Symbolic Parameter Estimation of Continuous-Time Markov Chains

Sebastian Aaholm*, Lars Emanuel Hansen†, Daniel Runge Petersen‡,

Abstract—This is a placeholder abstract. The whole template is used in semester projects at AAU.

Index Terms—Formal Verification, Parameter Estimation, Decision Diagram

1 Introduction

This paper is about improving the runtime of jajapy - a tool for estimating parameters in parametric models [1].

Markov Chain (MC) - A chain of events described as a sequence of events without knowledge of prior. Hidden Markov Model (HMM) - A markov chain with emission probabilities. Markov Decision Process (MDP) - A markov chain with actions that influence the transitions. Continuous Time Markov Chain - A markov chain with traces that have dwell times as well as label emissions. Baum-Welch algorithm (BW) - Expectation-Maximization algorithm for finding the parameters of a Hidden Markov Model. Algebraic Decision Diagram (ADD) - A data structure of states and binary decisions, also called a Multi-Terminal Binary Decision Diagram (MTBDD).

2 Definitions

Definition 1 (Markov Chain). A Markov chain 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.
- $\ell : S \to \mathcal{L}$ is a labeling function, which assigns a label to each state.
- $\tau : S \to \mathcal{D}(S)$ is a transition function. The model moves from state s to state s' with probability $\tau(s, s')$.
- π: is the initial distribution, the model starts in state s with probability $\pi(s)$.

Intuitively, a Markov chain is a model that starts in a state s with probability $\pi(s)$, and then transitions to a new state s' with probability $\tau(s, s')$. The model continues to transition between states according to the transition function.

Definition 2 (Hidden Markov Model). A Hidden Markov Model (HMM) is a tuple $\mathcal{M} = (S, \mathcal{L}, \ell, \tau, \pi)$, where $S, \mathcal{L}, \tau, \pi$ are defined as above, and:

- $\ell : S \to D(\mathcal{L})$ is the emission function. The model emits a label l in state s with probability $\ell(s, l)$.

- *All authors are with the Dept. of Computer Science, Aalborg University, Aalborg, Denmark*
- *E-mails: *saahol20, †leha20, ‡dpet20 @student.aau.dk*

Intuitively, an HMM is a model that starts in a state s with probability $\pi(s)$, then emits a label l with probability $\ell(s, l)$, and transitions to a new state s' with probability $\tau(s, s')$. The model continues to emit labels and transition between states according to the emission and transition functions.

Definition 3 (Markov Decision Process). A Markov Decision Process (MDP) is a tuple $\mathcal{M} = (S, \mathcal{L}, \ell, A, \{\tau_a\}_{a \in A}, \pi)$ where $S, \mathcal{L}, \ell, \pi$ are defined as above, and:

- A is a finite nonempty set of actions.
- $\tau_a : S \to \mathcal{D}(S)$ is a transition function for each action $a \in A$. The model moves from state s to state s' with probability $\tau_a(s, s')$ when action a is taken.

Intuitively, an MDP is a model that starts in a state s with probability $\pi(s)$, then emits a label $\ell(s)$ and, it can recieve an action $a \in A$ and transition to a new state s' with probability $\tau_a(s, s')$.

2.1 Continuous-Time

In the previous definitions, the models are discrete-time models, where time advances in fixed, regular steps. For example, in a discrete-time Markov chain, the system transitions between states at each step or tick of a clock, and the probability of moving from one state to another is governed by the transition function $\tau(s, s')$. This means that transitions can only happen at specific time intervals (e.g., after every second, every minute, etc.).

In contrast, continuous-time models allow transitions to occur at any time, rather than at fixed intervals. The time between transitions is variable and follows a continuous distribution. This introduces the concept of transition rates rather than discrete transition probabilities.

Definition 4 (Continuous-Time Markov Chain). A Continuous-Time Markov Chain (CTMC) is a tuple $\mathcal{M} = (S, \mathcal{L}, \ell, R, \pi)$, where $S, \mathcal{L}, \ell, \pi$ are defined as above, and:

- $R : S \times S \to \mathbb{R}_{\geq 0}$ is the rate function. The model transitions from state s to state s' with rate $R(s, s')$.

For two states s and s', $R(s, s')$ gives the rate at which the system moves from state s to state s'. A higher rate means a faster transition.

A Continuous-Time Markov Chain (CTMC) is a type of Markov model where the time between transitions is not fixed but is governed by exponential distributions. If there are more then one outgoing transition from a state, we get race-conditions, the first transition to occur is the one that will be taken. The time spent in a state before transitioning to a new state is called $dwell - time$. This is exponentially distributed with a rate $E(s) = \sum_{s' \in S} R(s, s')$. The probability of transitioning from state s to state s' is $R(s, s')/E(s)$, the time spent in s is independent from the properbility of transitioning to s'.

2.2 Matrix Representation

The transition function τ can be represented as a matrix, where each element $\tau(s, s')$ is the probability of transitioning from state s to state s'. The matrix representation of τ is called the transition matrix. The transition matrix is a square matrix with dimensions $|S| \times |S|$, where $|S|$ is the number of states in the model. The transition matrix is a stochastic matrix, meaning that the sum of each row is equal to 1, meaning all the probabilities of transitioning from state s to all other states sum to 1.

If we take an example of a model with two states $S = \{s_1, s_2\}$, the transition matrix τ is defined as:

$$\tau = \begin{bmatrix} \tau(s_1, s_1) & \tau(s_1, s_2) \\ \tau(s_2, s_1) & \tau(s_2, s_2) \end{bmatrix} \tag{1}$$

We can give an example of a transition matrix for a model with two states, where the model transitions from state s_1 to state s_2 with probability 0.4 and transitions from state s_2 to state s_1 with probability 0.5:

$$\tau = \begin{bmatrix} 0.6 & 0.4 \\ 0.5 & 0.5 \end{bmatrix} \tag{2}$$

The initial distribution π is a vector that represents the probability of starting in each state. The initial distribution is a stochastic vector, meaning that the sum of all probabilities is equal to 1. The initial distribution π is a vector with dimensions $|S|$, where $|S|$ is the number of states in the model. Each element $\pi(s)$ is the probability of starting in state s.

$$\pi = \begin{bmatrix} 0.6 \\ 0.5 \end{bmatrix} \tag{3}$$

The labeling function ℓ can be represented as a matrix, where each element $\ell(s, l)$ is the probability of emitting label l in state s. The matrix representation of ℓ is called the emission matrix. The emission matrix is a matrix with dimensions $|S| \times |\mathcal{L}|$, where $|\mathcal{L}|$ is the number of labels in the model. The emission matrix is a stochastic matrix, meaning that the sum of each row is equal to 1, meaning all the probabilities of emitting a label in state s sum to 1.

