Baum-Welch Algorithm for Hidden Markov Models Using Algebraic Decision Diagrams

Sebastian Aaholm[1], Lars Emanuel Hansen[1], and Daniel Runge Petersen[1][0009-0004-6529-4342]

Dept. of Computer Science, Aalborg University, Aalborg, Denmark

Abstract. This is a placeholder abstract. The whole template is used in semester projects at Aalborg University (AAU).

1 Introduction

In this section we present some introductory ways to use the tools within LaTeX in general, and this template in particular. For example, this is a citation [1], while this is a multi-citation[1, 2].

The column width of the IEEE template is 3.5 inches, so if you generate your plots with this width or less, the output will be the best. For example, Listing 1.1 contains the code to generate the image in Figure 1 using Python with matplotlib, and exported as pgf (TeX).

1.1 Tables and Figures

Table 1. Example of a pretty, twocolumn table.

	Klasser				
Hændelser	Reservation	Gæst	Borgerforening	Kalender	Betaling
Anmodet	✓	✓	✓		
Godkendt	✓		✓		
Afvist	✓		✓		
Redigeret	✓	✓	✓		
Annulleret	✓	✓	✓		✓
Betalt					✓
Refunderet					✓
Kvitteret		✓	✓		
Registreret	✓			✓	
Påmindet		✓	✓		

```python
import matplotlib.pyplot as plt

plt.rcParams.update({
    "pgf.texsystem": "pdflatex",
    "font.family": "serif",  # use serif/main font for text elements
    "pgf.preamble": "\n".join([
         r"\usepackage[utf8x]{inputenc}",
         r"\usepackage[T1]{fontenc}",
    ]),
})

fig, ax = plt.subplots(figsize=(3.5, 3.5))

ax.plot(range(5))
ax.text(0.5, 3., "serif")
ax.text(0.5, 2., "monospace")
ax.text(2.5, 2., "sans-serif")
ax.set_xlabel(r"µ is not $\mu$")

fig.tight_layout(pad=.5)
fig.savefig("graph.pgf")
```

Listing 1.1. Code to generate the graph.pgf

1.2 Algorithms, Theorems, and Proofs

There are a few different things outside the normal figure and table floats that are very relevant when writing a scientific paper or article. For example, you may wish to typeset theorems as in Theorem 1.

Theorem 1 (Pythagorean theorem). *This is a theorem about right triangles and can be summarized in the next equation*

$$x^2 + y^2 = z^2$$

Or ref like Theorem 1 Similarly, for proofs:

Proof. To prove it by contradiction try and assume that the statement is false, proceed from there and at some point you will arrive to a contradiction.

Note that proofs are not a numbered environment, and as such can't be referenced by default.

2 Preliminaries

This section provides an overview of the theoretical background necessary to understand the rest of the report. We begin by defining the key concepts of a Hidden Markov Model (HMM) and a Markov Decision Process (MDP), which are the two main models used in this report.

