_j(d) \delta_{t-d}^*(i). \tag{3.16}$$

Similar to the Viterbi Algorithm for HMMs, the most probable state sequence can be determined through backtracking. At each recursive step, pointers to the previous state (i^*) and state duration (d^*) pair that maximize $\delta_t^*(j)$ values are stored. These are subsequently used to backtrack from the end to retrieve the most likely state sequence. The full algorithm, called Explicit-Duration HMM Viterbi, is presented in Algorithm 1.

An important part of this algorithm is the computation of $u_t(j,d)$ values. If this is done naively, it greatly increases the complexity of the algorithm. Fortunately, the conditional independence assumption of observations in the ED-HMM setup enables efficient computation of $u_t(j,d)$ values through a recursive formula, as discussed in [36]:

$$\begin{aligned} u_t(j,d) &= P(O_{t-d+1:t} | Q_{t-d+1:t} = s_j), && (3.17)\\ &= P(O_t | Q_{t-d+1:t} = s_j, O_{t-d+1:t-1}) P(O_{t-d+1:t-1} | Q_{t-d+1:t} = s_j), && (3.18)\\ &= b_j(o_t) u_{t-1}(j, d-1) = P(O_t | Q_t = s_j) P(O_{t-d+1:t-1} | Q_{t-d+1:t-1} = s_j). && (3.19) \end{aligned}$$

Since the waiting time for each state is bounded by D', it's only required to keep track of the $u_t(j,d)$ values for the last D' steps. For ease of implementation, these $u_t(j,d)$ values are stored in a matrix format, where each row represents a different state and each column represents a different waiting time as given in (3.20).

Algorithm 1 Explicit-Duration HMM Viterbi Algorithm

Input : $\mathbf{A}, \mathbf{B}, \pi, p_j(d), \mathbf{O}$ ▷ Model parameters and output sequence
Output : $\hat{\mathbf{Q}}$ ▷ Most likely state sequence
Initialization
$u_1(j,1) = b_j(O_1)$ ▷ Initialize $u_t(j,d)$
for j = 1 upto N **do**
  $\delta_1^*(j) = \pi_j p_j(1) u_1(j,1)$
  $\psi_{1,j}(i,d) = (-1,1)$ ▷ Pointer to previous state and current duration
end for
Recursion
for t = 2 upto D' **do**
  Update $u_t(j,d)$
  for j = 1 upto N **do**
    $\delta_t^1(j) = \pi_j p_j(t) u_t(j,t)$ ▷ Occupying state j from start
    $\delta_t^2(j) = max_{i \neq j} max_{d<t} \{u_t(j,d) p_j(d) a_{ij} \delta_{t-d}^*(i)\}$ ▷ Moving to state j
    $\delta_t^*(j) = \max\{\delta_t^1(j), \delta_t^2(j)\}$ ▷ Get the most likely among two
    $\psi_{t,j}(i,d) = argmax_{(i \neq j,d)} \{\delta_t^1(j), \delta_t^2(j)\}$ ▷ Pointer to (i^*, d^*)
  end for
end for
for t = $D'+1$ upto T **do**
  Update $u_t(j,d)$
  for j = 1 upto N **do**
    $\delta_t^*(j) = max_{i \neq j} max_d \{u_t(j,d) p_j(d) a_{ij} \delta_{t-d}^*(i)\}$
    $\psi_{t,j}(i,d) = argmax_{(i \neq j,d)} \{u_t(j,d) p_j(d) a_{ij} \delta_{t-d}^*(i)\}$ ▷ Pointer to (i^*, d^*)
  end for
end for
Backtrack
$t = T$ ▷ Time index
$j^* = argmax_j \delta_T^*(j)$ ▷ Last state of most likely sequence
while t > 0 **do**
  $d^* = \psi_{t,j^*}(d)$ ▷ Get the duration of current state from pointer ψ
  $\hat{Q}_{t-d^*+1:t} = j^*$ ▷ State sequence from $t - d^* + 1$ upto t is equal to j^*
  $t = t - d^*$ ▷ Update time, move to the end of previous state
  $j^* = \psi_{t,j^*}(i)$ ▷ Move to preceeding state
end while

$$\mathbf{U} = \begin{bmatrix} u_t(1,1) & u_t(1,2) & \dots & u_t(1,D') \\ u_t(2,1) & u_t(2,2) & \dots & u_t(2,D') \\ \vdots & \vdots & \vdots & \vdots \\ u_t(N,1) & u_t(N,2) & \dots & u_t(N,D') \end{bmatrix} \tag{3.20}$$

To update the $u_t(j,d)$ values at the next time step, the columns of $\mathbf{U}$ are shifted and the first column is replaced with a vector consisting of ones. Here, the d^{th} column of $\mathbf{U}$ provides the probability of observing the output segment $O_{t-d+1:t-1}$ for each state, which is equal to $u_{t-1}(j,d-1)$.

$$\mathbf{U} = \begin{bmatrix} 1 & u_{t-1}(1,1) & \dots & u_{t-1}(1,D'-1) \\ 1 & u_{t-1}(2,1) & \dots & u_{t-1}(2,D'-1) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & u_{t-1}(N,1) & \dots & u_{t-1}(N,D'-1) \end{bmatrix} \tag{3.21}$$

Using the recursive relation given in (3.19), the $u_t(j,d)$ values can be determined from the matrix denoted in (3.21). This is done by multiplying the j^{th} row with the probability of observing the current observation, represented as $b_j(o_t)$:

$$\mathbf{U} = \begin{bmatrix} b_1(o_t) & u_{t-1}(1,1)b_1(o_t) & \dots & u_{t-1}(1,D'-1)b_1(o_t) \\ b_2(o_t) & u_{t-1}(2,1)b_2(o_t) & \dots & u_{t-1}(2,D'-1)b_2(o_t) \\ \vdots & \vdots & \vdots & \vdots \\ b_N(o_t) & u_{t-1}(N,1)b_N(o_t) & \dots & u_{t-1}(N,D'-1)b_N(o_t) \end{bmatrix}. \tag{3.22}$$

The Explicit-Duration HMM Viterbi Algorithm has a time complexity of $\mathcal{O}(T(N^2D' + ND'))$. This complexity, much like the traditional Viterbi algorithm, is mainly due to the recursion step which contributes the $\mathcal{O}(TN^2D')$ term. The recursive computation of u_t values reduces the complexity from $\mathcal{O}(TND'^2)$ to $\mathcal{O}(TND')$,

making it less complex compared to the naive method.

3.4.2 Forward-Backward Algorithm for Graph Smoothing

The Viterbi Algorithm returns the most likely state sequence from the observed sequence in the HSMM setting. On the other hand, the Forward-Backward Algorithm offers an alternative solution by identifying the most probable state at each individual time step. The significance of the Forward-Backward Algorithm extends beyond state estimation, as the Forward and Backward variables are crucial in learning the model parameters from the observed state sequences.

The Forward and Backward variables, as well as the recursive algorithms for computing them, can be defined for the general HSMMs as given in [34] and [30]. Nevertheless, just as with the Viterbi Algorithm, the assumptions of ED-HMM can be utilized to obtain simpler definitions and algorithms as discussed in [32]. In the context of the ED-HMM, the Forward variables are defined as follows:

$$\alpha_t(j) = P(Q_{t]} = S_j, O_{1:t}), \tag{3.23}$$

$$\alpha^*_{t+1}(j) = P(Q_{[t+1} = S_j, O_{1:t}). \tag{3.24}$$

The $\alpha_t(j)$ is defined as the probability of exiting state j at time t while observing $O_{1:t}$. $\alpha^*_{t+1}(j)$ is the probability of observing $O_{1:t}$ and transitioning to state j at time $t+1$. By definition, $\alpha^*_{t+1}(j)$ can be expressed in terms of $\alpha_t(i)$:

$$\alpha^*_{t+1}(j) = \sum_{i \neq j} P(Q_{[t+1} = S_j, Q_{t]} = S_i, O_{1:t}) = \sum_{i \neq j} \alpha_t(i) a_{ij}. \tag{3.25}$$

The ED-HMM Forward Algorithm, given in Algorithm 2, recursively calculates the $\alpha_t(j)$ values based on previous Forward variables and model parameters:

$$\alpha_t(j) = \sum_{d \in D} u_t(j,d) p_j(d) \alpha^*_{t-d+1}(j). \tag{3.26}$$

The $\alpha^*_t(j)$ values are also updated according to (3.25). The algorithm takes as input the model parameters and the observed sequence and returns the desired Forward variables. At each time step, the computation of $\alpha_t(j)$ for each state requires a summation over D' state durations, leading to a time complexity of $\mathcal{O}(ND')$. The computation of $\alpha^*_t(j)$ requires a summation over N states, resulting in a time complexity of $\mathcal{O}(N^2)$ per time step. The probabilities of observing each state ($u_t(j,d)$) are updated recursively, as in the Viterbi Algorithm, and this carries a time complexity of $\mathcal{O}(ND')$. As a result, the total asymptotic complexity of the ED-HMM Forward Algorithm is $\mathcal{O}(T(N^2 + ND'))$.

The Backward Algorithm for the ED-HMM is derived similarly, as discussed in [32]. The Backward variables are defined as follows:

$$\beta_t(j) = P(O_{t+1:T} | Q_{t]} = S_j), \tag{3.27}$$

$$\beta^*_{t+1}(j) = P(O_{t+1:T} | Q_{[t+1} = S_j). \tag{3.28}$$

Here, $\beta_t(j)$ is equal to the probability of observing $O_{t+1:T}$ given that state j is left at t, $\beta^*_{t+1}(j)$ is the conditional probability of observing the same partial output sequence ($O_{t+1:T}$) given a transition to state j occurs at $t+1$. $\beta_t(j)$ can be written in terms of $\beta^*_{t+1}(j)$ as follows:

Algorithm 2 Explicit-Duration HMM Forward Algorithm

Input : $\mathbf{A}, \mathbf{B}, \pi, p_j(d), \mathbf{O}$ ▷ Model parameters and output sequence
Output : $\alpha_t(j), \alpha_t^*(j)$ ▷ Forward variables
Initialization
for j = 1 upto N **do**
    $\alpha_1^*(j) = \pi_j$
end for
Recursion
for t = 1 upto D' **do**
    Update $u_t(j, d)$
    for j = 1 upto N **do**
        $\alpha_t(j) = \sum_{d \leq t} u_t(j, d) p_j(d) \alpha_{t-d+1}^*(j)$ ▷ Update $\alpha_t(j)$
    end for
    for j = 1 upto N **do**
        $\alpha_t^*(j) = \sum_{i \neq j} \alpha_t(i) a_{ij}$ ▷ Update $\alpha_t^*(j)$
    end for
end for
for t = $D' + 1$ upto T **do**
    Update $u_t(j, d)$
    for j = 1 upto N **do**
        $\alpha_t(j) = \sum_{d \in D} u_t(j, d) p_j(d) \alpha_{t-d+1}^*(j)$ ▷ Update $\alpha_t(j)$
    end for
    for j = 1 upto N **do**
        $\alpha_t^*(j) = \sum_{i \neq j} \alpha_t(i) a_{ij}$ ▷ Update $\alpha_t^*(j)$
    end for
end for

$$\beta_t(j) = \sum_{i \neq j} P(O_{t+1:T}, Q_{[t+1} = S_i | Q_{t]} = S_j) = \sum_{i \neq j} \beta^*_{t+1}(i) a_{ji}. \tag{3.29}$$

The $\beta^*_{t+1}(j)$ values are calculated recursively in the ED-HMM Backward Algorithm outlined in Algorithm 3 as follows:

$$\beta^*_{t+1}(j) = \sum_{d \in D} \beta_{t+d}(j) u_{t+d}(j, d) p_j(d). \tag{3.30}$$

The $\beta_t(j)$ values are also updated at each time step according to (3.29). The update of $\beta^*_t(j)$ values using (3.30) requires $\mathcal{O}(ND')$ computations at each time step. The recursive computation of $\beta_t(j)$ values using (3.29) takes $\mathcal{O}(N^2)$ time per time step. Additionally, updating the observation probabilities $u_{t+d}(j, d)$ also requires $\mathcal{O}(ND')$ operations. Consequently, the total time complexity of the ED-HMM Backward Algorithm is $\mathcal{O}(T(N^2 + ND'))$, which is identical to the complexity of the ED-HMM Forward Algorithm.

State sequence estimation using the Viterbi Algorithm is also referred to as the Maximum Likelihood (ML) Estimation in the literature. On the other hand, the alternative approach which employs the Forward and Backward variables to find the sequence consisting of the most probable states at each time step is referred to as the Maximum a Posteriori (MAP) Estimation. Originally proposed for the HMM setup in [37], this method is often called the BCJR algorithm. With simple adjustments, it can be applied to the HSMM framework as well.

The $\gamma_t(j)$ varible is defined to express the joint probability of occupying state j at time t while observing the given fixed sequence $O_{1:T}$ as follows:

$$\gamma_t(j) = P(Q_t = S_j, O_{1:T}). \tag{3.31}$$

Algorithm 3 Explicit-Duration HMM Backward Algorithm

Input : $\mathbf{A}, \mathbf{B}, p_j(d), \mathbf{O}$ ▷ Model parameters and output sequence
Output : $\beta_t(j), \beta_t^*(j)$ ▷ Backward variables
Initialization
for j = 1 upto N **do**
  $\beta_T(j) = 1$
  $u_T(j, 1) = b_j(O_T)$
end for
Recursion
for t = $T - 1$ downto $T - D' + 1$ **do**
  for j = 1 upto N **do**
    $\beta_{t+1}^*(j) = \sum_{d \leq T-t} \beta_{t+d}(j) u_{t+d}(j, d) p_j(d)$ ▷ Update β^*
  end for
  Update $u_{t+d}(j, d)$
  for j = 1 upto N **do**
    $\beta_t(j) = \sum_{i \neq j} \beta_{t+1}^*(i) a_{ji}$ ▷ Update β
  end for
end for
for t = $T - D'$ downto 1 **do**
  for j = 1 upto N **do**
    $\beta_{t+1}^*(j) = \sum_{d \in D} \beta_{t+d}(j) u_{t+d}(j, d) p_j(d)$
  end for
  Update $u_{t+d}(j, d)$
  for j = 1 upto N **do**
    $\beta_t(j) = \sum_{i \neq j} \beta_{t+1}^*(i) a_{ji}$
  end for
end for
for j = 1 upto N **do**
  $\beta_1^*(j) = \sum_{d \in D} \beta_{1+d}(j) u_{1+d}(j, d) p_j(d)$ ▷ Find β_1^*
end for

$\gamma_t(j)$ can be calculated recursively using Forward and Backward variables as described in [32] as follows:

$$\gamma_t(j) = \gamma_{t-1}(j) - \beta_{t-1}(j)\alpha_{t-1}(j) + \beta_t^*(j)\alpha_t^*(j). \tag{3.32}$$

This smoothing algorithm is named ED-HMM MAP Estimation Algorithm. It takes the Forward and Backward variables as inputs and returns the sequence of states that maximizes $\gamma_t(j)$ at each time step, along with the $\gamma_t(j)$ values themselves. Given that the Forward and Backward variables are already known, the calculation of $\gamma_t(j)$ values requires $\mathcal{O}(N)$ operations. Finding the argument (state) that maximizes $\gamma_t(j)$ has the same complexity. Consequently, the total time complexity of the algorithm is $\mathcal{O}(TN)$. A complete description of the algorithm is provided in Algorithm 4.

Algorithm 4 Explicit-Duration HMM MAP Estimation Algorithm

Input : $\pi, \alpha_t(j), \alpha_t^*(j), \beta_t(j), \beta_t^*(j)$ ▷ Forward and Backward variables
Output : $\hat{\mathbf{Q}}_{\mathbf{MAP}}, \gamma_t(j)$ ▷ Estimated sequence and $\gamma_t(j)$ values
Initialization
for j = 1 upto N **do**
  $\gamma_1(j) = \pi_j \beta_1^*(j)$ ▷ Initialize γ
  $\hat{Q}_{MAP}(1) = \arg\max_j\{\gamma_1(j)\}$ ▷ Most likely initial state
end for
Recursion
for t = 2 upto $T-1$ **do**
  for j = 1 upto N **do**
    $\gamma_t(j) = \gamma_{t-1}(j) - \beta_{t-1}(j)\alpha_{t-1}(j) + \beta_t^*(j)\alpha_t^*(j)$ ▷ Update γ
    $\hat{Q}_{MAP}(t) = \arg\max_j\{\gamma_t(j)\}$ ▷ Most likely state at t
  end for
end for
$\gamma_T(j) = \alpha_T(j)$
$\hat{Q}_{MAP}(T) = \arg\max_j\{\gamma_T(j)\}$ ▷ Most likely last state

3.4.3 Approximate Smoothing Algorithms for Graphs with Large State Space and Long Duration

In the proposed framework, each graph generated by the semantic extractor consists of multiple small atomic sub-graphs as demonstrated in Figure 2.3. The components within a sub-graph interact solely with the components within the same sub-graph. Following the goal-oriented filtering, only a limited number of these atomic sub-graphs are expected to survive, and distinct sub-graphs can be smoothed independently. This approach enables the parallel smoothing of multiple simple graphs with an additive complexity.