If we take an example of a model with two states $S = \{s_1, s_2\}$ and two labels $\mathcal{L} = \{l_1, l_2\}$, the emission matrix ℓ is defined as:

$$\ell = \begin{bmatrix} \ell(s_1, l_1) & \ell(s_1, l_2) \\ \ell(s_2, l_1) & \ell(s_2, l_2) \end{bmatrix} \tag{4}$$

We can give an example of an emission matrix for a model with two states and two labels, where the model emits label l_1 in state s_1 with probability 0.7 and emits label l_2 in state s_2 with probability 0.6:

$$\ell = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} \tag{5}$$

The rate function R can be represented as a matrix, where each element $R(s, s')$ is the rate of transitioning from state s to state s'. The matrix representation of R is called the rate matrix. The rate matrix is a square matrix with dimensions $|S| \times |S|$, where $|S|$ is the number of states in the model. The rate matrix is a non-negative matrix, meaning that all elements are greater than or equal to 0.

$$R = \begin{bmatrix} R(s_1, s_1) & R(s_1, s_2) \\ R(s_2, s_1) & R(s_2, s_2) \end{bmatrix} \tag{6}$$

If we take an example of a model with two states $S = \{s_1, s_2\}$, the rate matrix R is defined as:

$$R = \begin{bmatrix} 0.5 & 0.3 \\ 0.2 & 0.4 \end{bmatrix} \tag{7}$$

3 HMM Example

3.1 Setup

We have a simple HMM with, two hidden states S_1 and S_2, two observation symbols: O_1 and O_2 and an observation sequence $O = \{O_1, O_2, O_1\}$.

The HMM parameters are:

Transition matrix A (probability of moving from one state to another):

$$A = \begin{bmatrix} 0.6 & 0.4 \\ 0.5 & 0.5 \end{bmatrix}$$

Emission matrix B (probability of emitting observation given a state):

$$B = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}$$

Initial state probability vector π (probability of starting in each state):

$$\pi = \begin{bmatrix} 0.8 & 0.2 \end{bmatrix}$$

3.2 Expectation step

In the expectation step we calculate α and β.

3.2.1 *Forward step* α

We first compute the forward probabilities $\alpha_t(i)$, which represent the probability of being in state i at time t after observing the first t symbols.

3.2.1.1 Initialization at $(t = 1)$:

$$\alpha_1 = \pi \circ B_{y1}$$

Where B_{y1} is the first column of the emission matrix, corresponding to observation O_1

(i.e., $B_{y1} = \begin{bmatrix} 0.7 \\ 0.4 \end{bmatrix}$) and $\circ$ represents the Hadamard product.

So, we get:

$$\alpha_1 = \begin{bmatrix} 0.8 \\ 0.2 \end{bmatrix} \circ \begin{bmatrix} 0.7 \\ 0.4 \end{bmatrix} = \begin{bmatrix} 0.56 \\ 0.08 \end{bmatrix}$$

3.2.1.2 Induction (for $t = 2, 3, \ldots, T$): For subsequent timesteps, we compute:

$$\alpha_{t+1} = B_{yt+1} \circ (A^T \alpha_t)$$

Where A^T is the transpose of the transition matrix. Let's apply this to compute the forward probabilities for $t = 2$ and $t = 3$:

At $t = 2$ (observation O_2):

$$\alpha_2 = B_{y2} \circ (A^T \alpha_1)$$

We have:

$$B(y2) = \begin{bmatrix} 0.3 \\ 0.6 \end{bmatrix}$$

and

$$A^T = \begin{bmatrix} 0.6 & 0.5 \\ 0.4 & 0.5 \end{bmatrix}$$

We get:

$$\alpha_2 = \begin{bmatrix} 0.3 \\ 0.6 \end{bmatrix} \circ \left(\begin{bmatrix} 0.6 & 0.5 \\ 0.4 & 0.5 \end{bmatrix} \cdot \begin{bmatrix} 0.56 \\ 0.08 \end{bmatrix} \right) = \begin{bmatrix} 0.1128 \\ 0.1584 \end{bmatrix}$$

At $t = 3$ (observation O_1):

$$\alpha_2 = B_{y1} \circ (A^T \alpha_2)$$

We get:

$$\alpha_3 = \begin{bmatrix} 0.7 \\ 0.4 \end{bmatrix} \circ \left(\begin{bmatrix} 0.6 & 0.5 \\ 0.4 & 0.5 \end{bmatrix} \cdot \begin{bmatrix} 0.1 \\ 0.1584 \end{bmatrix} \right) = \begin{bmatrix} 0.102816 \\ 0.049728 \end{bmatrix}$$

3.2.2 Backward step β

The backward probabilities $\beta_t(i)$ represent the probability of observing the rest of the sequence starting from time $t + 1$, given that the system is in state i at time t.

Initialization (at $t = T = 3$)

$$\beta_T = \mathbf{1} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

3.2.2.1 Induction (for $t = T-1, T-2, \ldots, 1$): For earlier timesteps, we compute:

$$\beta_t = A(\beta_{t+1} \circ B_{yt+1})$$

At $t = 2$ (observation O_1):

$$\beta_2 = A(\beta_3 \circ B_{y1})$$

$$B_{y1} = \begin{bmatrix} 0.7 \\ 0.4 \end{bmatrix}, \quad \beta_3 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

We get:

$$\beta_2 = \begin{bmatrix} 0.6 & 0.4 \\ 0.5 & 0.5 \end{bmatrix} \cdot \left(\begin{bmatrix} 1 \\ 1 \end{bmatrix} \circ \begin{bmatrix} 0.7 \\ 0.4 \end{bmatrix} \right)$$

$$\beta_2 = \begin{bmatrix} 0.6 & 0.4 \\ 0.5 & 0.5 \end{bmatrix} \cdot \begin{bmatrix} 0.7 \\ 0.4 \end{bmatrix} = \begin{bmatrix} 0.58 \\ 0.55 \end{bmatrix}$$