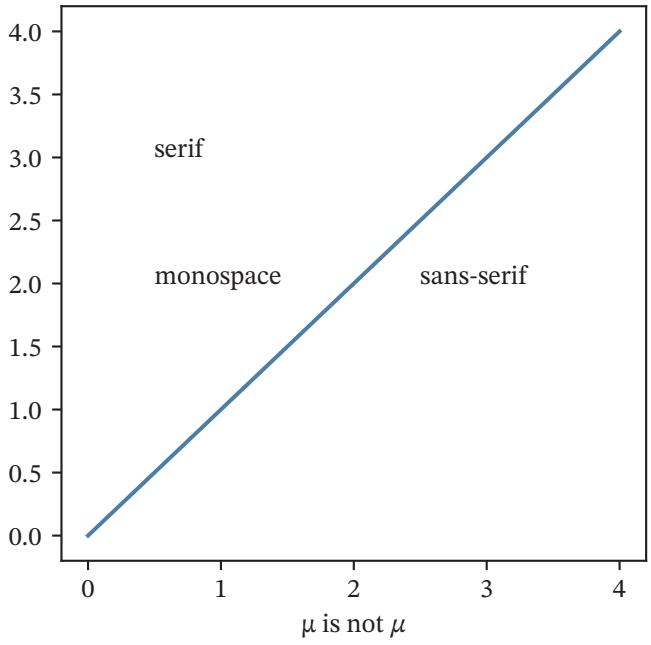

Fig. 1. An example graph drawn using Python's matplotlib library.

```
INSERTION-SORT(A, n)
1  for i ← 2 to n
2      key ← A[i]
3      // Insert A[i] into the sorted subarray A[1 : i − 1].
4      j ← i − 1
5      while j > 0 and A[j] > key
6          A[j + 1] ← A[j]
7          j ← j − 1
8      A[j + 1] ← key
```

Algorithm 1.1. Test

2.1 Hidden Markov Model

HMMs were introduced by Baum and Petrie in 1966 [**NOTFOUND**] and have since been widely used in various fields, such as speech recognition [**NOTFOUND**], bioinformatics [**NOTFOUND**], and finance [**NOTFOUND**].

Definition 1 (Hidden Markov Model). *A Hidden Markov Model (HMM) is a tuple* $\mathcal{M} = (S, \mathcal{L}, \ell, \tau, \pi)$*, where:*

- S *is a finite set of states.*

- $\mathcal{L}$ *is a finite set of labels.*
- $\ell : S \to D(\mathcal{L})$ *is the emission function.*
- $\tau : S \to D(S)$ *is the transition function.*
- $\pi \in D(S)$ *is the initial distribution.*

$D(X)$ denotes the set of probability distributions over a finite set X. The emission function ℓ describes the probability of emitting a label given a state. The transition function τ describes the probability of transitioning from one state to another. The initial distribution π describes the probability of starting in a given state. An HMM is a statistical model that describes a system that evolves over time. The system is assumed to hold the Markov property, meaning that the future state of the system only depends on the current state and not on the past states. The system is also assumed to be unobservable, meaning that the states are hidden and cannot be directly observed. Instead, the system emits observations, which are used to infer the hidden states.

An example of an HMM is a weather model where the hidden state represents the actual weather (sunny, rainy, or cloudy), but we only observe indirect signals, such as whether someone is carrying an umbrella.

2.2 Continuous Time Hidden Markov Model

In the above definition, we have defined a discrete-time HMM, meaning that the system evolves in discrete time steps. In this report, we are interested in continuous-time systems, where the system evolves in continuous time. To model continuous-time systems, we use a Continuous Time Hidden Markov Model (CTHMM), which is an extension of the HMM to continuous time.

Definition 2 (Continuous Time Hidden Markov Model). *A Continuous Time Hidden Markov Model (CTHMM) is a tuple $\mathcal{M} = (S, \mathcal{L}, \ell, \tau, \pi, \lambda)$, where S, $\mathcal{L}$, ℓ, τ, and π are as defined in the HMM definition.*

- $\lambda : S \times S \to \mathbb{R}_{\geq 0}$ *is the rate function.*

The rate function λ determines the transition rate between states, meaning that the time spent in a state before transitioning follows an exponential distribution.

2.3 Markov Decision Process

MDPs were introduced by Bellman in 1957 [**NOTFOUND**] and have since been widely used in various fields, such as robotics [**NOTFOUND**], finance [**NOTFOUND**], and healthcare [**NOTFOUND**].

Definition 3 (Markov Decision Process). *A Markov Decision Process (MDP) is a tuple $\mathcal{M} = (S, A, \mathcal{L}, \ell, \tau, R, \gamma)$, where:*

- S *is a finite set of states.*
- A *is a finite set of actions.*
- $\mathcal{L}$ *is a finite set of labels.*

- $\tau : S \times A \to D(S)$ *is the transition function.*
- $R : S \times A \to \mathbb{R}$ *is the reward function.*
- $\pi \in D(S)$ *is the initial distribution.*

A MDP works by an agent interacting with an environment. Unlike an HMM, the agent takes actions in the environment and receives rewards. An agent is the decision-maker that interacts with the environment by taking actions. The environment is modeled as a MDP, which consists of a set of states S, a set of actions A, a set of labels $\mathcal{L}$. Each state is directly observable, meaning the agent always knows which state it is in. When the agent takes an action in a state, it transitions to a new state according to the transition function τ.