The ED-HMM Viterbi Algorithm is expected to work on these sub-graphs with small state spaces in parallel; hence, the algorithm is expected to be feasible regarding the time complexity. However, it is possible to have sub-graphs with large state spaces or long state durations in practice. In such cases, checking every state and duration pair as a possible predecessor at the recursion step, as stated in (3.16), might be infeasible due to computational constraints. This problem can be solved by substituting the smoothing algorithm with an algorithm with reduced time complexity that generates approximate results.

In the case of a graph with a large state space (N), complexity of the smoothing algorithm can be decreased by searching over only a subset of the possible states instead of all the states, similar to the M-Algorithm used for decoding HMMs [38],[39]. For graphs with long state durations (large D'), the recursion can be made over quantized state duration distributions, where the predecessor state is searched over a subset of the original duration set D instead of all D' values. Additionally, a third algorithm is proposed by combining these two approaches.

The first two algorithms which recursively search over a subset of the state space and a subset of the durations decrease the complexity of ED-HMM Viterbi algorithm

linearly. The parameters M and n are user-defined and determine the trade-off between the accuracy and the speed for the M-Algorithm and Quantized-Duration Search Algorithm, respectively. Recursive search of the original algorithm requires $\mathcal{O}(N^2D')$ operations, but this number reduces to $\mathcal{O}(NMD')$ for the M-Algorithm and $\mathcal{O}(N^2n)$ for the Quantized Duration Search Algorithm. For the third algorithm, complexity of the recursion step decreases quadratically to $\mathcal{O}(NMn)$ since it searches over subsets of both the state and the duration sets.

Although the proposed algorithms bring some additional complexity due to the intermediate steps required for sorting and quantization, they work faster compared to the original algorithm.

3.4.3.1 M-Algorithm for Smoothing in Explicit-Duration HMMs

The goal-filtered graph might have a large state space requiring excessive computational power for smoothing the observed state sequence. In such cases, the most likely state sequence explaining the observed sequence can be estimated from a subset of all possible state sequences to reduce the complexity of the algorithm. A heuristic approach is to search for the path over the most likely M states at the previous time steps instead of all N states; this idea is identical to the M-Algorithm in the HMM setting [38],[39].

ED-HMM Viterbi is modified for this new algorithm by searching over the most likely M states (S'_t) at previous time steps. S'_t is a subset of the original state space S, it contains the states with M largest δ^*_t values at each time step as defined in (3.34). The recursion in (3.16) is updated such that maximization is performed over the set S'_t instead of S:

$$\delta_t^*(j) = \max_{(i \in S'_{t-d} \setminus \{j\}, d \in D)} u_t(j,d) a_{ij} p_j(d) \delta_{t-d}^*(i). \tag{3.33}$$

$$S'_t = \underset{S'_t \subset S : |S'_t| = M}{\arg\max} \; \delta_t^*(j) \tag{3.34}$$

The new algorithm is called as the ED-HMM M-Algorithm and its complete description is given in Algorithm 5. Figure 3.4 illustrates a simple example for its recursion step.

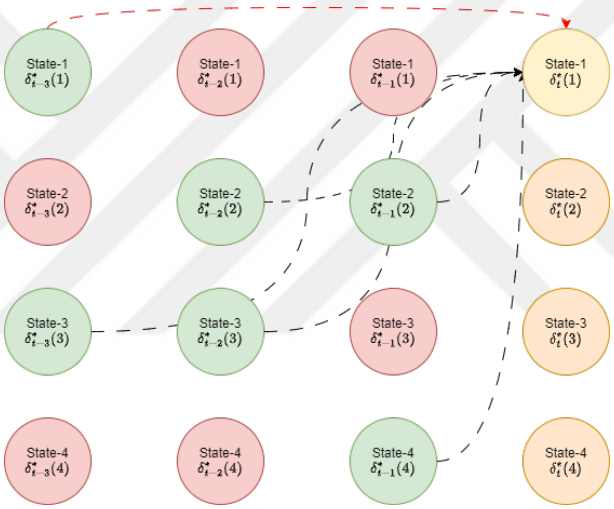


Figure 3.4: M-Algorithm for ED-HMM with four states. D' is equal to three in this example and M value is chosen as two for the Explicit-Duration HMM M-Algorithm, therefore only the transitions from the two states with largest δ_t^* values at the last three steps is considered. To calculate $\delta_t^*(1)$, the two most likely states at time steps $t-3$, $t-2$, and $t-1$ are taken into account, which are $S'_{t-3} = \{S_1, S_3\}$, $S'_{t-2} = \{S_2, S_3\}$, and $S'_{t-1} = \{S_2, S_4\}$, respectively. Since the self-transition probability is zero, transition from State-1 at time $t-3$ is ignored.

In the ED-HMM Viterbi Algorithm, the complexity is dominated by the recursion step. By using the M-Algorithm, the complexity of this step is decreased from $\mathcal{O}(N^2 D')$ to $\mathcal{O}(NMD')$. However, the M-Algorithm introduces an additional step to identify the M most likely states, S'_t, at each time instance.

Algorithm 5 Explicit-Duration HMM M-Algorithm

Input : $\mathbf{A}, \mathbf{B}, \pi, p_j(d), \mathbf{O}, M$ ▷ Model, output sequence and M
Output : $\hat{\mathbf{Q}}$ ▷ Most likely state sequence
Initialization
$u_1(j,1) = b_j(O_1)$ ▷ Initialize $u_t(j,d)$
for j = 1 upto N **do**
  $\delta_1^*(j) = \pi_j p_j(1) u_1(j,1)$
  $\psi_{1,j}(i,d) = (-1,1)$ ▷ Pointer to previous state and current duration
end for
Get $S_1' \subset S,\ |S_1'| = M$ ▷ Find the M states with largest $\delta_1^*(j)$ values
Recursion
for t = 2 upto D' **do**
  Update $u_t(j,d)$
  for j = 1 upto N **do**
    $\delta_t^1(j) = \pi_j p_j(d) u_t(j,t)$ ▷ Occupying state j from start
    $\delta_t^2(j) = max_{(i \in S_{t-d}', d<t)}\{u_t(j,d) p_j(d) a_{ij} \delta_{t-d}^*(i)\}$ ▷ Moving to state j
    $\delta_t^*(j) = \max\{\delta_t^1(j), \delta_t^2(j)\}$ ▷ Get the most likely among two
    $\psi_{t,j}(i,d) = argmax_{(i \in S_{t-d}' \setminus \{j\}, d)}\{\delta_t^1(j), \delta_t^2(j)\}$ ▷ Pointer to (i^*, d^*)
  end for
  Get $S_t' \subset S,\ |S_t'| = M$ ▷ Find the M states with largest $\delta_t^*(j)$ values
end for
for t = $D' + 1$ upto T **do**
  Update $u_t(j,d)$
  for j = 1 upto N **do**
    $\delta_t^*(j) = max_{(i \in S_{t-d}', d)}\{u_t(j,d) p_j(d) a_{ij} \delta_{t-d}^*(i)\}$
    $\psi_{t,j}(i,d) = argmax_{(i \in S_{t-d}' \setminus \{j\}, d)}\{u_t(j,d) p_j(d) a_{ij} \delta_{t-d}^*(i)\}$ ▷ Pointer
  end for
  Get $S_t' \subset S,\ |S_t'| = M$ ▷ Find the M states with largest $\delta_t^*(j)$ values
end for
Backtrack
$t = T$ ▷ Time index
$j^* = argmax_j \delta_T^*(j)$ ▷ Last state of most likely sequence
while t > 0 **do**
  $d^* = \psi_{t,j^*}(d)$ ▷ Get the duration of current state from pointer ψ
  $\hat{Q}_{t-d^*+1:t} = j^*$ ▷ State sequence from $t - d^* + 1$ upto t is equal to j^*
  $t = t - d^*$ ▷ Update time, move to the end of previous state
  $j^* = \psi_{t,j^*}(i)$ ▷ Move to preceding state
end while

At each time step, S'_t can be determined by sorting the δ^*_t values, which requires $\mathcal{O}(Nlog(N))$ operations. Alternatively, S'_t can be identified by monitoring the largest M values during the recursion, this approach is faster than sorting when M is smaller than $logN$. For every state, its δ^*_t can be compared with the current largest M values in $\mathcal{O}(logM)$ steps. If the calculated δ^*_t value is larger than any of the current M values, the smaller values in the array need to be shifted below with a computational complexity of $\mathcal{O}(M)$. Therefore, this method brings a total complexity of $\mathcal{O}(N(M + logM))$. With the assumption that $logN$ is smaller than M, the M-Algorithm is implemented using the sorting approach. This results in the M-Algorithm having an additional complexity of $\mathcal{O}(Nlog(N))$.

The computation of the u_t values in the M-Algorithm is the same as in the original ED-HMM Viterbi algorithm, resulting in a complexity of $\mathcal{O}(TND')$. Therefore, the total time complexity of the ED-HMM M-Algorithm is $\mathcal{O}(T(NMD' + ND' + Nlog(N)))$. Compared to the complexity of the original algorithm, which is $\mathcal{O}(T(N^2D' + ND'))$, the M-Algorithm exhibits a significantly lower complexity for small values of M/N.

3.4.3.2 Quantized-Duration Search for Semantic Graph Smoothing

Smoothing of graphs with long waiting durations, high D' values, might be infeasible, similar to those with large state spaces, especially under computational constraints. A reduced time complexity smoothing algorithm for such graphs can be developed following a similar approach to the M-Algorithm. Rather than searching for the most likely sequence across all state and duration pairs, the recursion step can focus on a subset of the state waiting time durations (D_q).

Choosing the new waiting time durations carefully is critical to avoid a significant

decrease in the accuracy of the smoothing algorithm while decreasing its complexity. A heuristic solution is to quantize the waiting times such that duration values with higher probabilities are represented in finer detail and long intervals with small probability are quantized more heavily. This approach is expected to work well with relatively smooth distributions.

As in the M-Algorithm, the cumulative distribution function ($F_p(d)$) of the waiting time distribution, $p_j(d)$, is partitioned into intervals with equal probability of $1/n$, where n determines the number of quantization levels. Boundaries (x_i) are chosen to construct intervals of probability $\Delta p = 1/n$:

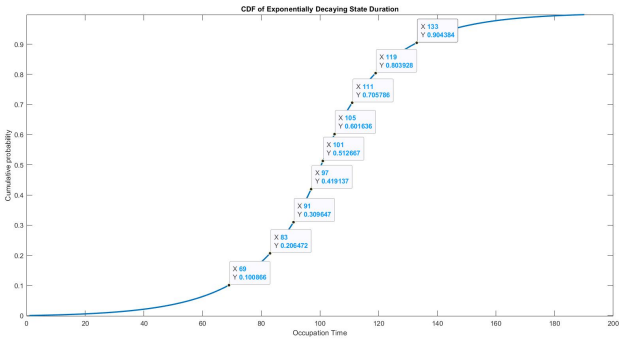
$$x_i = \min\{d : F_p(d) \geq i\Delta p\},\ i \in \{1, 2, \ldots, n\},\ d \in D. \tag{3.35}$$

The average of the boundary values of an interval is used to represent the values in that interval:

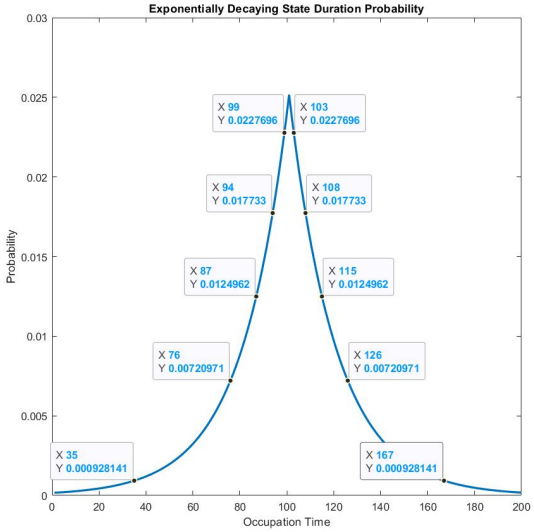
$$d_i = \frac{x_{i-1} + x_i}{2},\ i \in \{1, 2, \ldots, n\}. \tag{3.36}$$

An illustration of this quantization method for an exponentially decaying distribution is shown in Figure 3.5. The boundaries of the intervals, x_i, are indicated on the cumulative distribution function plot, and the representative points (d_i) are marked on the distribution. As this distribution is exponentially decaying, the more likely values around the peak of the distribution are represented in finer detail, while the less likely tail values are quantized coarsely.

The new algorithm differs from the ED-HMM Viterbi due to its recursion step. Instead of searching for the predecessor state over the previous D' time steps, it takes into account the quantized values D_q, which are used to represent n intervals.



(a) Boundary points



(b) Representer values

Figure 3.5: Quantization of an exponentially decaying (Laplacian) distribution centered at 101. The CDF is partitioned into ten intervals of equal probability. The boundary points are further away at the edges as the probability decreases and they are closer around the middle, this enables to quantize the more likely values finer and less likely values in a more coarse way.

The recursion for this new algorithm, which is called as the Quantized-Duration ED-HMM Viterbi Algorithm, can be expressed as follows:

$$\delta_t^*(j) = \max_{(i \in S \setminus \{j\}, d \in D_q = \{d_1, d_2, \ldots, d_n\})} u_t(j,d) a_{ij} \Delta p \delta_{t-d}^*(i). \tag{3.37}$$

This approach reduces the recursion complexity from $\mathcal{O}(N^2 D')$ to $\mathcal{O}(N^2 n)$, resulting in a linear improvement similar to the M-Algorithm. Since quantization of waiting time distributions is only required once at the beginning with a complexity of $\mathcal{O}(ND')$, it does not affect the total asymptotic complexity. Therefore, with the other steps remaining unmodified, the total complexity of the new algorithm becomes $\mathcal{O}(T(N^2 n + ND'))$. The complete algorithm is provided in Algorithm 6.

3.4.3.3 The Quantized-Duration M-Algorithm for Graph Smoothing

The algorithms with reduced time complexity discussed above are suitable for smoothing graphs with large state spaces (large N) or extended waiting time durations (large D'). In practical applications, it may be necessary to work with graphs that have both large state spaces and long state durations. In such scenarios, an algorithm combining the two previous algorithms can be employed. The recursive step of the resulting algorithm searches the preceding state over the subset of states (S'_t) and quantized time steps (D_q):

$$\delta_t^*(j) = \max_{(i \in S'_{t-d} \setminus \{j\},\ d \in D_q = \{d_1, d_2, \ldots, d_n\})} u_t(j,d) a_{ij} \Delta p \delta_{t-d}^*(i). \tag{3.38}$$

This approach results in a quadratic drop in recursion complexity from $\mathcal{O}(N^2 D')$ to $\mathcal{O}(NMn)$. Quantization complexity is negligible since it occurs once and requires $\mathcal{O}(ND')$ operations. The most probable path for each state is searched over the M

Algorithm 6 Quantized-Duration Explicit Duration HMM Viterbi Algorithm

Input : $\mathbf{A}, \mathbf{B}, \pi, p_j(d), \mathbf{O}, n$ ▷ Model parameters, output sequence and n
Output : $\hat{\mathbf{Q}}$ ▷ Most likely state sequence
Obtain quantized durations $D_q = \{d_1, \ldots, d_n\}$ for each distribution $p_j(d)$
Initialization
$u_1(j,1) = b_j(O_1)$ ▷ Initialize $u_t(j,d)$
for j = 1 upto N **do**
  $\delta_1^*(j) = \pi_j p_j(1) u_1(j,1)$
  $\psi_{1,j}(i,d) = (-1,1)$ ▷ Pointer to previous state and current duration
end for
Recursion
for t = 2 upto D' **do** ▷ Filter the first D' steps completely
  Update $u_t(j,d)$
  for j = 1 upto N **do**
    $\delta_t^1(j) = \pi_j p_j(d) u_t(j,t)$ ▷ Occupying state j from start
    $\delta_t^2(j) = max_{i \neq j} max_{d<t} \{\delta_{t-d}^*(i) p_j(d) u_t(j,d) a_{ij}\}$ ▷ Moving to state j
    $\delta_t^*(j) = \max\{\delta_t^1(j), \delta_t^2(j)\}$ ▷ Get the most likely among two
    $\psi_{t,j}(i,d) = argmax_{(i,d)} \{\delta_t^1(j), \delta_t^2(j)\}$ ▷ Pointer to (i^*, d^*)
  end for
end for
for t = $D'+1$ upto $T - D'$ **do** ▷ Filter the middle part approximately
  Update $u_t(j,d)$
  for j = 1 upto N **do**
    $\delta_t^*(j) = max_{(i \neq j, d \in D_q)} \{\delta_{t-d}^*(i) \Delta p u_t(j,d) a_{ij}\}$ ▷ Use D_q instead of D
    $\psi_{t,j}(i,d) = argmax_{(i,d)} \{\delta_{t-d}^*(i) \Delta p u_t(j,d) a_{ij}\}$ ▷ Pointer to (i^*, d^*)
  end for
end for
for t = $T - D' + 1$ upto T **do** ▷ Filter last D' steps completely
  Update $u_t(j,d)$
  for j = 1 upto N **do**
    $\delta_t^*(j) = max_{(i \neq j, d \in D)} \{\delta_{t-d}^*(i) p_j(d) u_t(j,d) a_{ij}\}$
    $\psi_{t,j}(i,d) = argmax_{(i,d)} \{\delta_{t-d}^*(i) p_j(d) u_t(j,d) a_{ij}\}$ ▷ Pointer to (i^*, d^*)
  end for
end for
Backtrack
$t = T$ ▷ Time index
$j^* = argmax_j \delta_t^*(j)$ ▷ Last state of most likely path
while t > 0 **do**
  $d^* = \psi_{t,j^*}(d)$ ▷ Get the duration of current state from pointer ψ
  $\hat{Q}_{t-d^*+1:t} = j^*$ ▷ State sequence from $t - d^* + 1$ upto t is equal to j^*
  $t = t - d^*$ ▷ Update time, move to the end of previous state
  $j^* = \psi_{t,j^*}(i)$ ▷ Move to preceeding state
end while

most likely states at the previous time steps, necessitating sorting of δ_t^* values with a complexity of $\mathcal{O}(N \log N)$ at each time step. Updating the observation probabilities, $u_t(j, d)$, takes $\mathcal{O}(ND')$ operations, resulting in a total asymptotic complexity of $\mathcal{O}(T(NMn + ND' + N \log N))$. The resulting algorithm is presented in Algorithm 7.