At $t = 1$ (observation O_2):

$$\beta_1 = A(\beta_2 \circ B_{y2})$$

We have:

$$B_{y2} = \begin{bmatrix} 0.3 \\ 0.6 \end{bmatrix}, \quad \beta_2 = \begin{bmatrix} 0.58 \\ 0.55 \end{bmatrix}$$

$$\beta_1 = \begin{bmatrix} 0.6 & 0.4 \\ 0.5 & 0.5 \end{bmatrix} \cdot \left(\begin{bmatrix} 0.3 \\ 0.6 \end{bmatrix} \circ \begin{bmatrix} 0.58 \\ 0.55 \end{bmatrix} \right)$$

$$\beta_1 = \begin{bmatrix} 0.6 & 0.4 \\ 0.5 & 0.5 \end{bmatrix} \cdot \begin{bmatrix} 0.174 \\ 0.33 \end{bmatrix} = \begin{bmatrix} 0.2364 \\ 0.252 \end{bmatrix}$$

3.3 Step 3: Compute γ and ξ

3.3.1 Compute γ

We can compute γ by

$$\gamma_t = (\mathbb{1}^T \cdot \alpha_T)^{-1} \cdot (\alpha_t \circ \beta_t)$$

$$\alpha_T = \begin{bmatrix} 0.089628 \\ 0.053328 \end{bmatrix}$$

$$\mathbb{1}^T \cdot \alpha_T = 0.089628 + 0.053328 = 0.152544$$

This is the total probability of oberserving our sequence $O = \{O_1, O_2, O_1\}$

Now we can compute γ_t for each time stamp.

At t=1: We have

$$\alpha_1 = \begin{bmatrix} 0.56 \\ 0.08 \end{bmatrix}, \quad \beta_1 = \begin{bmatrix} 0.2364 \\ 0.252 \end{bmatrix}$$

We take the Hadamard product of this.

$$\alpha_1 \circ \beta_1 = \begin{bmatrix} 0.56 \cdot 0.2364 \\ 0.08 \cdot 0.252 \end{bmatrix} = \begin{bmatrix} 0.132384 \\ 0.02016 \end{bmatrix}$$

We normalize the first part and take the scalar product.

$$\gamma_1 = \frac{1}{0.152544} \cdot \begin{bmatrix} 0.132384 \\ 0.02016 \end{bmatrix} = \begin{bmatrix} 0.8678414 \\ 0.1321589 \end{bmatrix}$$

At t = 2:

We have:

$$\alpha_2 = \begin{bmatrix} 0.1074 \\ 0.1584 \end{bmatrix}, \quad \beta_2 = \begin{bmatrix} 0.58 \\ 0.55 \end{bmatrix}$$

The Hadamard product is:

$$\alpha_2 \circ \beta_2 = \begin{bmatrix} 0.1074 \cdot 0.58 \\ 0.1584 \cdot 0.55 \end{bmatrix} = \begin{bmatrix} 0.062292 \\ 0.08712 \end{bmatrix}$$

We normalize the first part and take the scalar product.

$$\gamma_2 = \frac{1}{0.152544} \cdot \begin{bmatrix} 0.062292 \\ 0.08712 \end{bmatrix} = \begin{bmatrix} 0.42888609 \\ 0.57111391 \end{bmatrix}$$

At t = 3

We have:

$$\alpha_3 = \begin{bmatrix} 0.089628 \\ 0.053328 \end{bmatrix}, \quad \beta_3 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

The Hadamard product is:

$$\alpha_3 \circ \beta_3 = \begin{bmatrix} 0.089628 \\ 0.053328 \end{bmatrix}$$

We normalize the first part and take the scalar product.

$$\gamma_3 = \frac{1}{0.152544} \cdot \begin{bmatrix} 0.089628 \\ 0.053328 \end{bmatrix} = \begin{bmatrix} 0.67400881 \\ 0.32599119 \end{bmatrix}$$

3.3.2 Calculating ξ

We calculate ξ by

$$\xi_t = ((\mathbb{1}^T\alpha_T)^{-1} \cdot A) \circ (\alpha_t \otimes (\beta_{t+1} \circ B_{yt+1})^T)$$

We start by calculating $((\mathbb{1}^T\alpha_T)^{-1} \cdot A)$: From before, we have

$$(\mathbb{1}^T\alpha_T)^{-1} = \frac{1}{0.152544} = 6.996$$

We have:

$$A = \begin{bmatrix} 0.6 & 0.4 \\ 0.5 & 0.5 \end{bmatrix}$$

We get:

$$8.996 \cdot A = \begin{bmatrix} 6.996 \cdot 0.6 & 6.996 \cdot 0.4 \\ 6.996 \cdot 0.5 & 6.996 \cdot 0.5 \end{bmatrix} = \begin{bmatrix} 4.198 & 2.798 \\ 3.498 & 3.498 \end{bmatrix}$$

We can now calculate $\alpha_1 \otimes (\beta_2 \circ B_{y2})^T$. We have :

$$\alpha_1 = \begin{bmatrix} 0.56 \\ 0.08 \end{bmatrix}, \quad \beta_2 = \begin{bmatrix} 0.58 \\ 0.55 \end{bmatrix}, \quad B_{y2} = \begin{bmatrix} 0.3 \\ 0.6 \end{bmatrix}$$

We calculate $\beta_2 \circ B_{y2}$:

$$\beta_2 \circ B_{y2} = \begin{bmatrix} 0.58 \\ 0.55 \end{bmatrix} \circ \begin{bmatrix} 0.3 \\ 0.6 \end{bmatrix} = \begin{bmatrix} 0.174 \\ 0.33 \end{bmatrix}$$

Outer product:

$$\alpha_1 \otimes (\beta_2 \circ B_{y2})^T = \begin{bmatrix} 0.56 \\ 0.08 \end{bmatrix} \otimes \begin{bmatrix} 0.174 & 0.33 \end{bmatrix}$$

$$= \begin{bmatrix} 0.09744 & 0.1848 \\ 0.01392 & 0.0264 \end{bmatrix}$$

We can now calculate ξ_1

$$\xi_1 = \begin{bmatrix} 4.198 & 2.798 \\ 3.498 & 3.498 \end{bmatrix} \circ \begin{bmatrix} 0.09744 & 0.1848 \\ 0.01392 & 0.0264 \end{bmatrix}$$

$$\xi_1 = \begin{bmatrix} 0.38325991 & 0.03650094 \\ 0.60572687 & 0.08653241 \end{bmatrix}$$

At t=2:

We have:

$$B_{y1} = \begin{bmatrix} 0.7 \\ 0.4 \end{bmatrix}, \quad \alpha_2 = \begin{bmatrix} 0.1074 \\ 0.1584 \end{bmatrix}, \quad \beta_3 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Hadamard product for $\beta_3 \circ B_{y1}$

$$\beta_3 \circ B_{y1} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \circ \begin{bmatrix} 0.7 \\ 0.4 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 0.4 \end{bmatrix}$$