The reward function R describes the reward received when taking an action in a state. The goal of an MDP is to find a policy $\theta : S \to D(A)$ that maximizes the expected cumulative reward. The policy θ describes the probability of taking an action given a state.

Definition 4 (Policy). *A policy θ is a function that maps states to actions, i.e., $\theta : S \to D(A)$.*

An example of an MDP is a robot navigating a grid, where it can choose actions (move up, down, left, or right) and receives rewards based on reaching certain goal locations

3 Methodology

This is a section.

3.1 Decision Diagrams

Binary Decision Diagrams (BDDs) are data structures for efficiently representing and manipulating Boolean functions. They are a compressed representation of truth tables, capturing the logical structure of a function in a graph-based format by eliminating redundancy, reducing memory usage, and improving computational efficiency [3].

A Binary Decision Diagram (BDD) is a directed acyclic graph derived from a decision tree, where each non-terminal node represents a Boolean variable, edges correspond to binary assignments (0 or 1), and terminal nodes store function values (0 or 1). To reduce the size of the decision tree, BDDs exploit redundancy by merging equivalent substructures, resulting in a canonical form (when reduced and ordered) that allows for efficient operations such as function evaluation, equivalence checking, and Boolean operations [3].

BDDs have been widely used in formal verification, model checking, and logic synthesis due to their ability to compactly represent large Boolean functions while maintaining efficient computational properties. However, BDDs suffer from exponential blowup in certain cases, particularly when dealing with functions that lack inherent structure or when representing numerical computations that go beyond Boolean logic.

From BDDs to ADDs Algebraic Decision Diagrams (ADDs) generalize the concept of BDDs by allowing terminal nodes to take values beyond Boolean constants (0 and 1). Instead of restricting values to true/false, Algebraic Decision Diagrams (ADDs) can store arbitrary numerical values, making them useful for representing and manipulating functions over discrete domains [4]. This generalization enables the efficient representation of functions such as cost functions [5], probabilities [6], and other numerical relationships that arise in probabilistic reasoning.

The fundamental structure of an ADD remains similar to a BDD, where a decision tree is compacted by merging redundant substructures. However, instead of performing Boolean operations, ADDs allow for arithmetic operations such as addition and multiplication, making them well-suited for applications like dynamic programming, MDPs, and linear algebraic computations [4].

3.2 Recursive vs. Matrix vs. ADD-based Approaches

When designing algorithms for solving complex problems, different approaches can be taken to optimize computational efficiency. Three common strategies are recursive, matrix-based, and ADD-based approaches, each with distinct advantages and limitations.

- **Recursive Approach:** Conceptually simple, recursion follows a divide-and-conquer strategy but can suffer from exponential time complexity due to redundant computations of overlapping subproblems [7, Chapter 4].
- **Matrix Representation:** Reformulating algorithms using matrix operations leverages algebraic properties for parallel computation and efficient processing. However, matrix-based methods can be memory-intensive, particularly for sparse or structured data [7, Chapter 4, 15 & 28].
- **ADD-based Approach:** ADDs provide a compact representation that eliminates redundancy in recursive computations. By reusing previously computed substructures, they improve efficiency and reduce memory overhead [4]. Compared to matrices, ADDs can offer a more space-efficient alternative for structured data while extending BDD techniques to handle both Boolean and numerical computations.

In this work we explore the benefits of ADD-based approaches for solving complex problems, focusing on parameter estimation in Discrete Time Markov Chains (DTMCs) and Continuous Time Markov Chains (CTMCs). We compare the performance of ADD-based algorithms against recursive-based implementations, highlighting the advantages of using ADDs for efficient computation and memory management.