3.4.4 Multi-Sensor Data Fusion for Semantic-Graphs

A more accurate understanding of a semantic signal can be obtained by utilizing semantic information generated from multiple correlated sensors, as opposed to relying on the information extracted by a single sensor. This can be achieved by using smoothed graphs that have been extracted by different sensors, with each graph depending on the unique observation sequence and statistical model of the respective sensor. However, within the proposed framework, a better approach would be to leverage the correlation between the extracted graphs and the statistical models of different sensors during the smoothing process. This allows the generation of a single smoothed graph sequence that provides the best fit to all sequences extracted by distinct sensors observing a common area of interest.

Correlation in the semantic information extracted by different sensors can arise due to various factors. For instance, such a correlation could occur when two cameras with overlapping field-of-views are used as sensors, or when different sensors observe the exact same scene, thus producing multiple semantic-graphs from the same region of interest. In the latter case, a modified version of the ED-HMM Viterbi Algorithm can be employed for the fusion of semantic information. This new algorithm called Multi-Sensor Viterbi Algorithm is designed to estimate the underlying graph sequence using multiple observed graph sequences generated by different sensors that are observing the same scene, and hence producing graph sequences based on the same underlying semantic information. Figure 3.6 illustrates a scenario where two

Algorithm 7 Explicit-Duration HMM Quantized M-Algorithm

Input : $\mathbf{A}, \mathbf{B}, \pi, p_j(d), \mathbf{O}, M, n$ ▷ Model, output sequence, M and n
Output : $\hat{\mathbf{Q}}$ ▷ Most likely state sequence
Obtain quantized durations $D_q = \{d_1, \ldots, d_n\}$ for each distribution $p_j(d)$
Initialization
$u_1(j,1) = b_j(O_1)$ ▷ Initialize $u_t(j,d)$
for j = 1 upto N **do**
  $\delta_1^*(j) = \pi_j p_j(1) u_1(j,1)$
  $\psi_{1,j}(i,d) = (-1,1)$ ▷ Pointer to previous state and current duration
end for
Get $S_1' \subset S,\ |S_1'| = M$ ▷ Find the M states with largest $\delta_1^*(j)$ values
Recursion
for t = 2 upto D' **do** ▷ Filter the first D' steps without quantization
  Update $u_t(j,d)$
  for j = 1 upto N **do**
    $\delta_t^1(j) = \pi_j p_j(d) u_t(j,t)$ ▷ Occupying state j from start
    $\delta_t^2(j) = max_{i \in S_{t-d}'} max_{d<t} \{\delta_{t-d}^*(i) p_j(d) u_t(j,d) a_{ij}\}$ ▷ Moving to state j
    $\delta_t^*(j) = \max\{\delta_t^1(j), \delta_t^2(j)\}$ ▷ Get the most likely among two
    $\psi_{t,j}(i,d) = argmax_{(i,d)} \{\delta_t^1(j), \delta_t^2(j)\}$ ▷ Pointer to (i^*, d^*)
  end for
  Get $S_t' \subset S,\ |S_t'| = M$ ▷ Find the M states with largest $\delta_t^*(j)$ values
end for
for t = $D' + 1$ upto $T - D'$ **do** ▷ Filter the middle with quantization
  Update $u_t(j,d)$
  for j = 1 upto N **do**
    $\delta_t^*(j) = max_{(i \in S_{t-d}', d \in D_q)} \{\delta_{t-d}^*(i) \Delta p u_t(j,d) a_{ij}\}$
    $\psi_{t,j}(i,d) = argmax_{(i,d)} \{\delta_{t-d}^*(i) \Delta p u_t(j,d) a_{ij}\}$ ▷ Pointer to (i^*, d^*)
  end for
  Get $S_t' \subset S,\ |S_t'| = M$ ▷ Find the M states with largest $\delta_t^*(j)$ values
end for
for t = $T - D' + 1$ upto T **do** ▷ Filter the last D' without quantization
  Update $u_t(j,d)$
  for j = 1 upto N **do**
    $\delta_t^*(j) = max_{(i \in S_{t-d}', d \in D)} \{\delta_{t-d}^*(i) p_j(d) u_t(j,d) a_{ij}\}$
    $\psi_{t,j}(i,d) = argmax_{(i,d)} \{\delta_{t-d}^*(i) p_j(d) u_t(j,d) a_{ij}\}$ ▷ Pointer to (i^*, d^*)
  end for
  Get $S_t' \subset S,\ |S_t'| = M$ ▷ Find the M states with largest $\delta_t^*(j)$ values
end for
Backtrack

cameras observe the same scene, offering complementary information to each other.

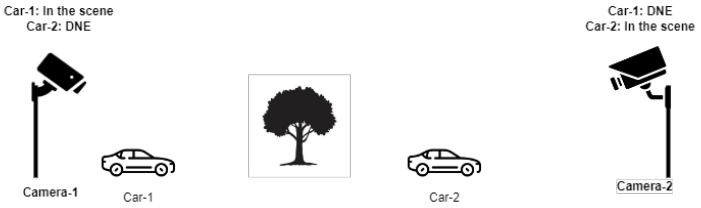


Figure 3.6: Multi-camera setup for the sensor fusion problem. The two cameras are placed at the opposite ends of a road. A tree blocks their view, preventing an accurate view of half the road. In this scene, there are two cars present, yet each camera can only detect one of them independently. By fusing the extracted graph sequences from both cameras to estimate the underlying semantic graph, instead of relying on two individually smoothed graphs that fail to leverage the correlation between the extracted sequences, a more accurate representation of the scene can be obtained.

Multi-Sensor Viterbi Algorithm incorporates observations from two sensors to infer the underlying graph (state) sequence. It is important to note that although the two sensors share identical model parameters, the observation kernels differ which leads to the generation of two different observation sequences from the same underlying graph sequence. This algorithm can be extended to cases with more than two sensors easily.

The mathematical objective of this algorithm is to estimate the state sequence (Q) that is the most likely, given the observed output sequences ($O^{(1)}, O^{(2)}$):

$$\hat{Q} = \arg\max_{Q_{1:T}} P(Q_{1:T}|O^{(1)}_{1:T}, O^{(2)}_{1:T}) = \frac{P(Q_{1:T}, O^{(1)}_{1:T}, O^{(2)}_{1:T})}{P(O^{(1)}_{1:T}, O^{(2)}_{1:T})}, \tag{3.39}$$

$$\hat{Q} = \arg\max_{Q_{1:T}} P(Q_{1:T}, O^{(1)}_{1:T}, O^{(2)}_{1:T}). \tag{3.40}$$

The definition of the intermediate variable, as presented in (3.8), is slightly modified to account for multiple observation sequences:

$$\delta_t^*(j) = \max_{Q_{1:t-1}} P(Q_{t]} = s_j, Q_{1:t-1}, O_{1:T}^{(1)}, O_{1:T}^{(2)}). \tag{3.41}$$

The two sensors in our model are assumed to share identical parameters, except for their respective observation kernels, represented as $b_j^{(1)}(o_t^{(1)})$ for the first sensor and $b_j^{(2)}(o_t^{(2)})$ for the second sensor. The probabilities corresponding to the output segments between $t-d+1$ and t are expressed as $u_t^{(1)}(j,d)$ and $u_t^{(2)}(j,d)$ for the first and second sensor respectively. The recursive expression can be derived in a manner similar to the steps followed in the derivation of the ED-HMM Viterbi Algorithm, as presented in (3.13) - (3.15). The new recursive expression incorporating these multiple observation sequences is given below:

$$\delta_t^*(j) = \max_{s_i \in S \setminus \{s_j\}} \max_{d \in D} \; u_t^{(1)}(j,d) u_t^{(2)}(j,d) a_{ij} p_j(d) \delta_{t-d}^*(i). \tag{3.42}$$

This algorithm can be easily extended to fuse information coming from more than two sensors. The only required modification in the recursive expression, as shown in (3.42), is the inclusion of the observation from the k^{th} sensor via the multiplicative term $u_t^{(k)}(j,d)$. For finding the most likely state sequence, pointers for the state-duration pairs are stored. The complete algorithm, with two observation sequences, is described in Algorithm 8.

The presented algorithm shares the same computational steps as the Viterbi Algorithm, with the key difference being in the update of the output sequences. Specifically, updating $u_t(j,d)$ requires $\mathcal{O}(ND')$ operations when considering a single output sequence. But when there are K output sequences, this update requires $\mathcal{O}(KND')$ operations at each time step. Given that the maximization step has a complexity of $\mathcal{O}(ND')$ and is executed for each state, the total time complexity of the algorithm becomes $\mathcal{O}(T(KND' + N^2D'))$.

Algorithm 8 Multi-Sensor Viterbi Algorithm

Input : $\mathbf{A}, \mathbf{B^{(1)}}, \mathbf{B^{(2)}}, \pi, p_j(d), \mathbf{O^{(1)}}, \mathbf{O^{(2)}}$ ▷ Models and output sequences
Output : $\hat{\mathbf{Q}}$ ▷ Most likely state sequence
Initialization
$u_1^{(1)}(j,1) = b_j^{(1)}(O_1^{(1)})$ ▷ Initialize $u_t^{(1)}(j,d)$
$u_1^{(2)}(j,1) = b_j^{(2)}(O_1^{(2)})$ ▷ Initialize $u_t^{(2)}(j,d)$
for j = 1 upto N **do**
    $\delta_1^*(j) = \pi_j p_j(1) u_1^{(1)}(j,1) u_1^{(2)}(j,1)$
    $\psi_{1,j}(i,d) = (-1,1)$ ▷ Pointer to previous state and current duration
end for
Recursion
for t = 2 upto D' **do**
    Update $u_t^{(1)}(j,d)$
    Update $u_t^{(2)}(j,d)$
    for j = 1 upto N **do**
        $\delta_t^1(j) = \pi_j p_j(t) u_t^{(1)}(j,t) u_t^{(2)}(j,t)$ ▷ Occupying state j from start
        $\delta_t^2(j) = max_{i \neq j} max_{d<t} \{u_t^{(1)}(j,d) u_t^{(2)}(j,d) p_j(d) a_{ij} \delta_{t-d}^*(i)\}$
        $\delta_t^*(j) = \max\{\delta_t^1(j), \delta_t^2(j)\}$ ▷ Get the most likely among two
        $\psi_{t,j}(i,d) = argmax_{(i \neq j,d)} \{\delta_t^1(j), \delta_t^2(j)\}$ ▷ Pointer to (i^*, d^*)
    end for
end for
for t = $D'+1$ upto T **do**
    Update $u_t^{(1)}(j,d)$
    Update $u_t^{(2)}(j,d)$
    for j = 1 upto N **do**
        $\delta_t^*(j) = max_{i \neq j} max_d \{u_t^{(1)}(j,d) u_t^{(2)}(j,d) p_j(d) a_{ij} \delta_{t-d}^*(i)\}$
        $\psi_{t,j}(i,d) = argmax_{(i \neq j,d)} \{u_t^{(1)}(j,d) u_t^{(2)}(j,d) p_j(d) a_{ij} \delta_{t-d}^*(i)\}$ ▷ Pointer
    end for
end for
Backtrack
$t = T$ ▷ Time index
$j^* = argmax_j \delta_T^*(j)$ ▷ Last state of most likely sequence
while t > 0 **do**
    $d^* = \psi_{t,j^*}(d)$ ▷ Get the duration of current state from pointer ψ
    $\hat{Q}_{t-d^*+1:t} = j^*$ ▷ State sequence from $t - d^* + 1$ up to t is equal to j^*
    $t = t - d^*$ ▷ Update time, move to the end of previous state
    $j^* = \psi_{t,j^*}(i)$ ▷ Move to preceding state
end while

3.4.5 Learning Model Parameters from Observed Data

The algorithms discussed so far require knowledge of the model parameters. These parameters, as explained in the earlier discussions on the state space representation relating to graph configurations, correspond to the probabilities of state transitions, state waiting times, and observation probabilities.

However, when a sensor is set up in real-world scenarios, these statistics are typically not available. Therefore, they must be learned from the observed sequences. For instance, consider a scenario where a camera is installed to observe a hallway. The graph language utilized to interpret this scene consists of person component nodes and three predicate nodes corresponding to standing, walking, and running. While these three states (graphs) can be extracted from the video stream, initially there is not any knowledge regarding the transition probabilities between different states, likelihood of the walking state being confused as the running state, or the duration distribution of the standing state. The only information available is the observed sequence, which possibly contains errors, and perhaps an initial guess for the model parameters.

In the context of the ED-HMM setting, the problem is to identify the model parameters that best explain the observed sequence. This problem is well-known and has been extensively studied in the context of HMMs and HSMMs. The most common approach for model estimation is the ML estimation, which aims to find the parameters that best explain the observed sequence by maximizing the likelihood. However, due to the presence of hidden states (variables), explicitly determining the parameters that maximize the likelihood is not possible.

In the case of HMMs, a viable approach to obtain estimates for the model parameters is the Baum-Welch Algorithm [40], which is an Expectation-Maximization (EM) algorithm that iteratively learns the parameters. The algorithm consists of

two major steps: the E-step and the M-step. In the E-step, the lower bound on the likelihood is computed given the current parameter estimates. This step involves estimating the latent variables or missing data using the current set of parameters. In the M-step, the parameters are updated to maximize the lower bound or, equivalently, to maximize the expected log-likelihood. These two steps are iterated until a convergence or stopping criteria is met. With slight modifications, the Baum-Welch Algorithm can also be applied to learn HSMMs [41], [32].

The Baum-Welch Algorithm can be applied to learn general HSMMs [41]. However, for ED-HMMs, the derivation of the Baum-Welch Algorithm becomes simpler as discussed in [32]. The Baum-Welch Algorithm requires to define three intermediate variables, which are needed for re-estimating the model parameters. These variables can be expressed in terms of the current model parameters and Forward-Backward variables defined in (3.23), (3.24) and (3.27), (3.28).

The first variable required for the Baum-Welch Algorithm is $\gamma_t(j)$, it is defined in (3.31). The second intermediate variable is $\eta_t(i,d)$, which is defined as the probability of occupying state i for d time steps before leaving it at t while observing the sequence $O_{1:T}$:

$$\eta_t(i,d) = P(Q_{[t-d+1:t]} = S_i, O_{1:T}). \tag{3.43}$$

Where $\eta_t(i,d)$ can be expressed in terms of Forward-Backward variables and model parameters as follows:

$$\eta_t(i,d) = \alpha^*_{t-d+1}(i) p_i(d) u_t(i,d) \beta_t(i). \tag{3.44}$$

The last variable required for the derivation of Baum-Welch Algorithm is $\xi_t(i,j)$

which is the probability of state transition from state i to state j at time instance t while observing the output sequence $O_{1:T}$:

$$\xi_t(i,j) = P(Q_{t]} = S_i, Q_{[t+1} = S_j, O_{1:T}). \tag{3.45}$$

$\xi_t(i,j)$ can also be expressed in terms of the model parameters and the Forward-Backward variables as:

$$\xi_t(i,j) = \alpha_t(i) a_{ij} \beta^*_{t+1}(j). \tag{3.46}$$

In the M-step of the Baum-Welch Algorithm, the model parameters are re-estimated using the intermediate variables. The model parameters consist of the state transition matrix $\mathbf{A}$, observation matrix $\mathbf{B}$, initial state probabilities π, and state duration distribution $p_j(d)$. The updates for the model parameters can be performed as follows:

$$a_{ij} = \frac{\sum_t \xi_t(i,j)}{\sum_j \sum_t \xi_t(i,j)}, \tag{3.47}$$

$$b_i(j) = \frac{\sum_{t \ s.t. \ (o_t = S_j)} \gamma_t(i)}{\sum_t \gamma_t(i)}, \tag{3.48}$$

$$p_i(d) = \frac{\sum_t \eta_t(i,d)}{\sum_d \sum_t \eta_t(i,d)}, \tag{3.49}$$

$$\pi_i = \frac{\gamma_1(i)}{\sum_j \gamma_1(j)}. \tag{3.50}$$

The complete description of the Baum-Welch Algorithm for learning ED-HMMs is provided in Algorithm 9. The algorithm iteratively computes the intermediate variables and performs parameter estimation until a convergence criterion is met. Like

many EM algorithms, the Baum-Welch Algorithm is guaranteed to converge to a locally optimal solution, so the initialization of the parameters is important. To further improve the results obtained, the algorithm can be started with different initializations to find the best model.