Outer product:

$$\alpha_2 \otimes \begin{bmatrix} 0.7 & 0.4 \end{bmatrix} = \begin{bmatrix} 0.07518 & 0.04296 \\ 0.11088 & 0.06336 \end{bmatrix}$$

We can now calculate ξ_2:

$$\xi_2 = \begin{bmatrix} 4.198 & 2.798 \\ 3.498 & 3.498 \end{bmatrix} \circ \begin{bmatrix} 0.07518 & 0.04296 \\ 0.11088 & 0.06336 \end{bmatrix}$$

$$\xi_2 = \begin{bmatrix} 0.07341938 & 0.06873304 \\ 0.03726872 & 0.0523348 \end{bmatrix}$$

At t=3:

We have:

$$B_{y1} = \begin{bmatrix} 0.7 \\ 0.4 \end{bmatrix}, \quad \alpha_3 = \begin{bmatrix} 0.089628 \\ 0.053328 \end{bmatrix}, \quad \beta_3 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Hadamard product for $\beta_3 \circ B_{y1}$

$$\beta_3 \circ B_{y1} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \circ \begin{bmatrix} 0.7 \\ 0.4 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 0.4 \end{bmatrix}$$

Outer product:

$$\alpha_3 \otimes \begin{bmatrix} 0.7 & 0.4 \end{bmatrix} = \begin{bmatrix} 0.062740 & 0.035852 \\ 0.037329 & 0.021331 \end{bmatrix}$$

We can now calculate ξ_3:

$$\xi_2 = \begin{bmatrix} 4.198 & 2.798 \\ 3.498 & 3.498 \end{bmatrix} \circ \begin{bmatrix} 0.062740 & 0.035852 \\ 0.037329 & 0.021331 \end{bmatrix}$$

$$\xi_3 = \begin{bmatrix} 0.2839837 & 0.09127753 \\ 0.13480176 & 0.06519824 \end{bmatrix}$$

3.4 Update values

$$\hat{\pi} = \gamma_1 = \begin{bmatrix} 0.86784141 \\ 0.1321589 \end{bmatrix}$$

$$\hat{A} = (\mathbb{1} \oslash \gamma) \bullet \xi$$

$$\hat{B} = (\mathbb{1} \oslash \gamma) \bullet (\sum_{t=1}^{T} \gamma_t \otimes \mathbb{1}_{yt}^T)$$

When referring to γ, we use the sum of the probabilities:

$$\gamma = \sum_{t=1}^{T} \gamma_t$$

and ξ:

$$\xi = \sum_{t=1}^{T} \xi_t$$

We therefore calculate:

$$\gamma = \begin{bmatrix} 0.86784141 \\ 0.1321589 \end{bmatrix} + \begin{bmatrix} 0.42888609 \\ 0.57111391 \end{bmatrix} + \begin{bmatrix} 0.67400881 \\ 0.32599119 \end{bmatrix} = \begin{bmatrix} 1.97073631 \\ 1.02926369 \end{bmatrix}$$

And

$$\xi = \begin{bmatrix} 0.38325991 & 0.03650094 \\ 0.60572687 & 0.08653241 \end{bmatrix} + \begin{bmatrix} 0.07341938 & 0.06873304 \\ 0.03726872 & 0.0523348 \end{bmatrix}$$

$$+ \begin{bmatrix} 0.2830837 & 0.09127753 \\ 0.13480176 & 0.06519824 \end{bmatrix} = \begin{bmatrix} 0.739763 & 0.19651152 \\ 0.77779736 & 0.20406545 \end{bmatrix}$$

We can now calculate

$$\mathbb{1} \oslash \gamma = \begin{bmatrix} \frac{1}{2.0923} \\ \frac{1}{1.1352} \end{bmatrix}$$

We can now calculate $\hat{A}$

$$\hat{A} = \begin{bmatrix} \frac{1}{2.0923} \\ \frac{1}{1.1352} \end{bmatrix} \bullet \begin{bmatrix} 0.9897 & 0.7370 \\ 0.5670 & 0.3888 \end{bmatrix} = \begin{bmatrix} 0.37537391 & 0.19092437 \\ 0.39467348 & 0.198226353 \end{bmatrix}$$

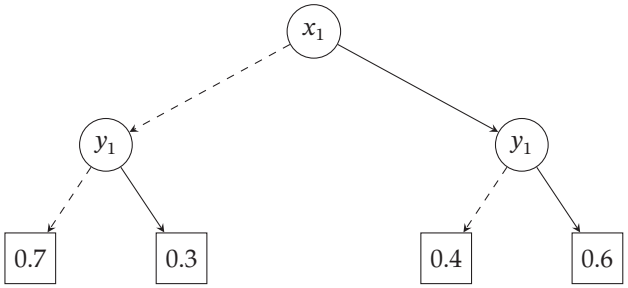

Fig. 1. B-matrix representation in ADD

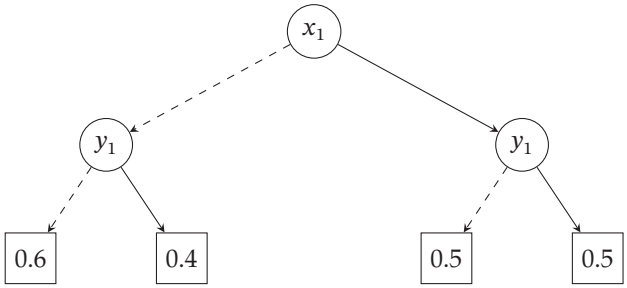

Fig. 2. A-matrix representation in ADD

We calculate $\hat{B}$ We first calculate the sum of the outer products:

$$\sum_{t=1}^{T} \gamma_t \otimes \mathbb{1}_{yt}^{T}$$

At t = 1:

$$\gamma_1 \otimes \begin{bmatrix} 1 & 0 \end{bmatrix} = \begin{bmatrix} 0.86784141 \\ 0.1321589 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \end{bmatrix} = \begin{bmatrix} 0.86784141 & 0.13215859 \\ 0 & 0 \end{bmatrix}$$

At t = 2:

$$\gamma_2 \otimes \begin{bmatrix} 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.42888609 \\ 0.57111391 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0.42888609 & 0.57111391 \end{bmatrix}$$