3.3 CuDD

Colorado University Decision Diagram (CuDD) is a library for implementing and manipulating BDDs and ADDs developed at the University of Colorado. The Colorado University Decision Diagram (CuDD) library [8] is a powerful tool for implementing and manipulating decision diagrams, including BDDs and ADDs.

Implemented in C, the CuDD library ensures high-performance execution and can be seamlessly integrated into C++ programs, which we utilize in this paper. By leveraging the CuDD library, we demonstrate the benefits of ADD-based approaches for solving parameter estimation problems in DTMCs and CTMCs.

In this project, we use the CuDD library to store ADDs and perform operations on them. Its optimized algorithms and efficient memory management enable symbolic handling of large and complex matrices, significantly improving performance compared to traditional methods.

4 Experiments

In this section, we describe the experiments conducted to evaluate the performance of the symbolic implementation of the Baum-Welch algorithm in the CuPAAL library by comparing it to the recursive implementation in Jajapy. The evaluation is based on two key aspects: execution time and accuracy.

We conduct two experiments:

- **Performance Comparison** - Measuring runtime and accuracy across different models.
- **Scalability Analysis** - Evaluating performance as the number of states increases.

These experiments aim to answer the following research questions:

- **Question 1**: How does the symbolic implementation of the Baum-Welch algorithm in CuPAAL compare to the recursive implementation in Jajapy in terms of runtime and accuracy?
- **Question 2**: How does the performance of the CuPAAL implementation scale with the number of states in the model?

4.1 Experimental Setup

All experiments are conducted using a set of DTMCs and CTMCs obtained from publicly available benchmarks[1].

Each experiment is run ten times, and results are reported as the average runtime for the full run, average runtime for each iteration, the average number of iterations, log-likelihood for each iteration, and average relative parameter error for each iteration. We stop the experiments when we reach a convergence threshold of 0.05, which was the default value in the Jajapy implementation or after 4 hours of runtime. The training data is randomly generated based on these models, consisting of 30 observation sequences of length 10 for each model.

[1] The models are available at `https://qcomp.org/benchmarks/`. The source files describe the parameters and what is observable.

Experiment 1: Performance Comparison of Forward-Backward Implementations The first experiment is based on the ideas from the experiment conducted in [9]. The models used in this experiment are shown in Table 2 and Table 3. All models has a fixed number of parameters of, and the number of states varies.

The experiment evaluates the efficiency and accuracy of the symbolic approach (CuPAAL) versus the recursive approach (Jajapy). We measure:

- **Runtime Efficiency** - The average time per run.
- **Convergence Speed** - The average number of iterations required.
- **Accuracy** - Measured using log-likelihood and average relative parameter error.

The **relative parameter error** is calculated as follows $\frac{|r-e|}{e}$, where r is the real value and e is the expected value.

Log-likelihood: Measures how well the learned model explains the observed data. Given an observation sequence O and a model M, the log-likelihood is calculated as:

$\log P(O \mid M) = \sum_{t=1}^{T} \log P(O_t \mid M)$ where $P(O_t|M)$ is the probability of observing O_t given the model.

Following the approach in [10], initial parameters are randomly sampled from the range [0.00025, 0.0025]. The implementations used are:

1. The original Jajapy implementation.
2. The symbolic CuPAAL implementation.

We expect the CuPAAL implementation to have the same accuracy as the Jajapy implementation and have the same number of iterations. However, we expect the CuPAAL implementation to be faster due to the symbolic approach, which can avoid redundant calculations.

The results are presented for DTMCs in Table 4, Table 5, and Table 6.

For CTMCs, the results are shown in Table 7, Table 8, and Table 9. All tables show the average time per run for each implementation, average number of iterations and log-likelihood and relative parameter error.