In cases where the model parameters follow a parametric distribution, the M-step can be adjusted to find the parameters that maximize the likelihood [41]. The formulas detailed in (3.47) - (3.50) do not impose any specific structure on the distributions (pmfs) they estimate. Each row of both the transition and observation matrices is treated as non-parametric probability vectors, which consists of non-zero elements that sum up to one. Similarly, the waiting time distributions, $p_j(d)$, for each state are estimated as a pmf over set D, in a non-parametric fashion. This approach necessitates a long observation sequence to accurately estimate the model parameters, especially for the waiting time distributions. It is essential that each state-duration pairing is observed often enough to reflect their relative frequencies. However, if $p_j(d)$ follows a parametric distribution, or can be approximated well by one, then these parameters can be estimated. This reduces the number of variables that need to be learned from the observations.

3.5 Chapter Summary

This chapter begins by exploring the state-space representation for semantic graphs, followed by the introduction of essential definitions and mathematical notation needed for HSMM formulation. It then delves into graph-smoothing algorithms, particularly the Viterbi and Forward-Backward Algorithms, which aim to correct errors in the extracted graph sequences by utilizing prior knowledge and exploiting temporal continuity. Reduced-complexity variants of the Viterbi-based smoothing

Algorithm 9 Explicit-Duration HMM Baum-Welch Algorithm

Input : $\mathbf{A^{(0)}}, \mathbf{B^{(0)}}, \pi^{(0)}, p_j(d)^{(0)}, \mathbf{O}$ ▷ Initial model and output sequence
Output : $\mathbf{A}, \mathbf{B}, \pi, p_j(d)$ ▷ Estimated model parameters
while Convergence criteria is not satisfied **do**
  Find Forward – Backward variables with current model
  Calculate α *and* α^* ▷ Call Algorithm 2 with $A^{(k)}, B^{(k)}, \pi^{(k)}, p_j(d)^{(k)}$
  Calculate β *and* β^* ▷ Call Algorithm 3 with $A^{(k)}, B^{(k)}, \pi^{(k)}, p_j(d)^{(k)}$
  Get intermediate variables
  Calculate γ ▷ Call Algorithm 4
  for t = 1 upto T **do**
    Update $u_t(j,d)$
    for i = 1 upto N **do**
      for d = 1 upto D **do**
        $\eta_t(i,d) = \alpha^*_{t-d+1}(i) p_i(d) u_t(i,d) \beta_t(i)$ ▷ Get $\eta_t(i,d)$
      end for
    end for
  end for
  for t = 1 upto T-1 **do**
    for i = 1 upto N **do**
      for j = 1 upto N **do**
        $\xi_t(i,j) = \alpha_t(i) a_{ij} \beta^*_{t+1}(j)$ ▷ Get $\xi_t(i,j))$
      end for
    end for
  end for
  Update model parametes ▷ Find $A^{(k+1)}, B^{(k+1)}, \pi^{(k+1)}, p_j(d)^{(k+1)}$
  for i = 1 upto N **do**
    for j = 1 upto N **do**
      $a_{ij}^{(k+1)} = \frac{\sum_t \xi_t(i,j)}{\sum_j \sum_t \xi_t(i,j)}$ ▷ Update A
    end for
  end for
  for i = 1 upto N **do**
    $\pi_i^{(k+1)} = \frac{\gamma_1(i)}{\sum_j \gamma_1(j)}$ ▷ Update π
  end for
  for i = 1 upto N **do**
    for j = 1 upto N **do**
      $b_i^{(k+1)}(j) = \frac{\sum_{t \ s.t. \ (o_t = S_j)} \gamma_t(i)}{\sum_t \gamma_t(i)}$ ▷ Update B
    end for
  end for
  for i = 1 upto N **do**
    for d = 1 upto D **do**
      $p_i^{(k+1)}(d) = \frac{\sum_t \eta_t(i,d)}{\sum_d \sum_t \eta_t(i,d)}$ ▷ Update $p_i(d)$
    end for
  end for
end while

algorithm are discussed to address challenges related to large state spaces and extended state durations. Specifically, algorithms are proposed that search for the most likely path over a subset of states in cases of large state spaces, and that use quantized state distributions for extended state durations. The chapter further proposes an algorithm for fusing semantic information from different sensors observing the same scene. It concludes with a discussion on the Baum-Welch Algorithm for model learning.

Chapter 4

Empirical Validation of HSMM Algorithms: CARLA Experiments and MATLAB Simulations

In this chapter, results from MATLAB simulations and video streams produced in the controlled CARLA environment are presented. These simulations and experiments involve producing graph sequences, based on a Semi-Markov Process, with noisy observation sequences. This chapter also includes results for the reduced time complexity smoothing algorithms, the semantic information fusion algorithm, and the learning algorithm introduced in the preceding chapter. The results indicate the efficacy of the HSMM-based approach for improving the reliability of the extracted semantic-graphs.

4.1 MATLAB Simulations for HSMM Algorithms

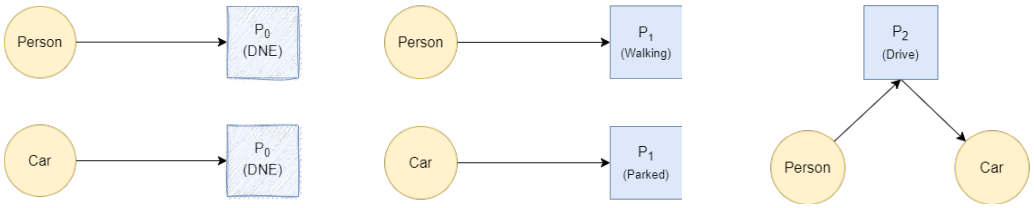
The graph used for MATLAB simulations consists of multiple sub-graphs. Each of these sub-graphs is identical and evolves independently from the others. Specifically, a sub-graph includes two components and three predicates, leading to a total of five possible states.

Sub-graph's representation in the proposed language is given in Figure 4.1. The embedded Markov Chain associated with this sub-graph is presented in Figure 4.2. To obtain the state transition matrix for the overall graph the state transition matrix of the original sub-graph given in (4.1) is combined using the Kronecker product. This allows for the construction of the state transition matrix that governs the evolution of the entire graph. Given that the graph is constituted of independent sub-graphs, each having five possible states, a graph containing n sub-graphs has 5^n states in total.

$$A_{sub} = \begin{bmatrix} 0.000 & 0.950 & 0.045 & 0.005 & 0.000 \\ 0.750 & 0.000 & 0.045 & 0.205 & 0.000 \\ 0.750 & 0.225 & 0.000 & 0.025 & 0.000 \\ 0.500 & 0.050 & 0.200 & 0.000 & 0.250 \\ 0.450 & 0.450 & 0.050 & 0.050 & 0.000 \end{bmatrix} \tag{4.1}$$

As discussed in the previous chapter, the key difference between HMM and HSMM lies in the modelling of state waiting (sojourn) times. In HSMMs, these distributions must be stated explicitly for each state. In practice, it is necessary to bound the waiting time distributions for implementation of algorithms. The waiting time distributions can be either parametric or non-parametric.

In the presented simulation results, a parametric distribution, Laplace distribution,



(a) Out of the scene (P_0) state

(b) Walking/parked (P_1) state

(c) Driving/interacting (P_2) state

Figure 4.1: Sub-graph used for MATLAB simulations. The sub-graph consists of two components: person and car. The P_0 predicate represents the situation where the component is out of the scene. The P_1 predicate indicates that the component is in the scene but not interacting with the other one. In the context of the person and car components, this corresponds to the person walking and the car being parked respectively. There is also a third predicate, P_2, which represents the scenario where the person is driving the car. This predicate indicates the interaction between the two components. The components can enter or leave the scene independently of each other, leading to four possible states represented by the combinations of connections to predicates P_0 and P_1. The fifth state of the sub-graph is when the person is driving the car, represented by the P_2 predicate.

is utilized. The probability density function (pdf) of the Laplace distribution is provided as:

$$f(x|\mu, b) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right). \tag{4.2}$$

For the presented simulation results, the waiting time distributions are derived by sampling this pdf over the set D which the waiting time is defined over. The sampled vector is normalized to ensure it forms a probability mass function (pmf) which sums up to one.

In the simulations, the underlying state (graph) sequence Q is generated based on the model parameters ($p_j(d)$, $\mathbf{A}$, π). At each time instance, an observed state is obtained by considering the underlying state (q_t) and the observation kernel ($b_{q_t}(o_t)$),

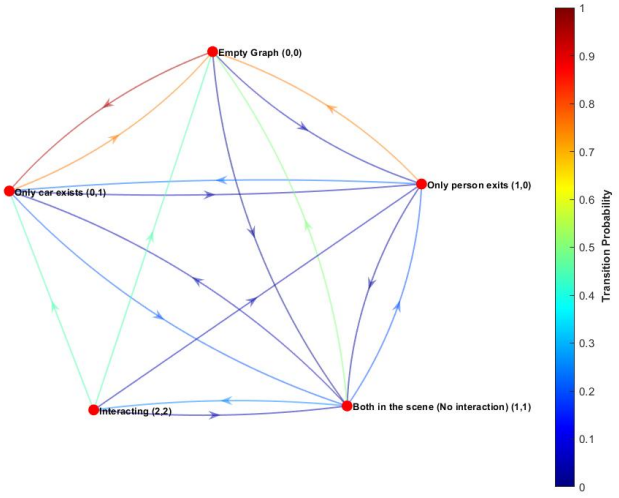


Figure 4.2: Embedded Markov Chain of the HSMM corresponding to the sub-graph. The states of the sub-graph are expressed as tuples with 0, 1, or 2, representing the predicates to which the components are connected. In this representation, each state transition corresponds to a change in the component-predicate connections. It is important to recall that the embedded Markov Chain does not include self-transitions, meaning that a state cannot transition to itself.

as depicted in Figure 3.3. It is important to note that the observed state may deviate from the true state with error probability P_e.

The distribution of error across different states is also important. In the simulation results presented, unless specified otherwise, the error probability is set proportional to the state transition probabilities:

$$b_i(j) = \frac{a_{ij} + a_{ji}}{\sum_{k \in S} a_{ik} + a_{ki}} P_e, \quad (j \neq i). \tag{4.3}$$

This decision is based on the idea that states with higher transition probabilities are more likely to be confused with each other in practice. Moreover, the smoothing

algorithm finds it challenging to distinguish between states that have high transition probabilities to and from each other.

The parameters, including the error rate for observation, state count (N), and waiting time distribution parameters (μ and b), are clearly specified for the simulation results presented. The smoothing algorithms are utilized to correct the errors in the observed state (graph) sequences. Error rates for both the raw observed sequence (O) and the smoothed sequence ($\hat{Q}$) are showcased across different observation error probabilities, denoted as P_e.

Whenever the observed state differs from the correct state at a given time step in the observed or smoothed sequence, it is classified as an error. This discrepancy matters because a different semantic-graph (state) implies a distinct semantic interpretation of the underlying signal. The error rate is calculated as the proportion of time steps with incorrect states to the overall duration of the sequence. It provides a measure of the accuracy of the observed and smoothed graph sequences. Additionally, in cases where relevant, the elapsed time of the algorithm is reported to evaluate the computational efficiency of the employed algorithms.

4.1.1 Smoothing Performance of HMM and HSMM Algorithms

Before diving into detailed results for the HSMM (ED-HMM) algorithms, a comparison is made between the Viterbi-based smoothing performances of HMM and HSMM formulations across various error rates. A graph containing a single sub-graph is simulated as a Semi-Markov Process with incorrect observations. The waiting time duration for each state follows the same Laplace distribution illustrated in Figure 4.3. Self-transition probabilities in the HMM are chosen such that the mean waiting time (sojourn time) for each state matches the mean of the true Laplace distribution.

The state transition matrix of the HMM, with $\mu = 50$, is presented below:

$$A_{HMM} = \begin{bmatrix} 0.980 & 0.019 & 0.001 & 0.000 & 0.000 \\ 0.015 & 0.980 & 0.001 & 0.004 & 0.000 \\ 0.015 & 0.005 & 0.980 & 0.001 & 0.000 \\ 0.010 & 0.001 & 0.004 & 0.980 & 0.005 \\ 0.009 & 0.009 & 0.001 & 0.001 & 0.980 \end{bmatrix}. \tag{4.4}$$

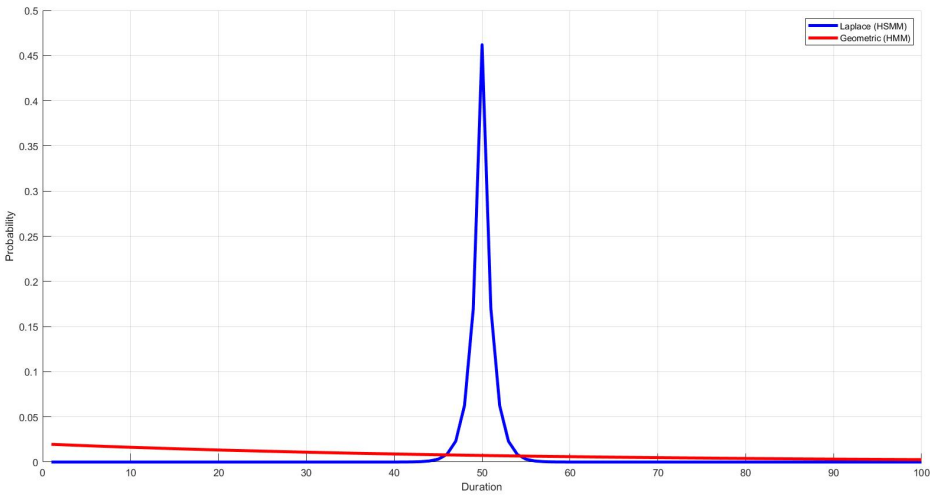


Figure 4.3: State waiting time distributions for HMM and HSMM formulations. The graph (state) sequences are generated according to the Laplace waiting time distribution, represented by the blue curve, with parameters $\mu = 50$ and $b = 1$. The HSMM-Viterbi Algorithm employs this distribution for smoothing observed sequences. In contrast, the waiting times in the Markov setting must follow a geometric distribution. The self-transition probability for each state is chosen such that the mean of this distribution matches the mean of the Laplace distribution. The state waiting time in the HMM formulation is represented by the red curve, which differs significantly from the correct one.

The same observation matrices are used for Viterbi Algorithms in both HMM and HSMM settings. The observed sequence is smoothed for ten different error rates, and

the ratio of time steps where the true sequence and the smoothed sequences differ is used to evaluate the correction performance. The error rates and proportions of incorrect states are presented in Table 4.1. While both algorithms perform similarly at low P_e values, the improvement with the HSMM formulation becomes apparent at higher error rates. Specifically, for $P_e = 0.5012$, the HSMM-Viterbi Algorithm reduces the error rate to 0.0230, marking a significant improvement over the HMM-Viterbi's error rate of 0.2030.

	Incorrect state proportion									
Error rate (P_e)	0.0010	0.0020	0.0040	0.0079	0.0158	0.0316	0.0631	0.1259	0.2512	0.5012
HSMM-Viterbi	0.0000	0.0001	0.0001	0.0002	0.0005	0.0010	0.0020	0.0041	0.0090	0.0230
HMM-Viterbi	0.0000	0.0001	0.0002	0.0004	0.0008	0.0018	0.0032	0.0069	0.0189	0.2032

Table 4.1: Error rate for HMM and HSMM Viterbi Algorithms. Each row presents the ratio of incorrect states to the total sequence length, $T = 20000$, for the corresponding sequence. These ratios, excluding the P_e values in the first row, represent averages calculated from 50 independent iterations for each unique error rate P_e. In each run, the same observed sequence is smoothed using the two algorithms. While the two algorithms perform similarly at low P_e values, the limitations of HMM become more apparent as P_e increases.

4.1.2 Viterbi and Forward-Backward Algorithms for Semantic-Graph Smoothing

To demonstrate the error correction capabilities of the Viterbi and the Forward-Backward algorithms, described in Algorithm 1 and Algorithm 4, a simulation involving two sub-graphs is conducted for a range of error rates. Given that the graph consists of two sub-graphs, the total number of states is 25 ($N = 25$). Each state waiting time distribution follows the same Laplace distribution, which is bounded by 200 ($D' = 200$). This distribution is given in Figure 4.4.