At t = 3:

$$\gamma_3 \otimes \begin{bmatrix} 1 & 0 \end{bmatrix} = \begin{bmatrix} 0.67400881 \\ 0.32599119 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \end{bmatrix} = \begin{bmatrix} 0.67400881 & 0.32599119 \\ 0 & 0 \end{bmatrix}$$

We summearize these to get:

$$\begin{bmatrix} 0.86784141 & 0.13215859 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0.42888609 & 0.57111391 \end{bmatrix} +$$

$$\begin{bmatrix} 0.67400881 & 0.32599119 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1.54185022 & 0.45814978 \\ 0.42888609 & 0.57111391 \end{bmatrix}$$

$$\hat{b} = \begin{bmatrix} \frac{1}{2.0923} \\ \frac{1}{1.1352} \end{bmatrix} \bullet \begin{bmatrix} 1.54185022 & 0.45814978 \\ 0.42888609 & 0.57111391 \end{bmatrix}$$

$$= \begin{bmatrix} 0.78237266 & 0.23247645 \\ 0.41669214 & 0.55487618 \end{bmatrix}$$

3.5 ADD representation

As we only need one bit to represent the the rows and columns with one bit, we only need one variable for the them, as x_1 is the variable for rows and y_1 is the variable for column.

We first make the matrices into ADD representation.

We can now use the ADD representation to calculate α and β.

When using ADD's it is important to remember, if we need to take a row from a matrix, we fix the input to the ADD by setting the x-variables to the desired row. An example is taking the third row of a matrix with 8 rows, we set, $x_1 = 1, x_2 = 1, x_3 = 0$ and $x_4 = 0$. if we need to take the second column, we set $y_1 = 1, y_2 = 0$ and $y_3 = 0, y_4 = 0$. Hadamard product is row-wise multiplication of the matrices.

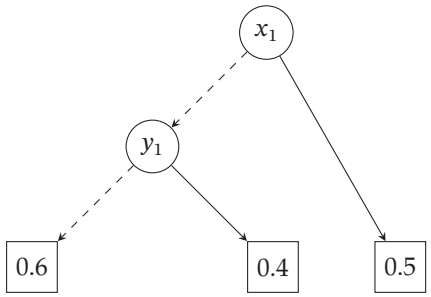

Fig. 3. A-matrix representation in ADD caption reduced

So to calculate the Hadamard product of two matrices, we set the x-variables to the same row in both matrices and multiply the corresponding nodes in the ADDs. To calculate Hadamard product in ADD, we multiply the corresponding nodes in the ADDs, as shown in the following figure.

Matrix multiplication is done by fixing the input to the first matrix and the output to the second matrix. We then sum the result of the Hadamard product of the rows of the first matrix and the columns of the second matrix. This is shown in the following figure.

4 Class Diagram

The class diagram in Figure 14 shows the relationships between the different classes in the system. The `Model` class

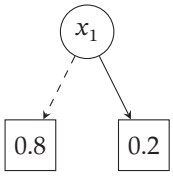

Fig. 4. π-matrix representation in ADD

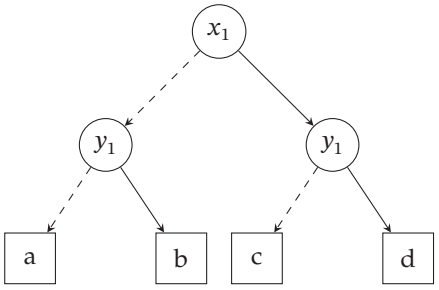

Fig. 5. Matrix A in ADD

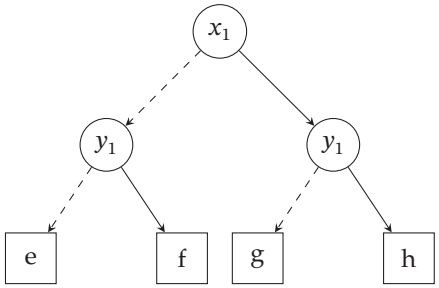

Fig. 6. Matrix B in ADD

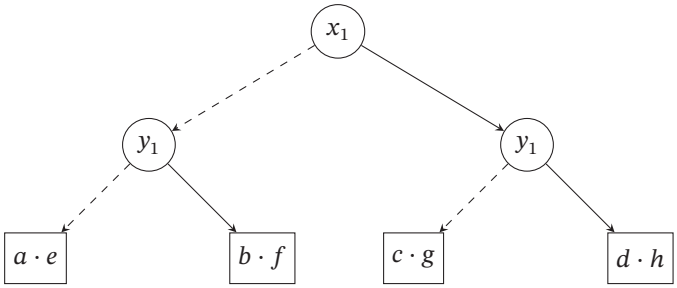

Fig. 7. Hadamard product of A and B in ADD

represents the underlying model of the system, which can be a CTMC, DTMC, HMM and MDP. The `Model` class has a aggregation relationship with the `Algorithm` class, which represents the functions used in the Baum-Welch algorithm, with and without ussing log-semiring. The `Algorithm` class has a dependency relationship with the `CUDD` class, which is a wrapper for the CUDD library. The `CUDD` class is used to perform the matrix operations as ADD's required by the Baum-Welch algorithm. The `Model` class also has a aggregation relationship with the `CUDD` class, as the `Model` class uses the `CUDD` class to perform the ADD operations.

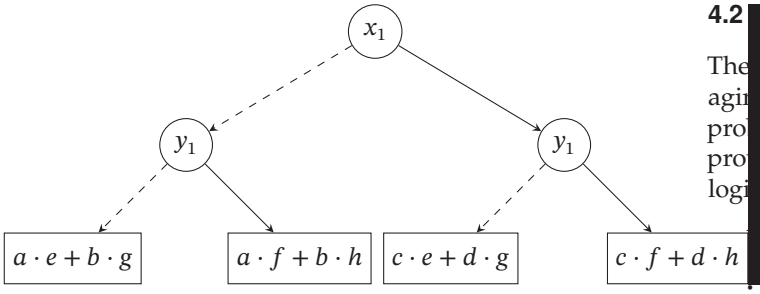

Fig. 8. Matrix multiplication of A and B in ADD

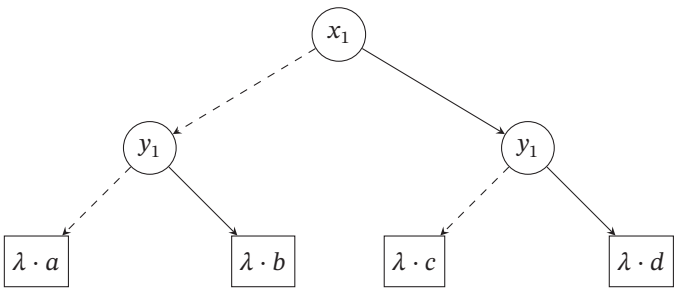

Fig. 9. Scalar product in ADD

4.1 `Model` Class

The `Model` class serves as the foundation for representing various probabilistic models like CTMC, MDP, and DTMC. It holds fields needed to describe these models, such as the transition matrices, emission probabilities, and initial states, all represented using Algebraic Decision Diagrams (ADDs). The `Model` class also provides methods for training models, such as the Baum-Welch algorithm.