Table 2. DTMC models

Name	Number of States
Leader_sync	274
Brp	886
Crowds	1145

4.2 Scalability Experiment

The primary objective of this experiment is to evaluate the scalability of the proposed symbolic implementation of the Baum-Welch algorithm in comparison to the recursive implementation in Jajapy. Specifically, we aim to measure the time required to learn DTMCs and CTMCs over the number of states. We measure:

Table 3. CTMC models

Name	Number of States
Mapk	118
Cluster	820
Embedded	3480

Table 4. Leader_sync results

Implementation	Iter	Time(s)	Avg δ	Log-likelihood
Jajapy	0	0	0	0
CuPAAL	0	0	0	0

Table 5. Brp results

implementation	Iter	Time(s)	avg δ	log-likelihood
Jajapy	0	0	0	0
CuPAAL	0	0	0	0

Table 6. Crowds results

implementation	Iter	Time(s)	avg δ	log-likelihood
Jajapy	0	0	0	0
CuPAAL	0	0	0	0

Table 7. Mapk results

implementation	Iter	Time(s)	avg δ	log-likelihood
Jajapy	0	0	0	0
CuPAAL	0	0	0	0

Table 8. Cluster results

implementation	Iter	Time(s)	avg δ	log-likelihood
Jajapy	0	0	0	0
CuPAAL	0	0	0	0

Table 9. Embedded results

implementation	Iter	Time(s)	avg δ	log-likelihood
Jajapy	0	0	0	0
CuPAAL	0	0	0	0

– **Runtime efficiency** - The average time per run.

The experiment is run on the polling model for CTMCs, where the number of states is increased from 36 to 1334, and the initial parameters are resampled in the range [0.00025, 0.0025]. For DTMCs, we use the leader_sync model, where the number of states is increased from 26 to 1050, and the initial parameters are resampled in the range [0.00025, 0.0025].

The two models were chosen because they are representative of the models used in the performance comparison experiment, and both models can be scaled to a large number of states.

This experiment is conducted to assess scalability, a key factor in Baum-Welch algorithm performance. It provides insights into the efficiency of the symbolic approach for both DTMCs and CTMCs

The scalability is determined by the time taken to learn the model as the number of states increases.

While we expect CuPAAL to have lower runtime and improved scalability, we also expect the accuracy to be consistent with the Jajapy implementation.

The results are shown in Figure 2.

missing picture

Fig. 2. Scalability results for the tandem model

5 Improvements

This section outlines the improvements gained by transitioning from the recursive implementation in Jajapy to the symbolic approach in Cupaal.

As discussed in (Ref to previous section talking about Jajapy), Jajapy uses a recursive implementation of the Baum-Welch algorithm to learn HMMs. In contrast, Cupaal implements the Baum-Welch algorithm using ADDs. By leveraging ADDs, Cupaal demonstrates significant improvements over the recursive approach used in Jajapy. The discussion of these improvements is based on the experimental results presented in Section 4.

5.1 Run Time

One of the most notable advantages of Cupaal over Jajapy is the reduction in run time, particularly for models with a large number of states.

The use of ADDs minimizes redundant computations by merging identical values within the structure. This optimization significantly reduces the computational needs, compared to a recursive implementation. As a result, the run time gap between Cupaal and Jajapy increases as the number of states grows, making Cupaal a more scalable solution for large HMMs.

The number of iterations of Cupaal for each model is also slightly reduced compared to Jajapy.

This advantage is particularly beneficial in scenarios where handling large probability matrices would otherwise lead to excessive computational costs.

5.2 Accuracy

While run time is a key advantage of Cupaal, it is equally important to assess whether these performance gains come at the cost of accuracy. Since both approaches implement the Baum-Welch algorithm, they are expected to converge to similar model parameters when learning HMMs.

As shown in Section 4, Cupaal achieves accuracy comparable to Jajapy across various models. These results can be seen in the values of avg delta and the log-likelihood, where the closer the value is to 0 the better. Displaying that a symbolic implementation does not introduce significant numerical errors, ensuring that the learned transition and emission probabilities remain consistent with those obtained using the recursive approach.