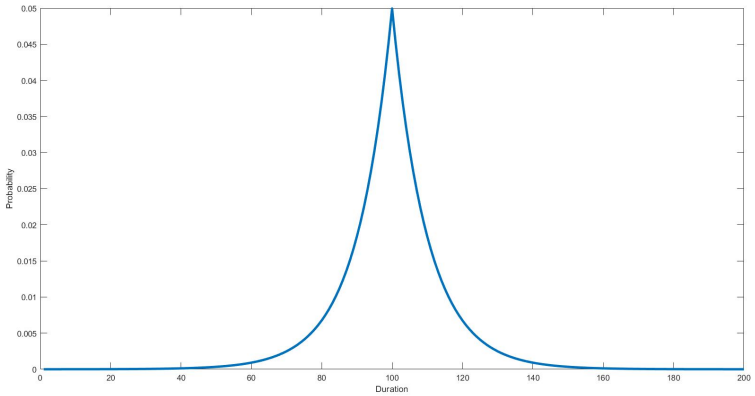


Figure 4.4: Waiting time distribution of each state. Duration is defined between 1 and 200. The location parameter (μ) is chosen as 100 and scale parameter (b) is set to 10.

Observation sequences are generated using ten different error rates (P_e), which are logarithmically spaced between 0.0010 and 0.5012. The sequence length (T) is set to 10000. In every iteration, a path is simulated based on the model parameters $\mathbf{A}$ and $p_j(d)$, beginning at state 1 (S_1). At every time step t, the observed state o_t is determined solely by the current state q_t. The probability of observing the correct state is $1 - P_e$. On the other hand, the probability of observing an incorrect state depends on the state transition probabilities, as detailed in (4.3).

For each P_e value, a state sequence and an observation sequence are generated. The observed sequence is smoothed using the Viterbi and Forward-Backward algorithms. Sequence generation and smoothing via the two algorithms is repeated 100 times for each error rate. Results presented in Table 4.2 are averages of 100 independent runs for each error rate.

As expected, smoothing the observed sequences considerably reduces the ratio of incorrect states (graphs), leading to a much more reliable representation of the underlying semantic information. For instance, with P_e values of 0.2512 and 0.5012,

	Incorrect state proportion									
Error rate (P_e)	0.0010	0.0020	0.0040	0.0079	0.0158	0.0316	0.0631	0.1259	0.2512	0.5012
Observed Sequence	0.0010	0.0020	0.0040	0.0080	0.0158	0.0319	0.0629	0.1258	0.2509	0.5009
Estimated Sequence (MAP)	0.0000	0.0000	0.0000	0.0001	0.0001	0.0003	0.0006	0.0013	0.0025	0.0057
Estimated Sequence (MLE)	0.0000	0.0000	0.0000	0.0001	0.0001	0.0003	0.0006	0.0013	0.0025	0.0059
Difference	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0001	0.0001	0.0002	0.0012

Table 4.2: Error rate for MAP and ML estimators. Each row provides the ratio of incorrect states to the sequence length, $T = 10000$, for the corresponding sequence. These ratios, except the P_e values in the first row, represent the averages obtained from 100 independent iterations for each unique error rate (P_e). Second ratio is the average error of observed sequence which is almost identical to the P_e as expected. The third and fourth rows provide the ratio of time steps where the sequences smoothed by the Forward-Backward (MAP) and Viterbi (ML) algorithms deviate from the correct sequence. The final row gives the ratio of steps at which the sequences from the MAP and ML estimators differ.

the observation sequence displays near-identical error rates of 0.2509 and 0.5009 respectively. This implies that for every four time steps in the former and one in every two in the latter, the observed graph description is incorrect. The ML estimator (Viterbi) and the MAP estimator (Forward-Backward) decrease these rates to 0.0025 and below 0.0060, 0.057 for MAP and 0.0059 for MLE.

The performance of the ML (Viterbi) and MAP (Forward-Backward) estimators is strikingly similar across all P_e values. The last row in Table 4.2 gives ratio of time steps which the two estimators differ which is very small. Even at the P_e value of 0.5012, the highest error rate for the observed sequence, the difference ratio for the estimated sequences is 0.0012. This suggests that both algorithms estimate nearly identical sequences.

4.1.3 Reduced Time Complexity Smoothing Algorithms

The simulation results showcase the improvement in semantic-graph accuracy afforded by the smoothing algorithms, in line with expectations. However, as discussed earlier, these algorithms could be too time-consuming for practical applications, especially under computational limitations.

This subsection presents simulation results for scenarios involving graphs with large state spaces and extended waiting time durations. The aim is to evaluate the efficacy of the reduced time complexity algorithms.

To compare the error correction performance of these algorithms with the original Viterbi Algorithm, the same ten logarithmically spaced P_e values given in the previous subsection are used. The reported error rates of smoothed sequences and the time taken for the algorithms to execute under various parameters represent average values.

To evaluate the performance of the ED-HMM M-Algorithm, provided in Algorithm 5, a simulation involving a graph with a large state space is conducted. The graph used in the simulation consists of four sub-graphs, resulting in a total of 625 states ($N = 625$). Every state follows the same Laplace waiting time distribution with parameters $\mu = 20$ and $b = 3$, defined within the range of 1 to 40 (where $D' = 40$). This distribution is given in Figure 4.5.

For each P_e value, a state (graph) sequence of length 10000 is generated. The corresponding observation sequence, which includes incorrect states (graphs), is smoothed using both the Viterbi algorithm and the M-Algorithm with M values set to $5, 25$ and 100. This process of generation and smoothing is repeated for 25 independent sequences for every P_e value.

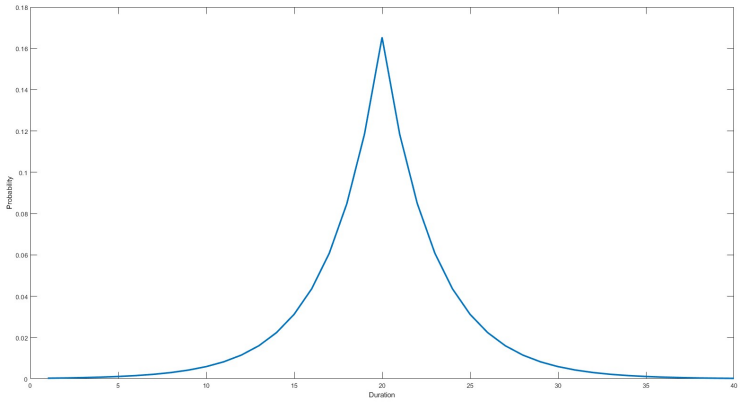


Figure 4.5: Waiting time distribution. Duration is defined between 1 and 40. The location parameter (μ) is chosen as 20 and scale parameter (b) is set to 3. The same distribution is used for each state in the M-Algorithm simulations.

The average error rates can be found in Table 4.3. Additionally, Table 4.3 presents the average elapsed time for each M value as well as for the Viterbi Algorithm. The elapsed times are calculated over 250 iterations, as the time required for the smoothing algorithm remains unaffected by P_e.

As expected, correcting errors in the observed sequences using the M-Algorithm takes considerably less time than the original Viterbi Algorithm. On average, searching the preceding state among the 5 and 25 most likely states, as opposed to all 625 states, reduces the time required by factors of 72 and 19, respectively. This aligns with the anticipated linear improvement.

It is interesting to observe performance identical to the Viterbi Algorithm when smoothing the same observed sequence using the M-Algorithm, even with varying M values. This can probably be attributed to the segmental structure of state (graph) sequences. Since the same state tends to be occupied over successive time steps, the correct state is likely present among the M most probable states. As a result, the M-Algorithm can find the most likely state for the relevant segment in the same

	Incorrect state proportion										Elapsed time (s)
Error rate (P_e)	0.0010	0.0020	0.0040	0.0079	0.0158	0.0316	0.0631	0.1259	0.2512	0.5012	
M-Algorithm ($M = 5$)	0.0000	0.0000	0.0000	0.0001	0.0001	0.0002	0.0005	0.0011	0.0020	0.0037	9.2989
M-Algorithm ($M = 25$)	0.0000	0.0000	0.0000	0.0001	0.0001	0.0002	0.0005	0.0011	0.0020	0.0037	34.5674
M-Algorithm ($M = 100$)	0.0000	0.0000	0.0000	0.0001	0.0001	0.0002	0.0005	0.0011	0.0020	0.0037	136.4369
Viterbi Algorithm	0.0000	0.0000	0.0000	0.0001	0.0001	0.0002	0.0005	0.0011	0.0020	0.0037	673.5741

Table 4.3: Error rate and elapsed time for M-Algorithm simulations. Each row shows the ratio of incorrect states to the sequence length of $T = 10000$, except for the last column which indicates the time taken by the algorithm to smooth the observed sequence. The error ratios are averages obtained from 25 independent iterations for each distinct error rate (P_e). The smoothing performance of the M-Algorithm, even at $M = 5$, matches that of the Viterbi Algorithm while the M-Algorithm at $M = 5$ is over 72 times faster. Given that the elapsed time is not dependent on P_e, the elapsed time column represents the average from 250 runs for each algorithm.

manner as the Viterbi Algorithm.

In addition to large state spaces with many possible graph configurations (large N), the Viterbi Algorithm can also become infeasible due to extended state waiting time durations (large D'). In such scenarios, the Quantized-Duration ED-HMM Viterbi Algorithm, presented in Algorithm 6, offers a way to reduce the computational complexity of the graph smoothing process.

To evaluate the error correction capabilities of the Quantized-Duration ED-HMM Viterbi Algorithm, a graph with extended waiting time durations is simulated. The simulated graph contains two sub-graphs hence the state space includes 25 distinct

graph configurations ($N = 25$). Each state follows the waiting time distribution given in Figure 4.6, which is a Laplace distribution defined from 1 to 600 ($D' = 600$) and is characterized by parameters $\mu = 300$ and $b = 25$. Observation sequences are generated using the same P_e value range as in previous simulations.

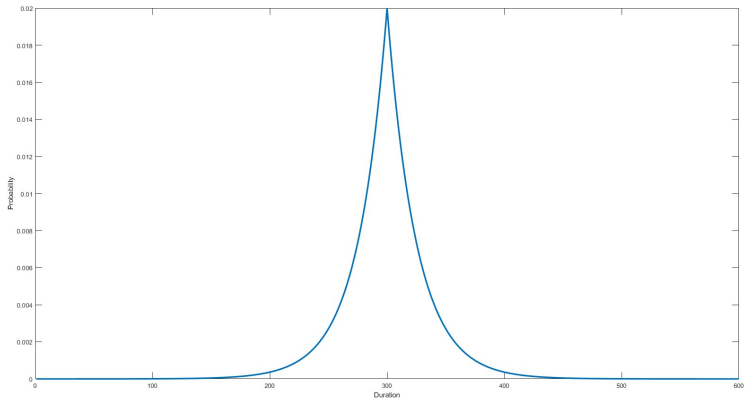


Figure 4.6: Waiting time distribution. Duration is defined between 1 and 600. The location parameter (μ) is chosen as 300 and scale parameter (b) is set to 25. The same distribution is used for each state in Quantized-Duration Algoritm simulations.

For each P_e, a state sequence of length 100000 is generated. This sequence is then smoothed using both the Viterbi Algorithm and the Quantized-Duration ED-HMM Viterbi Algorithm with quantization levels of $n = 10$, $n = 25$, $n = 50$ and $n = 150$. This process is repeated for 50 independent sequences for every P_e value. The error rates shown in Table 4.4 represent the averages from these iterations. Table 4.4 also presents the average elapsed time for smoothing using the Quantized-Duration ED-HMM Viterbi Algorithm across different quantization levels. These are averages of 500 values since the computational complexity of smoothing is independent of P_e value.

The error correction performance of the Quantized-Duration ED-HMM Viterbi Algorithm is significantly influenced by the number of quantization level n. This algorithm estimates the underlying state (graph) sequence using segments from a

	Incorrect state proportion										Elapsed time (s)
Error rate (P_e)	0.0010	0.0020	0.0040	0.0079	0.0158	0.0316	0.0631	0.1259	0.2512	0.5012	
Quantized ($n = 10$)	0.0237	0.0233	0.0237	0.0233	0.0259	0.0329	0.0437	0.0371	0.0289	0.0234	12.65
Quantized ($n = 25$)	0.0104	0.0101	0.0104	0.0098	0.0106	0.0129	0.0152	0.0140	0.0104	0.0094	16.98
Quantized ($n = 50$)	0.0054	0.0054	0.0055	0.0051	0.0056	0.0065	0.0075	0.0075	0.0057	0.0055	24.23
Quantized ($n = 150$)	0.0018	0.0018	0.0019	0.0017	0.0019	0.0022	0.0026	0.0028	0.0024	0.0031	54.06
Viterbi	0.0000	0.0000	0.0000	0.0000	0.0000	0.0001	0.0002	0.0004	0.0009	0.0019	175.81

Table 4.4: Error rate and elapsed time for simulations using the Quantized-Duration ED-HMM Viterbi Algorithm. Each row gives the ratio of incorrect states to the sequence length ($T = 100000$). These error rates are the averages of 50 independent iterations for each P_e value. The last column is the average smoothing time in seconds for different n values, computed over 500 runs. As expected, smoothing performance declines with decreasing n. Notably, for very small P_e values, the smoothing error might surpass P_e. However, for larger P_e values (P_e greater than 0.0316), the Quantized Viterbi reduces the incorrect state ratio considerably and does it faster than the standard Viterbi Algorithm.

restricted set of lengths, specifically the n values expressed in (3.36), as opposed to the whole state duration set D. This approximation is the primary source of error and decreases the smoothing performance compared to the Viterbi Algorithm. The error rate for a fixed quantization level (n) remains consistent across varying P_e values, mainly because the main source of error comes from the quantization. The algorithm successfully identifies the dominant state (semantic-graph) within a segment. However, the mismatch between the available segment lengths and the true ones leads to a discrepancy between the estimated and true sequences.

For large P_e values, the Quantized-Duration ED-HMM Viterbi Algorithm improves the graph accuracy significantly and does so more quickly than the Viterbi Algorithm. For example, with P_e value of 0.5012, using the Quantized-Duration ED-HMM Viterbi Algorithm with $n = 50$ reduces the average error rate by a factor of

100, bringing it down from 0.5012 to 0.0055. This performance is comparable to the Viterbi Algorithm, which achieves an error rate of 0.0019, but requires seven times less time.

The performance of the ED-HMM Quantized M-Algorithm, described in Algorithm 7, is evaluated as the last reduced time complexity smoothing algorithm through simulations.

The simulated graph comprises three sub-graphs, resulting in $N = 125$. Each state follows a Laplace waiting time distribution, defined from 1 to 250, characterized by parameters $\mu = 125$ and $b = 15$. This waiting time distribution is given in Figure 4.7.

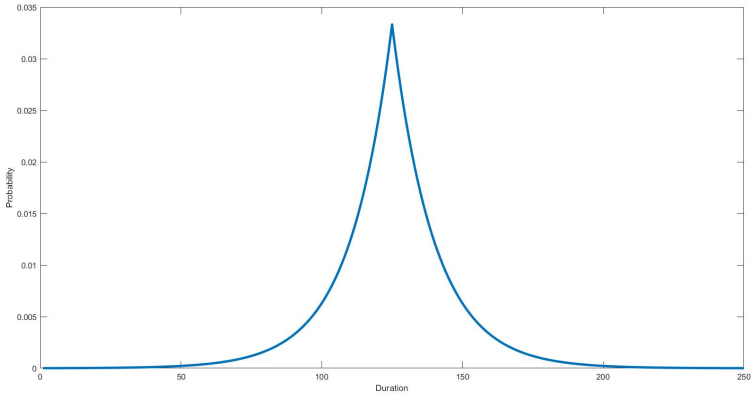


Figure 4.7: Waiting time distribution. Duration is defined between 1 and 250. The location parameter (μ) is chosen as 125 and scale parameter (b) is set to 15. The same distribution is used for every state in Quantized M-Algorithm simulations.

The ED-HMM Quantized M-Algorithm combines the ideas of restricting the state space and quantizing the state duration distribution of each state. For the estimated sequence, states are selected from $M = 5$, $M = 10$, $M = 25$ most likely states and quantization levels are set as $n = 25$, $n = 50$,$n = 125$. Observed state (graph) sequences are smoothed using each of the nine possible combinations of these model

parameters. Additionally, for comparison, the same sequence is smoothed using the Viterbi Algorithm.

The error correction performance of these algorithms is evaluated using the same ten P_e values from previous simulations. Sequence length (T) is chosen as 20000. For each P_e value, the sequence generation and smoothing processes are repeated 25 times using all model parameter combinations, as well as the Viterbi Algorithm. The error rate and elapsed time values presented in Table 4.5 represent the averages of these iterations.

Simulation results for the ED-HMM Quantized M-Algorithm align with those observed for the two original algorithms upon which it is based. The error correction performance is mostly same across different M values. However, using smaller M values reduces smoothing time. The number of quantization levels, n, significantly affects error correction performance, a decrease in n corresponds to a higher error rate, even if smoothing time is reduced. Particularly, with heavy quantization, $n = 25$, error rates appear to be consistent across the P_e value range. This indicates that the differences between the smoothed and true sequences primarily arise from limitations in segment lengths, rather than from incorrect observations.