Attributes:

- `Type_model`: Defines the type of model (e.g., CTMC, MDP, DTMC).
- `transfer`: A list of ADD structures representing state transition probabilities.
- `Emission`: An ADD for the emission probabilities (relevant in Hidden Markov Models).
- `pi`: The initial state distribution, also stored as an ADD.
- `training_set`: A collection of observed data.

Methods:

- `Instantiate_with_parameters(prismfile, parameters: Dictionary)`: Instantiates a model with specified parameters.
- `Instantiate_without_parameters(prismfile)`: Creates a model without additional parameters.
- `Baum-welsh(log, Model)`: This method implements the Baum-Welch algorithm for training Hidden Markov Models, utilizing various operations from the `Algorithm` class.

4.2 `CUDD` Class

The `CUDD` class (CUDD Manager) is responsible for managing ADDs. These ADDs are crucial in representing the probabilistic data structures used in the `Model` class. `CUDD` provides a set of operations that allow mathematical and logical manipulation of these diagrams.

Attributes:

- `rowvars, colvars`: Representing variables used in the ADD structures.
- `ADD`: The main data structure for storing probabilities or logical expressions.
- `Manager`: A control structure that coordinates operations on ADDs.

Methods:

- `Hadamard(), Log_Hadamard()`: Perform element-wise operations on ADDs.
- `Matrix_mul(), Log_matrix_mul()`: For matrix multiplications.
- `Sum(), Transpose()`: Additional helper methods for summing and transposing ADDs.

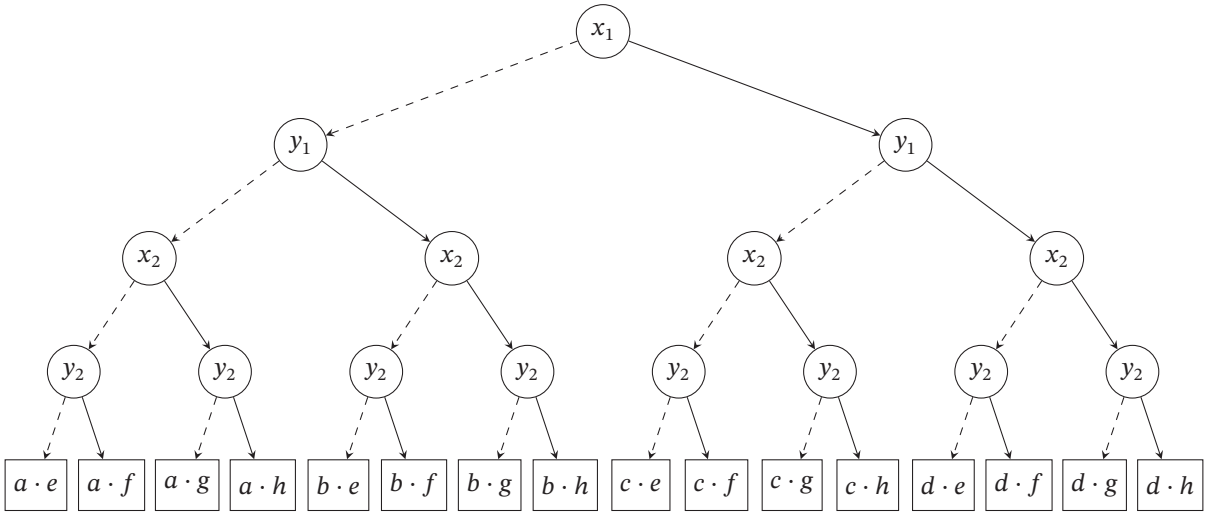

Fig. 10. Kroneker product in ADD

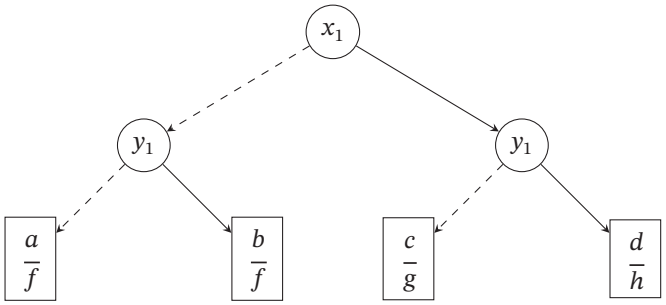

Fig. 11. Hadamard division of A and B in ADD

4.3 `Algorithm` Class

The `Algorithm` class encapsulates various methods for performing probabilistic calculations. These methods are mainly used for inference in models such as Hidden Markov Models (HMM) and Markov Chains.

Methods:

- `calculate_alpha(), calculate_beta()`: Compute the forward (alpha) and backward (beta) probabilities, respectively.
- `calculate_gamma(), calculate_xi()`: Intermediate probability calculations needed for parameter estimation and model training.

Each method operates on the ADD structures created and managed by the `CUDD` class, ensuring efficient computation of the probabilities.

4.4 Relationships Between Classes

4.4.1 `Model` to `Algorithm`: Association Relationship

The `Model` class uses the `Algorithm` class to compute the forward-backward probabilities and other values necessary for inference. The Baum-welsh(log, Model) method in `Model` invokes the relevant methods from `Algorithm` (calculate_alpha(), calculate_beta(), etc.) during the training process of HMMs. These methods, while called collectively in Baum-Welch, can also be used independently to perform specific calculations.

4.4.2 `Model` to `CUDD`: Aggregation Relationship

The `Model` class contains several attributes (transfer, Emission, pi) that are represented as ADDs, managed by the `CUDD` class. This relationship is best represented as an aggregation, where the `Model` holds instances of ADD but does not directly manage their internal workings. Instead, `CUDD` provides the operations required to manipulate and operate on these diagrams, such as matrix multiplication or element-wise functions (Hadamard products). The `Model` depends on `CUDD` for these operations, making it an integral part of the system's backend.