Furthermore, by eliminating redundant calculations, Cupaal may reduce floating-point errors that typically accumulate in recursive implementations. Importantly, Cupaal maintains accuracy even as the number of states increases, showcasing that its efficiency improvements do not compromise learning quality. This makes it particularly well-suited for handling large-scale HMMs.

5.3 Implementation

The implementation of Cupaal has been done in c++, compared to Jajapy which is implemented in Python. This could also be a factor aiding the performance improvement of Cupaal, as C++ is generally faster at computation compared to Python. This choice of implementation not only improves speed but also ensures that Cupaal can efficiently handle large models that would be infeasible in Python.

5.4 Final improvement overview

The improvements introduced by Cupaal stem from multiple factors: the adoption of ADDs, optimized run time and a high-performance C++ implementation. These enhancements make Cupaal a powerful alternative to recursive approaches like Jajapy, particularly when working with large, redundant, and complex HMMs.

Acronyms

AAU Aalborg University. 1
ADD Algebraic Decision Diagram. 5–7, 10, 11

BDD Binary Decision Diagram. 5, 6

CTHMM Continuous Time Hidden Markov Model. 4
CTMC Continuous Time Markov Chain. 6–10
CuDD Colorado University Decision Diagram. 6, 7

DTMC Discrete Time Markov Chain. 6–10

HMM Hidden Markov Model. 3, 4, 10, 11

MDP Markov Decision Process. 3–6

References

[1] M. Goossens, F. Mittelbach, and A. Samarin, *The LaTeX Companion*. Reading, Massachusetts: Addison-Wesley, 1993.

[2] G. D. Greenwade, "The Comprehensive Tex Archive Network (CTAN)," *TUGBoat*, vol. 14, no. 3, pp. 342–351, 1993.

[3] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 677–691, 1986.

[4] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebric decision diagrams and their applications," *Formal methods in system design*, vol. 10, pp. 171–206, 1997.

[5] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic symbolic model checking with prism: A hybrid approach," *International journal on software tools for technology transfer*, vol. 6, no. 2, pp. 128–142, 2004.

[6] C. Baier, E. M. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, and M. Ryan, "Symbolic model checking for probabilistic processes," in *Automata, Languages and Programming: 24th International Colloquium, ICALP'97 Bologna, Italy, July 7–11, 1997 Proceedings 24*, Springer, 1997, pp. 430–440.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.

[8] F. Somenzi, "Cudd: Cu decision diagram package," *Public Software, University of Colorado*, 1997.

[9] R. Reynouard et al., "On learning stochastic models: From theory to practice," 2024.

[10] G. Bacci, A. Ingólfsdóttir, K. G. Larsen, and R. Reynouard, "Mm algorithms to estimate parameters in continuous-time markov chains," 2023. arXiv: 2302.08588 [cs.LG].

A Cheatsheet

If something is represented with a greek letter, it is something we calculate.

Symbol	Meaning
$\mathbb{R}$	Real numbers
$\mathbb{Q}$	Rational numbers
$\mathbb{N}$	Natural numbers
$s \in S$	States
$l \in L$	Labels
$a \in A$	Actions
$\mathcal{M}$	Markov Model
$\mathcal{H}$	Hypothesis
$o \in O \in \mathcal{O}$	Observations
π	Initial distribution
τ	Transition function
ι or ω	Emission function
α	Forward probabilities
β	Backward probabilities
γ	State probabilities
ξ	Transition probabilities
$\lambda = (\pi, \tau, \omega)$	Model Parameters
ϕ or ψ	Scheduler
μ	Mean
σ	Standard deviation
$\theta = (\mu, \sigma^2)$	Parameters of a distribution
$P(\mathcal{O}; \lambda)$	Probability of $\mathcal{O}$ given λ
$\ell(\lambda; \mathcal{O})$	Log likelihood of λ under $\mathcal{O}$