	Incorrect state proportion										Elapsed time (s)
Error rate (P_e)	0.0010	0.0020	0.0040	0.0079	0.0158	0.0316	0.0631	0.1259	0.2512	0.5012	
$n = 25$, $M = 5$	0.0119	0.0117	0.0123	0.0116	0.0122	0.0134	0.0155	0.0195	0.0164	0.0132	5.6568
$n = 25$, $M = 10$	0.0119	0.0117	0.0123	0.0116	0.0122	0.0134	0.0155	0.0196	0.0164	0.0132	8.3685
$n = 25$, $M = 25$	0.0119	0.0117	0.0123	0.0116	0.0122	0.0134	0.0155	0.0196	0.0164	0.0132	16.6919
$n = 50$, $M = 5$	0.0058	0.0056	0.0062	0.0060	0.0058	0.0069	0.0075	0.0089	0.0091	0.0070	8.3697
$n = 50$, $M = 10$	0.0058	0.0056	0.0062	0.0060	0.0058	0.0069	0.0075	0.0089	0.0091	0.0070	13.3561
$n = 50$, $M = 25$	0.0058	0.0056	0.0062	0.0060	0.0058	0.0069	0.0075	0.0089	0.0091	0.0070	28.8000
$n = 125$, $M = 5$	0.0019	0.0021	0.0027	0.0023	0.0025	0.0027	0.0028	0.0032	0.0041	0.0040	16.5200
$n = 125$, $M = 10$	0.0019	0.0021	0.0027	0.0023	0.0025	0.0027	0.0028	0.0031	0.0041	0.0040	28.5920
$n = 125$, $M = 25$	0.0019	0.0021	0.0027	0.0023	0.0025	0.0027	0.0028	0.0031	0.0041	0.0040	64.0622
Viterbi ($n = 250$, $M = 125$)	0.0000	0.0000	0.0000	0.0000	0.0001	0.0001	0.0002	0.0004	0.0008	0.0015	323.4560

Table 4.5: Error rate and elapsed time for ED-HMM Quantized M-Algorithm.

4.1.4 Semantic Fusion

In practical applications, it is common to fuse information obtained from multiple sensors to make a decision based on a particular signal. Within the context of the proposed semantic signal processing framework, this problem corresponds to fusing semantic-graphs generated by distinct but correlated sensors. Algorithm 8 allows to fuse semantic-graphs generated by two sensors with perfect overlap while incorporating prior knowledge through the HSMM statistics.

A scenario including two sensors with perfect overlap is simulated to showcase the improvement achieved through graph fusion. The simulation scenario has a single sub-graph, state space consists of five distinct states in total. State duration distribution is bounded by 50 steps and it follows a Laplace distribution as given in Figure 4.8 for each state.

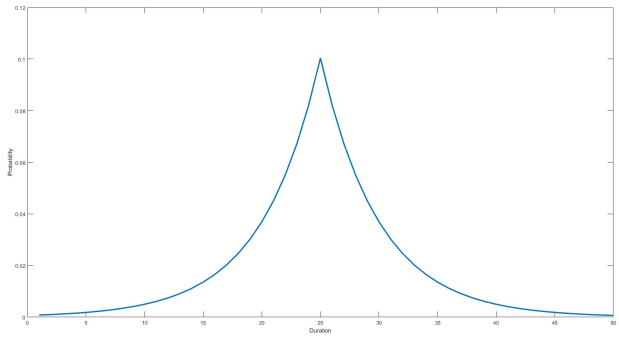


Figure 4.8: Waiting time distribution of each state. Duration is defined between 1 and 50, it is a Laplacian centered at 25 with scale parameter equal to 5.

The error patterns used to generate these observation sequences differ from the one given in (4.3). In these simulations, each state can be mistaken for two other states. Importantly, the states with which confusion arises are uniquely chosen for each sensor, ensuring they complement one another. The observation matrices for both sensors are provided below:

$$B_1 = \begin{bmatrix} 1-P_e & 0 & 0 & P_e/2 & P_e/2 \\ 0 & 1-P_e & P_e/2 & P_e/2 & 0 \\ 0 & P_e/2 & 1-P_e & P_e/2 & 0 \\ 0 & P_e/2 & P_e/2 & 1-P_e & 0 \\ P_e/2 & 0 & 0 & P_e/2 & 1-P_e \end{bmatrix}, \tag{4.5}$$

$$B_2 = \begin{bmatrix} 1-P_e & P_e/2 & P_e/2 & 0 & 0 \\ 0 & 1-P_e & 0 & P_e/2 & P_e/2 \\ 0 & P_e/2 & 1-P_e & 0 & P_e/2 \\ 0 & 0 & P_e/2 & 1-P_e & P_e/2 \\ 0 & P_e/2 & P_e/2 & 0 & 1-P_e \end{bmatrix}. \tag{4.6}$$

For each P_e value, a state sequence and two observation sequences are generated. In addition to estimation of the underlying sequence by fusing information from both observation sequences, the observation sequences are smoothed independently according to their own observation matrices. Sequence generation, smoothing and fusion are repeated 100 times for each P_e. Results presented in Table 4.6 are averages of 100 independent runs for each P_e value.

As anticipated from previous results, independently smoothing the observed sequences significantly reduces the ratio of incorrect states. However, fusing information from both sequences to estimate the underlying state (graph) sequence produces even more favorable outcomes. When evaluated individually, the second sequence performs better than the first. Yet, the sequence derived from the fusion of both demonstrates the best performance.

For small P_e values, from 0.0010 to 0.0316, the incorrect state ratio for both individually smoothed sequences and fused sequences is very small, less than 0.0001. In

	Error rate of observed sequence P_e									
	0.0010	0.0020	0.0040	0.0079	0.0158	0.0316	0.0631	0.1259	0.2512	0.5012
Observed Sequence 1	0.0010	0.0020	0.0040	0.0077	0.0158	0.0317	0.0629	0.1262	0.2509	0.5007
Observed Sequence 2	0.0010	0.0020	0.0039	0.0079	0.0158	0.0319	0.0632	0.1259	0.2513	0.5011
Estimated Sequence 1	0.0000	0.0001	0.0001	0.0002	0.0005	0.0010	0.0019	0.0041	0.0103	0.0587
Estimated Sequence 2	0.0000	0.0001	0.0001	0.0002	0.0004	0.0009	0.0017	0.0034	0.0076	0.0218
Fusion	0.0000	0.0000	0.0000	0.0000	0.0001	0.0001	0.0003	0.0006	0.0019	0.0070

Table 4.6: Error rate for Semantic Fusion. Each row provides the ratio of incorrect states to the sequence length for the corresponding sequence. These ratios represent the averages obtained from 100 independent iterations for each P_e value.

contrast, for higher P_e, particularly at 0.5012, the increase in reliability from utilizing information from both sequences to infer the underlying semantic graph becomes evident. Although, even when half of the states in the observed sequences are incorrect, this ratio decreases to 7 in 1000 with fusion. This is a notable improvement over the individually smoothed sequences which have error rates of 0.0587 and 0.0218 respectively.

4.1.5 Model Learning

The results presented thus far are obtained using algorithms with perfect knowledge of the model. However, in practical applications, it is not possible to have such accurate knowledge of the model. Mostly the model parameters need to be learned from observations. In this subsection, the practicality of the learning algorithm described in Algorithm 9 is investigated for estimating HSMM parameters from noisy

observations.

The following results are based on the simulation of a single sub-graph, which can be in one of five states. Every state follows a Laplace waiting time distribution with unique parameters, as illustrated in Figure 4.9. The performance of the learning algorithm is evaluated across ten distinct error rates (P_e). For each P_e value, an observation sequence of length 100000 is generated. This observation sequence is given to the learning algorithm with initial guesses for the model parameters.

For the Baum-Welch Algorithm, the initialization of the model parameters is crucial. The state transition matrix ($\mathbf{A}$) is initialized randomly, except for the zeros along its diagonal. In the observation matrix ($\mathbf{B}$), diagonal elements are set at 0.5 for every state, based on the assumption that the correct observation is most likely for each state. The off-diagonal elements of $\mathbf{B}$ are chosen randomly. Each row of $\mathbf{A}$ and $\mathbf{B}$ is normalized to sum up to 1. Waiting time distribution for each state is initialized as a uniform distribution. The initial state distribution (π) is assumed to be perfectly known. The first state is always S_1 and π is initialized accordingly. For each P_e, the Baum-Welch Algorithm is run for 25 iterations.

The learning algorithm, Algorithm 9, treats the model parameters as non-parametric distributions. However, when a certain model parameter is known to follow a parametric distribution, the algorithm can be modified to estimate the related parameters.

Alongside Algorithm 9, which estimates the model parameters non-parametrically, a variation is employed specifically for estimating the Laplace distribution parameters (μ and b). Except for the update specified in (3.49), two algorithms are identical.

For the parametric estimation of the Laplace waiting time distribution, the observed state sequence undergoes smoothing using the most recent model parameters.

Once the sequence is corrected, the histogram of segment lengths for each state is obtained. ML estimator for Laplace distribution is then used to determine the required parameters for each state as follows [42]:

$$\hat{\mu}_i = med\{X_{i,1}, \ldots, X_{i,n_i}\}, \tag{4.7}$$

$$\hat{b}_i = \frac{1}{n_i} \sum_{k=1}^{n_i} |X_{i,k} - \hat{\mu}_i|. \tag{4.8}$$

$X_{i,k}$ in (4.7) and (4.8) represents the length of k^{th} segment identified as state i (S_i). The location parameter, μ, is determined as the median of these segment lengths, n_i segments for S_i, while the scale parameter b is estimated as the average absolute deviation of n_i segments from the estimated μ value.

To evaluate the accuracy of the estimated model, 50 distinct graph sequences, separate from the one used for learning and each with a length of 100000, are generated for every P_e value. During each iteration, the Viterbi Algorithm is used to smooth the observation sequence based on the estimated model parameters. Furthermore, these sequences are also smoothed employing the correct model parameters. The average error rates from these smoothed sequences provided in Table 4.7 are used to evaluate the learning performance.

Running the Viterbi Algorithm on the observed sequences with the estimated model parameters substantially reduces the proportion of incorrect state (graph) instances, for both of the Baum-Welch Algorithm variants used. Even at the highest observation error rate, $P_e = 0.5$, smoothing with the estimated models brings the error rate down to 0.0796 and 0.0738 for parametric and non-parametric duration estimations, respectively.

The error rates of sequences smoothed using the estimated models closely match

	Error rate of observed sequence P_e									
	0.0010	0.1000	0.1500	0.2000	0.2500	0.3000	0.3500	0.4000	0.4500	0.5000
Smoothing with Perfect Model	0.0000	0.0046	0.0076	0.0114	0.0161	0.0220	0.0295	0.0395	0.0529	0.0706
Smoothing with Parametric Estimation	0.0001	0.0046	0.0076	0.0114	0.0161	0.0222	0.0295	0.0417	0.0666	0.0796
Smoothing with Non-Parametric Estimation	0.0010	0.0061	0.0096	0.0140	0.0186	0.0254	0.0332	0.0445	0.0562	0.0738

Table 4.7: Error rate for smoothing with estimated model parameters. Each row provides the ratio of incorrect states to the sequence length for the corresponding sequence. These ratios represent the averages obtained from 50 independent sequences for each unique error rate. The observation sequence is smoothed using both the perfect model and estimated models for parametric and non-parametric duration estimations. The error rate of smoothing with estimated models closely mirrors that of smoothing with the ideal model parameters.

the performance of the correct model for every P_e value, indicating the algorithm's proficiency in accurately estimating the parameters from the observed sequences. The length of the sequence utilized for learning, equal to 100000 for each P_e in the presented results, plays a pivotal role in ensuring the learning algorithm functions correctly. Especially the state-duration pairs and state-observation pairs need to be frequently visited for the algorithm to reliably estimate the state waiting time durations ($p_j(d)$) and the observation probabilities ($b_i(j)$).

For a specific error rate, $P_e = 0.25$, the waiting time durations of each state are given in Figure 4.9. Parametric approach produces smoother and more accurate

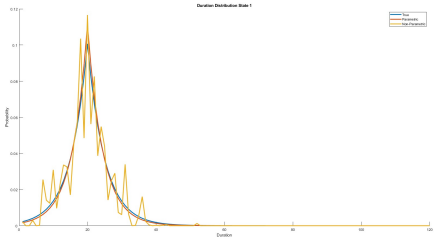
results as expected. The non-parametric estimation exhibits oscillatory behavior, particularly for S_5. This is because it does not operate with prior knowledge of the probability mass function (pmf) it is aiming to estimate. Thus, to determine their relative frequencies with precision, each state-duration pair must be observed sufficiently often.

The true state transition matrix (A) is provided in (4.1). For both versions of the learning algorithm, the estimated A closely mirrors the true matrix, with the estimated state transition probabilities also being notably similar:

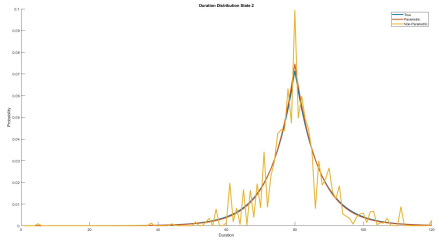
$$\hat{A}_{param} = \begin{bmatrix} 0.0000 & 0.9651 & 0.0322 & 0.0027 & 0.0000 \\ 0.7570 & 0.0000 & 0.0395 & 0.2035 & 0.0000 \\ 0.7343 & 0.2245 & 0.0000 & 0.0411 & 0.0000 \\ 0.5441 & 0.0247 & 0.2242 & 0.0000 & 0.2069 \\ 0.4267 & 0.4342 & 0.0280 & 0.1111 & 0.0000 \end{bmatrix}, \tag{4.9}$$

$$\hat{A}_{non-param} = \begin{bmatrix} 0.0000 & 0.9652 & 0.0321 & 0.0027 & 0.0000 \\ 0.7555 & 0.0000 & 0.0376 & 0.2068 & 0.0000 \\ 0.7274 & 0.2246 & 0.0000 & 0.0480 & 0.0000 \\ 0.5447 & 0.0250 & 0.2274 & 0.0000 & 0.2028 \\ 0.4211 & 0.4395 & 0.0278 & 0.1115 & 0.0000 \end{bmatrix}. \tag{4.10}$$

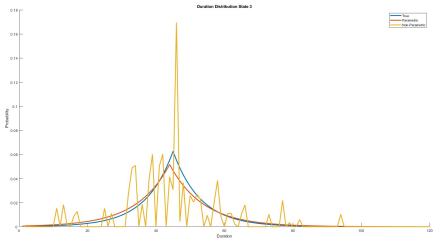
The observation matrix (B) is also accurately estimated, with both the parametric and non-parametric approaches yielding similar results. The actual and estimated matrices are provided below:



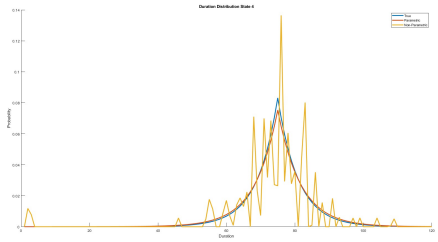
(a) State-1 (S_1): $\mu_1 = 20$, $b_1 = 5$. $\hat{\mu}_1 = 20$, $\hat{b}_1 = 4.52$.



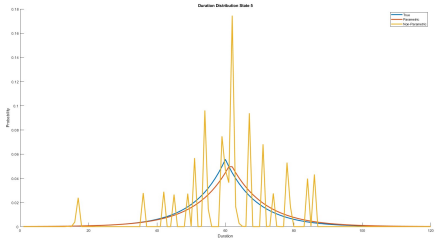
(b) State-2 (S_2): $\mu_2 = 80$, $b_2 = 7$. $\hat{\mu}_2 = 80$, $\hat{b}_2 = 6.69$.



(c) State-3 (S_3): $\mu_3 = 45$, $b_3 = 8$. $\hat{\mu}_3 = 44$, $\hat{b}_3 = 9.73$.



(d) State-4 (S_4): $\mu_4 = 75$, $b_4 = 6$. $\hat{\mu}_4 = 75$, $\hat{b}_4 = 6.63$.



(e) State-5 (S_5): $\mu_5 = 60$, $b_5 = 9$. $\hat{\mu}_5 = 62$, $\hat{b}_5 = 9.61$.

Figure 4.9: Estimated waiting time durations for P_e value of 0.25. The blue curves represent the true distributions, red and yellow curves correspond to parametric and non-parametric estimations respectively. As anticipated, the parametric estimation yields smoother results for every state compared to the non-parametric method. Notably, the duration distribution for the fifth state shows oscillations in the non-parametric estimation. Below each graph, the true and estimated location (μ) and scale (b) parameters are given.

