Parallel algorithms for fitting Markov Arrival Processes

Mindaugas Bražėnas[a], Gábor Horváth[b], Miklós Telek[b,c]

[a]*Department of Applied Mathematics, Kaunas University of Technology, Lithuania*
[b]*Department of Networked Systems and Services, Budapest University of Technology and Economics, Hungary*
[c]*MTA-BME Information Systems Research Group, Hungary*

Abstract

The fitting of Markov arrival processes (MAPs) with the expectation-maximization (EM) algorithm is a computationally demanding task. There are attempts in the literature to reduce the computational complexity by introducing special MAP structures instead of the general representation. Another possibility to improve the efficiency of MAP fitting is to reformulate the inherently serial classical EM algorithm to exploit modern, massively parallel hardware architectures.

In this paper we present three different EM-based fitting procedures that can take advantage of the parallel hardware (like Graphics Processing Units, GPUs) and apply a special MAP structure, the Erlang distributed - continuous-time hidden Markov chain (ER-CHMM) structure for reducing the computational complexity.

All the proposed parallel algorithms have their strengths: the first one traverses the samples only once per iteration, the second one is memory efficient (far more than the classical serial algorithm), and the third one has exceptionally low execution times.

These procedures are compared with the standard serial forward-backward procedure for performance comparison. The new algorithms are orders of magnitudes faster than the standard serial procedure, while (depending on the variant) using less memory.

Keywords: Markov arrival process, traffic model fitting, EM algorithm, parallel computation, GPU

1. Introduction

Markov arrival processes (MAPs) are being used for modeling correlated workload for traffic, performance and reliability analysis in several fields for many decades [8]. However, for the successful application of MAP-based models, efficient fitting procedures are needed to approximate the real traffic behavior as accurately as possible.

The MAP fitting approaches published so far can be divided into matching, distance minimization, and combined methods, where inter-arrival time fitting and correlation structure fitting are performed with different approaches. The matching algorithms aim to match certain statistical quantities of the traffic such as moments and auto-correlation. Fitting methods belonging to the second group aim to minimize a measure of distance between samples and the model. In case of MAP traffic models, the dominant distance measure is the likelihood, and the dominant optimization procedure is the expectation maximization (EM) method aiming to maximize the likelihood.

Explicit results for MAP matching methods exist only for the second-order case [2]. For larger models, a combined two-step procedure for the MAP fitting problem has been developed, e.g., in [4]. In the first step a phase-type (PH) distribution is created to fit the marginal distribution, and in the second step, a lag-k auto-correlation fitting is performed to capture the correlated nature of the traffic. Another two-step

Email addresses: mindaugas.brazenas@yahoo.com (Mindaugas Bražėnas), ghorvath@hit.bme.hu (Gábor Horváth), telek@hit.bme.hu (Miklós Telek)

matching approach is proposed in [7], where an acyclic PH distribution is extended into a MAP by adding correlations based on the lag-1 joint moments of the inter-arrival times. The first, PH fitting step can be improved by representation transformation (see [6, 13]), to make the PH representation more appropriate for correlation fitting. However, the common flaw of these two-step algorithms is that they are not able to take long-range correlations into account and that the underlying optimization problems are not easy to solve even with most recent optimization software.

An EM procedure to perform MAP fitting of a general structure is presented in [5]. That procedure requires a massive computational effort, and consequently, it is applicable only to fit small data traces consisting of a few thousand observations. To address the issue an EM procedure based on the aggregation of the inter-arrival times is presented in [11]. Similarly, [17] extends the EM algorithm for fitting MAPs to group data. To reduce the computational demand of EM-based MAP fitting algorithms a special MAP structure was introduced in [16] called Erlang distributed – continuous-time hidden Markov chain (ER-CHMM). A generalization of the ER-CHMM structure was given in [9], which relaxed some structural restriction of the ER-CHMM structure, but increased the number of the model parameters. Using the generalized ER-CHMM structure slightly higher likelihood values can occasionally be achieved, on the cost of increased numerical complexity.

In the survey [10], various MAP fitting approaches have been compared, and the EM-based algorithms have been found to be more beneficial than the matching ones (including the combined methods) from many aspects. The reason is that the EM method considers all information carried by the samples, while the matching methods consider only the statistical parameters which are matched.

While EM algorithms for fitting PH distributions can easily take advantage of the parallel hardware since the samples are independent, all EM-based procedures for MAP fitting published so far are inherently serial algorithms due to the dependent nature of the samples.

There are a few EM algorithms published in the literature for hidden Markov model (HMM) fitting, that is also based on dependent samples [12, 15, 20]. In all these papers, similar to ours, formalizing the problem with matrices was the key idea enabling the parallel implementation. [20, eq. (30)] provides an important relation for the iterative, parallel computation of the performance indexes for the HMM problem with continuous observations, but this idea has not been put forward to an efficient parallel implementation in [20]. [12] and [15] proposed parallel solutions, without using [20, eq. (30)], for a simpler HMM setting with discrete observations. These HMM models differ from the MAP fitting problem in the meaning of the parameters to estimate, but their computational approaches are similar to ours. Our contribution is that we adapt the principal relation [20, eq. (30)] for the MAP fitting problem, based on which we develop parallel algorithms organized similarly to the ones in [12, 15]. Mixing the elements of these algorithms in different ways we arrive to three different algorithm variants, which covers the practically relevant spectrum of the computation time – memory requirement trade-off. Having these algorithms defined and implemented in a unified manner enabled their fair performance comparison.