4.4.3 `Algorithm` to `CUDD`: Dependency Relationship

The `Algorithm` class depends on the `CUDD` class for all its operations on ADDs. Every method in `Algorithm` (e.g., calculate_alpha(), calculate_gamma()) relies on ADD operations provided by `CUDD`, such as Matrix_mul() and Hadamard(). This is represented by a dependency relationship, where `Algorithm` calls `CUDD`'s methods to perform its computations.

5 Experiment

The purpose of the experiments is to evaluate the performance of different implementations of the Baum-Welch algorithm when applied to various CTMC models. Specifically, we compare our implementation that utilize ADDs with and without the log semiring to other implementations from Jajapy, namely the original Jajapy implementation of the Baum-Welch algorithm and the improved SUDD implementation. By analyzing both parameter estimation accuracy and runtime, we aim to determine the efficiency and reliability of each implementation across multiple model types. We also investigate the scalability of the Baum-Welch algorithm by increasing the number of states in the model and comparing the runtime of each implementation.

We use 5 different models to test the performance of the Baum-Welch algorithm. Each model provides a different structure and complexity level for testing the Baum-Welch algorithm. The models are derived from CTMCs and include:

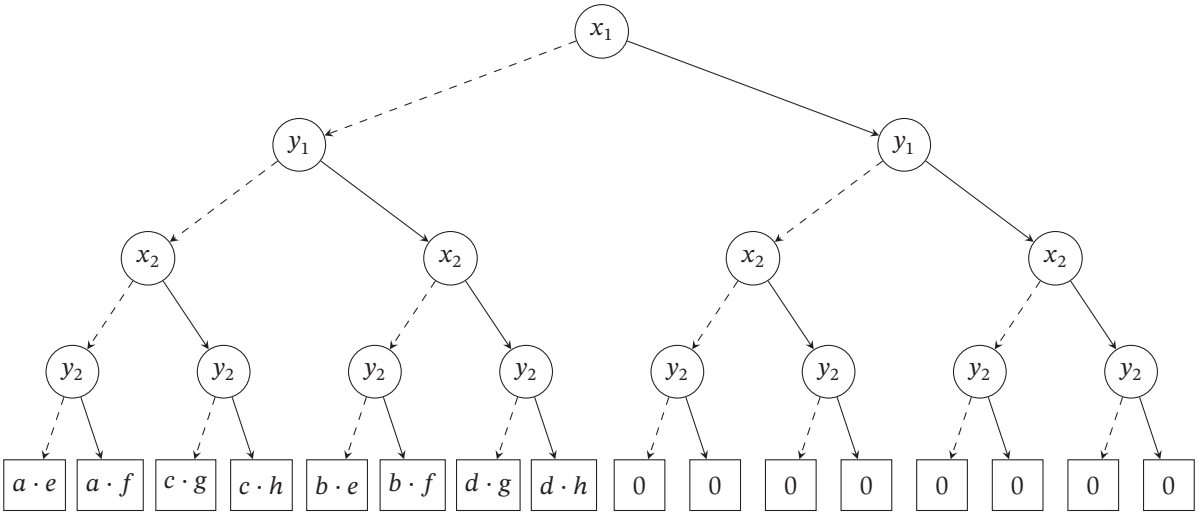

Fig. 12. Katri-Rao in ADD

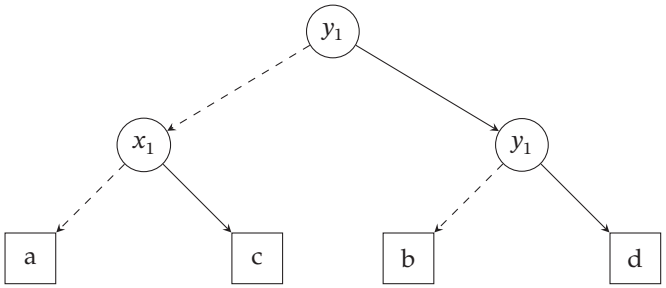

Fig. 13. transpose in ADD

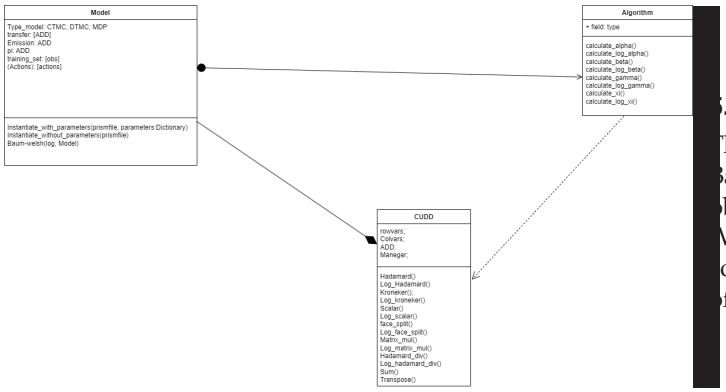

Fig. 14. Class diagram of the system

- **Polling**: A model representing a server polling multiple queues, often used in network communication scenarios.
- **Cluster**: Simulates clusters of entities competing for shared resources.
- **Tandem**: A series of interconnected queues where entities move sequentially from one queue to the next.
- **Philosophers(I) and Philosophers2(II)**: Models representing the classic synchronization problem where entities (philosophers) compete for limited resources (e.g., forks), with Philosophers2 containing one additional parameter.

In every model, we have a number of parameters we want to estimate. The number of parameters to estimate in each model is shown in Table 1. The polling model is the smallest model with 2 parameters to estimate, while philosophers(II) is the largest model, with 4 parameters to estimate.'

TABLE 1
Number of parameters to estimate in each model

Model	Num of parameters
Polling	2
Cluster	3
Tandem	3
Philosophers(I)	3
Philosophers(II)	4

5.1 Parameter Estimation Accuracy

The first experiment measures the time and accuracy of the Baum-Welch algorithm for each implementation. We generate observation sequences from each model and use the Baum-Welch algorithm to estimate the parameters of the model. We compare the estimated parameters with the true parameters of the model to evaluate the accuracy of the estimation.

The experiment is made with the following steps:

1) We load the model.
2) Generate an observation sequence from the model with untimed steps.
3) Calculate the model's parameters using each variant of the Baum-Welch algorithm, recording both the parameter estimates and runtime.
4) Repeat steps 2 and 3 ten times without changing the model, the sequence length, or whether the sequence is timed or untimed. The results are averaged to account for any variance in runtime and estimation accuracy.
5) Repeat with timed steps to observe the effects of timing information on estimation accuracy and runtime.
6) Compare the estimated parameters with the true parameters and record the runtime for each implementation.