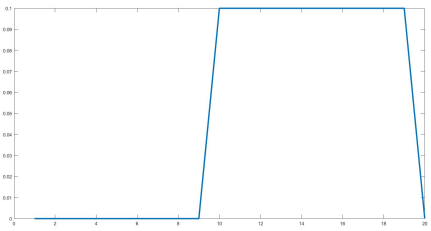
$$B_{true} = \begin{bmatrix} 0.7500 & 0.1232 & 0.0576 & 0.0366 & 0.0326 \\ 0.1589 & 0.7500 & 0.0252 & 0.0238 & 0.0421 \\ 0.1483 & 0.0504 & 0.7500 & 0.0420 & 0.0093 \\ 0.0982 & 0.0496 & 0.0438 & 0.7500 & 0.0584 \\ 0.0900 & 0.0900 & 0.0100 & 0.0600 & 0.7500 \end{bmatrix}, \tag{4.11}$$

$$\hat{B}_{param} = \begin{bmatrix} 0.7512 & 0.1222 & 0.0590 & 0.0349 & 0.0326 \\ 0.1561 & 0.7523 & 0.0248 & 0.0251 & 0.0418 \\ 0.1433 & 0.0501 & 0.7513 & 0.0460 & 0.0093 \\ 0.0980 & 0.0497 & 0.0442 & 0.7514 & 0.0567 \\ 0.0861 & 0.0984 & 0.0083 & 0.0548 & 0.7524 \end{bmatrix}, \tag{4.12}$$

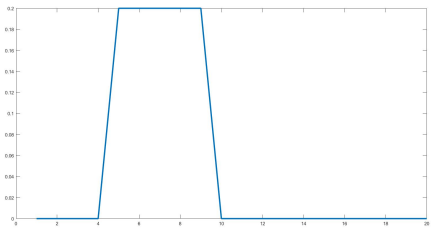
$$\hat{B}_{non-param} = \begin{bmatrix} 0.7525 & 0.1210 & 0.0589 & 0.0349 & 0.0327 \\ 0.1561 & 0.7523 & 0.0248 & 0.0250 & 0.0418 \\ 0.1428 & 0.0497 & 0.7537 & 0.0448 & 0.0089 \\ 0.0977 & 0.0495 & 0.0442 & 0.7519 & 0.0567 \\ 0.0855 & 0.0984 & 0.0081 & 0.0548 & 0.7532 \end{bmatrix}. \tag{4.13}$$

4.2 CARLA Experiments: Assessing HSMM Effectiveness in Realistic Video Scenarios

In this section, the effectiveness of HSMM-based algorithms in improving graph accuracy is demonstrated through experiments involving video streams. These streams are generated in the CARLA simulation environment, an autonomous driving simulator developed on the Unreal Engine platform [43]. CARLA enables the creation of scenarios in diverse urban environments and provides control over various objects, like vehicles and pedestrians. It also offers the capability to render videos from these simulations. The presented results are obtained by applying a segmentation-based



(a) Distribution of idle/parked state



(b) Distribution of slow and fast states

Figure 4.10: Waiting time distribution of three states used in CARLA simulations. The duration of the parked state is uniformly distributed between 10 and 19. For the slow and fast moving states, the waiting time is uniformly distributed between 5 and 9. These distributions are actually defined over integers (as pmfs), so the rising and falling edges in the plots can be ignored.

semantic extractor to the rendered video frames that include a single car.

In the presented results, a state (semantic-graph) sequence is generated based on a Semi-Markov Process (SMP) with three states. This sequence evolves according to the state transition matrix given in (4.14) and the waiting time distributions given in Figure 4.10. These three states correspond to a car being parked/idle (S_1), moving slowly (S_2), and moving quickly (S_3). The initial state is always S_1, the parked state in the simulations.

$$A = \begin{bmatrix} 0.0 & 0.9 & 0.1 \\ 0.8 & 0.0 & 0.2 \\ 0.0 & 1.0 & 0.0 \end{bmatrix} \tag{4.14}$$

In the CARLA simulations, the car moves along a road at speeds determined by the underlying state sequence. When the state is idle, the car remains stationary. In the slow state, the car moves at 1 m/s, while in the fast state, its speed increases to 20 m/s. The simulation starts with the car parked on the left side of the road, as

depicted in Figure 4.11. During each simulation, the car's speed varies and it often stops, all determined by the underlying state sequence. The sequence ends when the car leaves the scene from the right end of the road. Due to the variability in the car's movements, the sequence length is not consistent across different simulations.

To extract the semantic-graph corresponding to a frame in the CARLA simulations, a semantic segmentation algorithm is employed. The car's speed is determined by measuring the difference between its bounding box centers in consecutive frames. However, the main source of error arises when the car is not detected by the semantic extractor, often when it is parked under a tree or blocked by other objects like trees. Despite the car's known presence in these time steps, the semantic extractor cannot use this prior information, as it operates on snapshots. To address this issue, these steps are assigned a new state named DNE (does not exist, S_0) in the observation.

Five CARLA simulations are generated on the road shown in Figure 4.11, and state (graph) sequences are extracted using the segmentation algorithm, as depicted in Figure 4.12. To enhance the reliability of the extracted sequences, the Viterbi Algorithm is employed, which allows for the incorporation of prior information about the underlying process. As previously mentioned, the primary source of error in the extracted sequences is the DNE state, which corresponds to time steps when the car is not visible. The smoothing algorithm is utilized to correct these errors in the observed sequences, ensuring that the estimated sequence assigns zero probability to any path that includes the DNE state. The Viterbi Algorithm is used with the following parameters for the transition and observation matrices:

$$A = \begin{bmatrix} 0.0 & 0.1 & 0.8 & 0.1 \\ 0.0 & \mathbf{0.0} & \mathbf{0.9} & \mathbf{0.1} \\ 0.0 & \mathbf{0.8} & \mathbf{0.0} & \mathbf{0.2} \\ 0.0 & \mathbf{0.0} & \mathbf{1.0} & \mathbf{0.0} \end{bmatrix}, \tag{4.15}$$

(a) Car is stationary at the left end of the road

(b) Car is moving right

(c) Car is parked under the tree, not visible

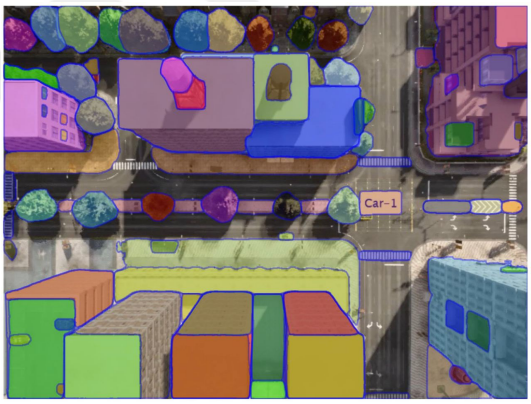
(d) Car starts moving and becomes visible after leaving the area under the tree

Figure 4.11: Snapshots from CARLA simulation. The sequence starts with the car standing still at the road's left end. Its speed changes based on the underlying state, and at times, it becomes stationary. As illustrated in (c), when the car is parked (or passing) under a tree, it goes undetected, leading to an incorrect graph representation. However, once the car resumes its movement, it becomes visible again, and the sequence terminates when the car exits the road on the right side.

(a) Car is stationary at the left end of the road

(b) Car is moving right

(c) Car is parked under the tree, not visible



(d) Car starts moving and becomes visible after leaving the area under the tree

Figure 4.12: Segmented versions of the images given in Figure 4.11. The bounding box of the car in the segmented images allows to detect its speed and assign the corresponding state (semantic-graph) to the observed scene. In instances like (c), where the car is not detected and thus does not have a bounding box, the observed state is labelled as DNE.

$$B = \begin{bmatrix} 0.00 & 0.33 & 0.33 & 0.34 \\ 0.40 & 0.40 & 0.10 & 0.10 \\ 0.05 & 0.15 & 0.70 & 0.10 \\ 0.10 & 0.10 & 0.10 & 0.70 \end{bmatrix}. \tag{4.16}$$

The Viterbi Algorithm is employed with four states, including the DNE state as well as the idle, slow, and fast states present in the true sequence. The correct state transition matrix, highlighted with bold elements, is used for the smoothing process. In this augmented formulation, DNE is designated as the first state (S_0), and the first row and first column in (4.15) and (4.16) correspond to it. Since the DNE state is known to be absent from the true sequence, transitions to the DNE state are assigned a zero probability. Therefore, the first column of (4.15) is filled with zeros. Given that the initial state is always S_1, the DNE state is never visited in the smoothed sequence. The correct waiting time distributions, depicted in Figure 4.10, are used for the state duration parameters.

In contrast to MATLAB simulations where the observation kernel is readily available, the parameters in (4.16) have to be estimated and are chosen based on educated guesses. The first row, corresponding to the DNE state, can be ignored because both the transition matrix and the initial state distribution are set to never visit this state. Consequently, any observed DNE state is decoded as one of the other three states.

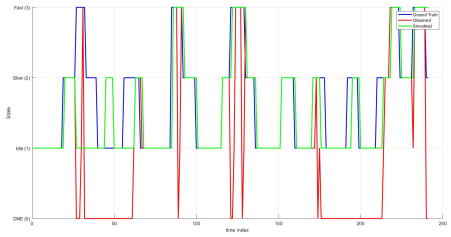
The sequences extracted from five different simulations, smoothed using these model parameters, are illustrated in Figure 4.13. As anticipated, the DNE state is eliminated after smoothing, and the smoothed sequence mostly aligns with the true sequence. However, there is some discrepancy between the smoothed and true sequences. Although the smoothed sequence captures the state transitions mostly accurately, the transition instances and elapsed time in states is not always accurate.

The discrepancy between the smoothed and true sequences arises from a temporal

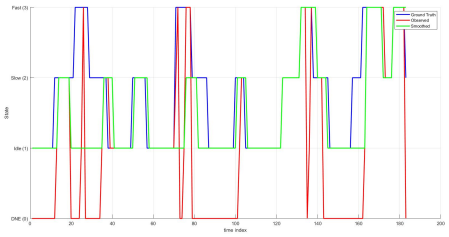
shift in the smoothed sequence, which is introduced by the the extraction algorithm that generates the observed sequence. The car's speed is determined by comparing centers of bounding boxes in consecutive frames; as a result, any change in the car's movement, starting to move, stopping, or changing speed, can be detected in the following frame. However, the presence of obstacles like trees adds complexity to this process.

For the car to be detected, its bounding box must exceed a size threshold. When the car is in the DNE state, parked or passing under a tree, it becomes undetectable and does not a bounding box. As it exits the tree's coverage, there is a delay before its bounding box becomes sufficiently large to be detected. Moreover, the car must be detected in two consecutive frames for its speed to be calculated. Consequently, in addition to the one-frame delay inherent in the state transition detection when the car is visible, the trees introduce a variable delay pattern to the observed sequence. Therefore, while the smoothed sequence generally captures the state transitions, it often falls short in accurately estimating the timing of these transitions and may not reliably estimate the duration of estimated states.

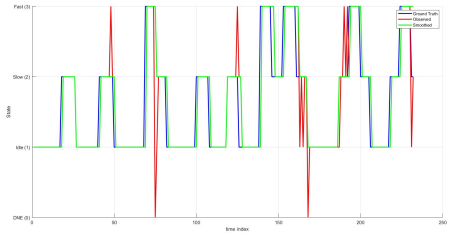
The confusion matrices for the extracted, observed, and smoothed states, which are presented in Figure 4.13, are given in Figure 4.14. The first row in the confusion matrix for the extracted states is empty, as the DNE state is not part of the true sequence. Similarly, the confusion matrix for the smoothed states excludes the DNE state, since it is absent from both the true and smoothed sequences. Although the Viterbi-based smoothing algorithm increases the ratio of time steps with correct states, the improvement is actually more significant than what the confusion matrices suggest. This is due to the shifting phenomenon discussed earlier. Since this delay pattern is not constant, simply shifting the smoothed sequence by a fixed number of steps does not resolve the issue.



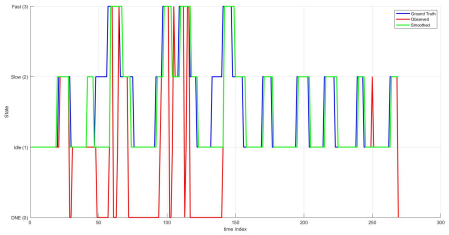
(a) Sequence 01



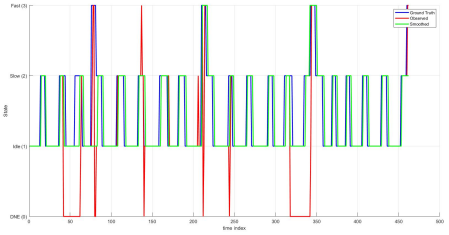
(b) Sequence 02



(c) Sequence 03

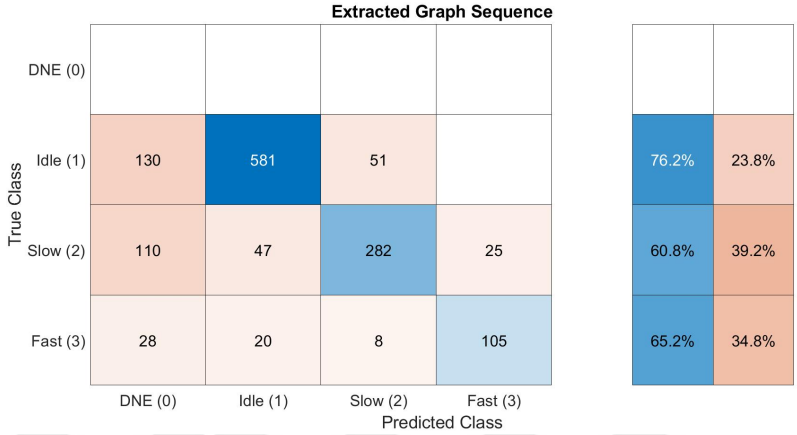


(d) Sequence 04

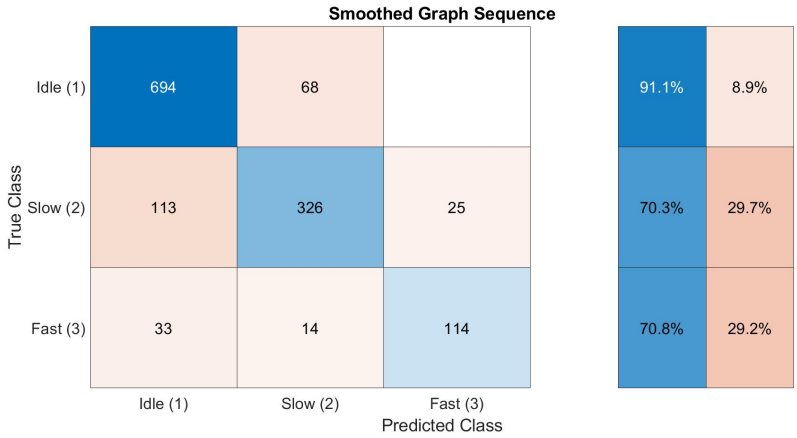


(e) Sequence 05

Figure 4.13: State sequences in CARLA simulations. True, observed, and smoothed sequences are illustrated using blue, red, and green curves, respectively, for five different simulations. Due to the shifts in the observed sequence, caused by the extractor, the smoothed sequence also experiences shifts. Apart from these shifts in transition instances, the smoothed sequence largely aligns with the true sequence. As a result, the smoothing algorithm provides a significantly more accurate representation of the underlying semantic information.



(a) Extracted States



(b) Smoothed States

Figure 4.14: Confusion matrices of observed and smoothed states. Since the DNE state appears in the observed sequence, it is included in the confusion matrix for the extracted states. However, the confusion matrix for the smoothed states is a 3×3 matrix, as the DNE state is not present in either the smoothed or true sequences. The ratio of correctly identified states improves notably with smoothing, especially for the idle/parked state. However, the actual improvement is even greater than what is visible in the confusion matrix, due to the inconsistent delays present in the observed sequence.

The performance of the Baum-Welch Algorithm is also tested using CARLA simulations. Given that the sequences generated in CARLA are relatively short, a modified version of Algorithm 9, as described in [41], is used. This algorithm estimates parameters from multiple observation sequences rather than just a single sequence.

In total, 1128 simulations with varying lengths are generated. The first 1028 are utilized for parameter estimation, while the remaining 100 simulations are smoothed using the estimated parameters.

The learning process is somewhat complicated due to the need to train the model over a state space that includes four states, one of which is the DNE state. During each iteration's M-step, transitions to the DNE state are set to have zero probability. Additionally, the irregular delay pattern present in the observed sequence further complicates the learning process, especially with respect to estimation of the waiting time duration distributions.

The Baum-Welch Algorithm is initialized with uniform waiting time durations for each state between 1 and 20. The state transition and observation matrices are initialized as follows:

$$A = \begin{bmatrix} 0.00 & 0.13 & 0.53 & 0.34 \\ 0.00 & 0.00 & 0.84 & 0.16 \\ 0.00 & 0.20 & 0.00 & 0.80 \\ 0.00 & 0.52 & 0.48 & 0.00 \end{bmatrix}, \tag{4.17}$$

$$B = \begin{bmatrix} 0.00 & 0.50 & 0.08 & 0.42 \\ 0.40 & 0.40 & 0.20 & 0.01 \\ 0.30 & 0.09 & 0.50 & 0.11 \\ 0.00 & 0.25 & 0.25 & 0.50 \end{bmatrix}. \tag{4.18}$$

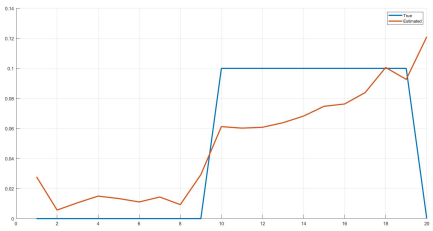
The estimated duration distributions for all three states are given in Figure 4.15. The state transition and observation matrices are estimated using the Baum-Welch Algorithm as follows:

$$\hat{A} = \begin{bmatrix} 0.00 & 0.41 & 0.47 & 0.12 \\ 0.00 & 0.00 & 0.92 & 0.08 \\ 0.00 & 0.89 & 0.00 & 0.11 \\ 0.00 & 0.23 & 0.77 & 0.00 \end{bmatrix}, \tag{4.19}$$

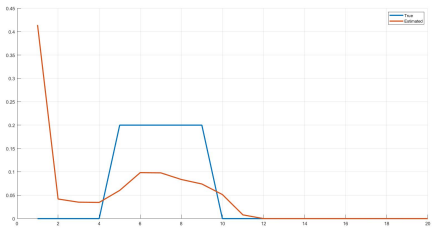
$$\hat{B} = \begin{bmatrix} 0.32 & 0.02 & 0.31 & 0.35 \\ 0.54 & 0.46 & 0.00 & 0.00 \\ 0.09 & 0.02 & 0.88 & 0.01 \\ 0.21 & 0.09 & 0.00 & 0.70 \end{bmatrix}. \tag{4.20}$$

The estimated state transition matrix approximates the true matrix, with notable exceptions. Specifically, the transition probabilities to the idle/parked state from the slow and fast moving states are overestimated. In the case of the fast moving state, this probability should actually be zero. This discrepancy likely occurs because the learning algorithm associates the DNE state with the idle/parked state. As a result, instances where the car disappears while passing under a tree are seen as transitions to the idle state. This interpretation is further supported by the high ratio of idle state predictions for the slow and fast states in the smoothed states' confusion matrix presented in Figure 4.16.

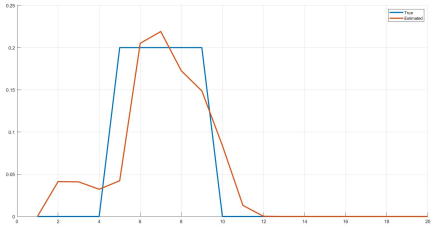
The primary source of error in the CARLA simulations, specifically the mismatch between the correct and smoothed sequences, occurs at the instances of state transition. This is mostly due to delays introduced by the extraction algorithm. The simulation scenarios are deliberately designed to have rapid state transitions. The car moves for short periods of time, stops, and then changes speed frequently, all while



(a) Idle/Parked (S_1)



(b) Slow Moving (S_2)



(c) Fast Moving (S_3)

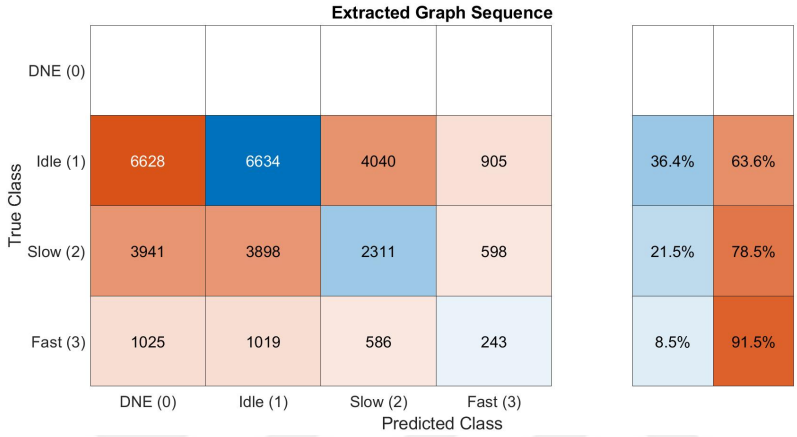
Figure 4.15: Estimated waiting time duration distributions. The blue and red curves represent the true and estimated state duration distributions, respectively.

traveling along a straight road. In practical applications, fewer state transitions with longer state durations are expected. As a result, the impact of errors occurring at state transitions on the discrepancy between the true and smoothed sequences will likely be reduced. This simulation intentionally features quick transitions between states.

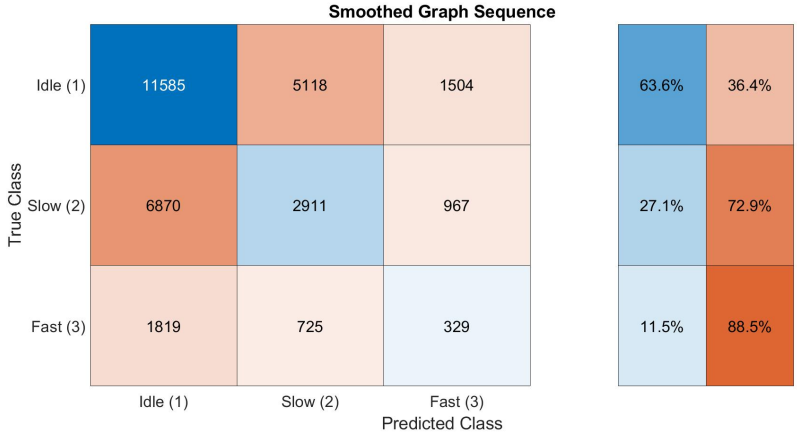
4.3 Chapter Summary

In this chapter, the efficacy of the HSMM algorithm is demonstrated for various tasks including graph smoothing, semantic-graph fusion, and model learning. This is done using MATLAB simulations and video streams generated in the CARLA simulation environment. The MATLAB simulations include a straightforward example that

showcases the improvement offered by the HSMM, specifically ED-HMM, over the more traditionally used HMM. In the CARLA experiments, the focus is on Viterbi-based graph smoothing and model learning. While inconsistent delay patterns in the segmentation-based semantic extractor add complexity to evaluating performance improvement, the comparison of smoothed and extracted graph sequences with the true sequence indicates an increase in the accuracy of the smoothed sequence.



(a) Extracted States



(b) Smoothed States

Figure 4.16: Confusion matrices of observed and smoothed states. The extracted sequences are smoothed with estimated parameters from 1028 other simulations. Slow and fast states are often confused with the idle/parked state. This is likely due to the delay in assigning speed to the car and estimation of DNE states as idle.

Chapter 5

Conclusions and Future Directions

In this thesis, firstly an overview of the Semantic Information Theory (SIT) is presented. Followed by an exploration of the recent advancements in Semantic Communication. Subsequently, the goal-oriented semantic signal processing framework, proposed in [23], is summarized focusing on the graph-based language. This hierarchical graph language allows the representation of the semantic content of the input signal using component and predicate nodes. The integration of goal-oriented filtering allows the framework to efficiently focus on the relevant aspects of the extracted semantic information.

The primary objective of this thesis is to introduce methods for increasing the accuracy of the graphs generated by the framework. Various factors, such as sensory errors or limitations of the semantic extraction algorithm, lead to errors in the extracted semantic-graphs. To address this problem, a stochastic process is defined over the state space of possible graph configurations. In the thesis, the Hidden Semi-Markov Model (HSMM) is adopted to overcome the shortcomings of the Hidden Markov Model (HMM) which was proposed for this purpose in an earlier work [33].

The state space representation for graphs leads to rigorous definition of semantic innovation at the graph level. Modeling the graph evolution with through an HSMM allows for the integration of prior knowledge and the exploitation of continuity of the semantic information through known error correction algorithms, such as the Viterbi Algorithm. In the HSMM framework, algorithms designed for estimating the correct sequence from observed noisy sequences are utilized for smoothing the extracted semantic-graph sequences.

Considering the practical applications, this thesis explores variants of the Viterbi Algorithm with reduced time complexity. These algorithms allow for the smoothing of complex graph sequences under computational constraints, at the cost of some performance decrease. Furthermore, algorithms for semantic information fusion and learning HSMM parameters are also introduced.

The feasibility of these algorithms and the improvement in graph accuracy, which enables accurate detection of graph-level semantic innovation, are demonstrated through MATLAB simulations and video streams from the CARLA environment.

The state space representation for semantic-graph configurations opens up multiple interesting research directions. While this thesis primarily concentrates on enhancing the accuracy of graph-level semantic innovation, the efficient storage and transmission of this information remains crucial. Novel source and channel coding algorithms can be developed for efficient storage and transmission of extracted semantic graphs.

The HSMM approach enables calculation of the ultimate compression limit, under some technical assumptions, which corresponds to the entropy rate of the process. Calculation of the entropy rate and comparing the performance of the employed compression methods with this limit is also an interesting problem.

In the presented results, the model is assumed to be perfectly known, except in

the context of model learning. The potential impact of a mismatch between the true model and the model utilized for smoothing could be further investigated to understand the level of precision required in model learning. Metrics like Kullback-Leibler Divergence between the estimated and true model parameters could be employed to investigate the effects of such a mismatch.

An algorithm for semantic information fusion, developed for scenarios with perfect overlap, is introduced and examined. As expected, this technique outperforms single-sensor methods. For future research, a potential direction is to quantify the degree of improvement by analyzing the correlation between the sensor models and the signals they operate on. Another potential direction for future research could be the comparative analysis of different fusion algorithms, such as estimate and measurement fusion.

Another promising problem is the integration of the smoothing process directly within the graph extraction algorithm. The semantic extractor, potentially a Neural Network, can be designed to leverage the correlation between the consecutive frames of the raw signal along with the prior knowledge during the generation of the semantic-graph.

With the advances in ML/DL techniques and the increasing computational capabilities of IoT devices, building semantically-aware systems is anticipated to become feasible. Such advancements are of growing importance to address the traffic load in large-scale smart sensor networks. Consequently, the methods for enhancing the reliability of extracted semantic information, along with the goal-oriented semantic signal processing framework, are expected to remain relevant in the upcoming years.

Bibliography

[1] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[2] J. Bao, P. Basu, M. Dean, C. Partridge, A. Swami, W. Leland, and J. A. Hendler, "Towards a theory of semantic communication," in *2011 IEEE Network Science Workshop*, pp. 110–117, 2011.

[3] W. Weaver, "Recent contributions to the mathematical theory of communication," *ETC: A Review of General Semantics*, vol. 10, no. 4, pp. 261–281, 1953.

[4] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, "Speech recognition using deep neural networks: A systematic review," *IEEE Access*, vol. 7, pp. 19143–19165, 2019.

[5] A. Shewalkar, D. Nyavanandi, and S. A. Ludwig, "Performance evaluation of deep neural networks applied to speech recognition: Rnn, lstm and gru," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 9, no. 4, pp. 235–245, 3919.

[6] T. Ogunfunmi, R. P. Ramachandran, R. Togneri, Y. Zhao, and X. Xia, "A primer on deep learning architectures and applications in speech processing," *Circuits, Systems, and Signal Processing*, vol. 38, pp. 3406–3432, 2019.

[7] J. Bharadiya, "A comprehensive survey of deep learning techniques natural language processing," *European Journal of Technology*, vol. 7, pp. 58 – 66, May 2023.

[8] B. Alshemali and J. Kalita, "Improving the reliability of deep neural networks in nlp: A review," *Knowledge-Based Systems*, vol. 191, p. 105210, 2020.

[9] L. Wu, Y. Chen, K. Shen, X. Guo, H. Gao, S. Li, J. Pei, and B. Long, "Graph neural networks for natural language processing: A survey," *Foundations and Trends® in Machine Learning*, vol. 16, no. 2, pp. 119–328, 2023.

[10] K. R. Suresh, A. Jarapala, and P. Sudeep, "Image captioning encoder–decoder models using cnn-rnn architectures: A comparative study," *Circuits, Systems, and Signal Processing*, vol. 41, no. 10, pp. 5719–5742, 2022.

[11] Y. Li, "Research and application of deep learning in image recognition," in *2022 IEEE 2nd International Conference on Power, Electronics and Computer Applications (ICPECA)*, pp. 994–999, 2022.

[12] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, "Deep learning for visual understanding: A review," *Neurocomputing*, vol. 187, pp. 27–48, 2016.

[13] G. T. Papadopoulos, A. Briassouli, V. Mezaris, I. Kompatsiaris, and M. G. Strintzis, "Statistical motion information extraction and representation for semantic video analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 10, pp. 1513–1528, 2009.

[14] W. L. Ramos, M. M. Silva, M. F. Campos, and E. R. Nascimento, "Fast-forward video based on semantic extraction," in *2016 IEEE International Conference on Image Processing (ICIP)*, pp. 3334–3338, IEEE, 2016.

[15] B. Zhong, L. Shen, X. Pan, and L. Lei, "Visual attention framework for identifying semantic information from construction monitoring video," *Safety science*, vol. 163, p. 106122, 2023.

[16] Y. Bar-Hillel and R. Carnap, "Semantic information," *The British Journal for the Philosophy of Science*, vol. 4, no. 14, pp. 147–157, 1953.

[17] H. Xie, Z. Qin, G. Y. Li, and B.-H. Juang, "Deep learning enabled semantic communication systems," *IEEE Transactions on Signal Processing*, vol. 69, pp. 2663–2675, 2021.

[18] Z. Weng and Z. Qin, "Semantic communication systems for speech transmission," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 8, pp. 2434–2444, 2021.

[19] J. Johnson, R. Krishna, M. Stark, L.-J. Li, D. Shamma, M. Bernstein, and L. Fei-Fei, "Image retrieval using scene graphs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[20] E. Uysal, O. Kaya, A. Ephremides, J. Gross, M. Codreanu, P. Popovski, M. Assaad, G. Liva, A. Munari, B. Soret, T. Soleymani, and K. H. Johansson, "Semantic communications in networked systems: A data significance perspective," *IEEE Network*, vol. 36, no. 4, pp. 233–240, 2022.

[21] E. Calvanese Strinati and S. Barbarossa, "6g networks: Beyond shannon towards semantic and goal-oriented communications," *Computer Networks*, vol. 190, p. 107930, 2021.

[22] M. Kountouris and N. Pappas, "Semantics-empowered communication for networked intelligent systems," *IEEE Communications Magazine*, vol. 59, no. 6, pp. 96–102, 2021.

[23] M. Kalfa, M. Gok, A. Atalik, B. Tegin, T. M. Duman, and O. Arikan, "Towards goal-oriented semantic signal processing: Applications and future challenges," *Digital Signal Processing*, vol. 119, p. 103134, 2021.

[24] R. Carnap, Y. Bar-Hillel, *et al.*, "An outline of a theory of semantic information," 1952.

[25] R. Carnap, *Logical foundations of probability*, vol. 2. Citeseer, 1962.

[26] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, (New Orleans, Louisiana), pp. 2227–2237, Association for Computational Linguistics, June 2018.

[27] H. Xie and Z. Qin, "A lite distributed semantic communication system for internet of things," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 142–153, 2021.

[28] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

[29] L. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[30] S.-Z. Yu, "Hidden semi-markov models," *Artificial Intelligence*, vol. 174, no. 2, pp. 215–243, 2010. Special Review Issue.

[31] K. P. Murphy, "Hidden semi-markov models (hsmms)," *unpublished notes*, vol. 2, 2002.

[32] S.-Z. Yu, "Chapter 5 - conventional hsmms," in *Hidden Semi-Markov Models* (S.-Z. Yu, ed.), pp. 103–120, Boston: Elsevier, 2016.

[33] M. Kalfa, S. Y. Yetim, A. Atalik, M. Gök, Y. Ge, R. Li, W. Tong, T. M. Duman, and O. Arıkan, "Reliable extraction of semantic information and rate of innovation estimation for graph signals," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 1, pp. 119–140, 2023.

[34] S.-Z. Yu, "Chapter 2 - general hidden semi-markov model," in *Hidden Semi-Markov Models* (S.-Z. Yu, ed.), pp. 27–58, Boston: Elsevier, 2016.

[35] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[36] C. Mitchell, R. Helzerman, L. Jamieson, and M. Harper, "A parallel implementation of a hidden markov model with duration modeling for speech recognition," in *Proceedings of 1993 5th IEEE Symposium on Parallel and Distributed Processing*, pp. 298–306, 1993.

[37] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *IEEE Transactions on Information Theory*, vol. 20, no. 2, pp. 284–287, 1974.

[38] G. Foschini, "A reduced state variant of maximum likelihood sequence detection attaining optimum performance for high signal-to-noise ratios," *IEEE Transactions on Information Theory*, vol. 23, no. 5, pp. 605–609, 1977.

[39] J. Anderson, "Limited search trellis decoding of convolutional codes," *IEEE Transactions on Information Theory*, vol. 35, no. 5, pp. 944–955, 1989.

[40] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains," *The Annals of Mathematical Statistics*, vol. 41, no. 1, pp. 164 – 171, 1970.

[41] S.-Z. Yu, "Chapter 3 - parameter estimation of general hsmm," in *Hidden Semi-Markov Models* (S.-Z. Yu, ed.), pp. 59–83, Boston: Elsevier, 2016.

[42] N. Balakrishnan and C. D. Cutler, *Maximum Likelihood Estimation of Laplace Parameters Based on Type-II Censored Samples*, pp. 145–151. New York, NY: Springer New York, 1996.

[43] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16, 2017.