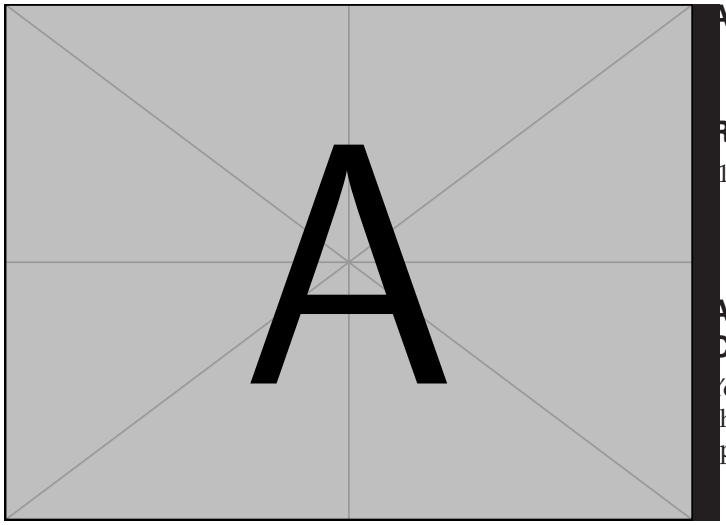

Fig. 15. Scalability of Baum-Welch implementations for the Tandem model

7) Move to the next model and repeat the entire process.

Following the experiment, the estimated parameters for each implementation are compared with the true parameters for accuracy assessment. Additionally, runtime for each Baum-Welch implementation is recorded. These results are presented in tables and plots to facilitate a direct comparison of performance across models and between implementations.

Table 2, 3, 4, 5 and 6 show detailed results for each model in terms of runtime and estimation accuracy (relative formula error and relave parameter error).

5.2 Scalability experiment

We also test the scalability of the Baum-Welch algorithm by increasing the number of states in a model. We use the Tandem model and increase the number of states from 28 to 1225. We then compare the runtime of the Baum-Welch algorithm for each implementation. We run the experiment 10 times for each number of states and compare the runtime of the Baum-Welch algorithm for each implementation.

We use plots to illustrate the scalability by showing runtime across an increasing number of states for the Tandem model, highlighting each implementation's computational efficiency.

5.3 Results

The comparison of the Baum-Welch implementations is based on the following criteria:

- **Parameter estimation accuracy**: The accuracy of the estimated parameters compared to the true parameters.
- **Runtime**: The time it takes to estimate the parameters.
- **Scalability**: How the runtime scales with the number of states in the model.

The results are displayed as tables and plots to facilitate a direct comparison of performance across models and between implementations.

Acronyms

AAU Aalborg University. 1

References

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

Appendix A
Compiling in draft

You can also compile the document in draft mode. This shows todos, and increases the space between lines to make space for your supervisors feedback.

TABLE 2
Comparison of Baum-Welch implementations for Polling

Implementation	Timed Observations				Untimed Observations			
	Time(s)	Iteration	avg δ	avg ϕ	Time(s)	Iteration	avg δ	avg ϕ
CuPAAL	132.5	3.5	0.1	0.1	132.5	3.5	0.1	0.1
CuPAAL_log	132.5	3.5	0.1	0.1	132.5	3.5	0.1	0.1
SUDD	23.5	3.5	0.1	0.1	23.5	3.5	0.1	0.1
SUDD_log	0.5	3.5	0.1	0.1	0.5	3.5	0.1	0.1
Jajapy	9.5	3.5	0.1	0.1	9.5	3.5	0.1	0.1

TABLE 3
Comparison of Baum-Welch implementations for Cluster

Implementation	Timed Observations				Untimed Observations			
	Time(s)	Iteration	avg δ	avg ϕ	Time(s)	Iteration	avg δ	avg ϕ
CuPAAL	132.5	3.5	0.1	0.1	132.5	3.5	0.1	0.1
CuPAAL_log	132.5	3.5	0.1	0.1	132.5	3.5	0.1	0.1
SUDD	23.5	3.5	0.1	0.1	23.5	3.5	0.1	0.1
SUDD_log	0.5	3.5	0.1	0.1	0.5	3.5	0.1	0.1
Jajapy	9.5	3.5	0.1	0.1	9.5	3.5	0.1	0.1

TABLE 4
Comparison of Baum-Welch implementations for Tandem

Implementation	Timed Observations				Untimed Observations			
	Time(s)	Iteration	avg δ	avg ϕ	Time(s)	Iteration	avg δ	avg ϕ
CuPAAL	132.5	3.5	0.1	0.1	132.5	3.5	0.1	0.1
CuPAAL_log	132.5	3.5	0.1	0.1	132.5	3.5	0.1	0.1
SUDD	23.5	3.5	0.1	0.1	23.5	3.5	0.1	0.1
SUDD_log	0.5	3.5	0.1	0.1	0.5	3.5	0.1	0.1
Jajapy	9.5	3.5	0.1	0.1	9.5	3.5	0.1	0.1

TABLE 5
Comparison of Baum-Welch implementations for Philosophers(I)

Implementation	Timed Observations				Untimed Observations			
	Time(s)	Iteration	avg δ	avg ϕ	Time(s)	Iteration	avg δ	avg ϕ
CuPAAL	132.5	3.5	0.1	0.1	132.5	3.5	0.1	0.1
CuPAAL_log	132.5	3.5	0.1	0.1	132.5	3.5	0.1	0.1
SUDD	23.5	3.5	0.1	0.1	23.5	3.5	0.1	0.1
SUDD_log	0.5	3.5	0.1	0.1	0.5	3.5	0.1	0.1
Jajapy	9.5	3.5	0.1	0.1	9.5	3.5	0.1	0.1

TABLE 6
Comparison of Baum-Welch implementations for Philosophers(II)

Implementation	Timed Observations				Untimed Observations			
	Time(s)	Iteration	avg δ	avg ϕ	Time(s)	Iteration	avg δ	avg ϕ
CuPAAL	132.5	3.5	0.1	0.1	132.5	3.5	0.1	0.1
CuPAAL_log	132.5	3.5	0.1	0.1	132.5	3.5	0.1	0.1
SUDD	23.5	3.5	0.1	0.1	23.5	3.5	0.1	0.1
SUDD_log	0.5	3.5	0.1	0.1	0.5	3.5	0.1	0.1
Jajapy	9.5	3.5	0.1	0.1	9.5	3.5	0.1	0.1