The rest of the paper is organized as follows. In Section 2 a short introduction on MAPs and the basic EM algorithm is provided. Possible implementations of the EM algorithm, including the parallel ones, are investigated in Section 3. Section 4 provides more implementation details on how to cope with the numerical problems and a detailed study on the memory requirement is also given. Section 5 demonstrates the efficiency of our parallel implementation on some numerical examples and concludes the paper.

2. Background information

2.1. Markov arrival processes

A Markov arrival process (MAP, [14]) is a point process where the arrivals are modulated by a finite state background continuous-time Markov chain (CTMC) with N states and generator matrix $\boldsymbol{D}$ (assumed to be irreducible in this paper). A set of the state transitions, with rates specified by matrix $\boldsymbol{D_1}$, are accompanied by an arrival event, while other state transitions described by matrix $\boldsymbol{D_0}$ are just internal transitions of the background process, not accompanied by an arrival (see Figure 1). Hence, we have that $\boldsymbol{D} = \boldsymbol{D_0} + \boldsymbol{D_1}$ and $(\boldsymbol{D_0} + \boldsymbol{D_1})\,\mathbb{1} = \underline{0}$. Let the stationary state distribution vector at arrival instants be denoted by $\underline{\alpha}$. It can

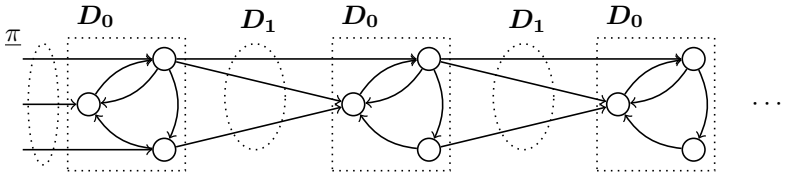

Figure 1: The structure of a general MAP.

be obtained from solving $\underline{\alpha}(-\boldsymbol{D_0})^{-1}\boldsymbol{D_1} = \underline{\alpha}, \underline{\alpha}\mathbb{1} = 1$ with $\mathbb{1}$ being a column vector of ones. The density function of the stationary inter-arrival time $f(x)$ and its nth moment $E(\mathcal{X}^n)$ can be expressed by

$$f(x) = \underline{\alpha} e^{\boldsymbol{D_0} x} \boldsymbol{D_1} \mathbb{1}, \tag{1}$$

$$E(\mathcal{X}^n) = n! \underline{\alpha}(-\boldsymbol{D_0})^{-n} \mathbb{1}, \tag{2}$$

while the joint density function of inter-arrival times $x_1, x_2, \ldots, x_T$ is given by

$$f(x_1, x_2, \ldots x_T) = \underline{\alpha} e^{\boldsymbol{D_0} x_1} \boldsymbol{D_1} e^{\boldsymbol{D_0} x_2} \boldsymbol{D_1} \ldots e^{\boldsymbol{D_0} x_T} \boldsymbol{D_1} \mathbb{1} = \underline{\alpha} \prod_{u=1}^{T} \boldsymbol{A}[u] \mathbb{1}, \tag{3}$$

where $\boldsymbol{A}[u] = e^{\boldsymbol{D_0} x_u} \boldsymbol{D_1}$.

The $\boldsymbol{D_0}, \boldsymbol{D_1}$ matrix representation of a MAP is not unique [18], which has important practical consequences on distance minimizing procedures. Most of them, including the EM-based ones, might go back and forth between MAP representations that are almost identical with respect to the underlying MAP but have very different matrix representations. To avoid this problem and to reduce computational complexity structurally restricted subclasses of MAPs are used.

2.2. The ER-CHMM structure

ER-CHMM is a structurally restricted subclass of MAPs, introduced in [16]. In this arrival process, the inter-arrival times are Erlang distributed (ER) and are modulated by a discrete time hidden Markov chain (HMM, hence the name of the structure). An important qualitative property of this structural restriction is that given the state of the modulating Markov chain, the inter-arrival time and the next state of the modulating Markov chain are independent random variables, which is not the case with general MAPs. Despite this independence, the inter-arrival times generated by the ER-CHMM structure are still correlated (in general).

The ER-CHMM structure is specified by the parameters $\underline{r} = \{r_i\}, \underline{\lambda} = \{\lambda_i\}, \boldsymbol{\Pi} = \{p_{i,j}\}$ for $i, j = 1, ..., R$, where R is the number of Erlang branches, r_i and λ_i are the order and the rate of the Erlang distribution in branch i. Hence, the density of the inter-arrival times generated by branch i is

$$f_i(x) = \frac{(\lambda_i x)^{r_i - 1}}{(r_i - 1)!} \lambda_i e^{-\lambda_i x}. \tag{4}$$

The process determining the branch providing the consecutive inter-arrival times, $\{\mathcal{Y}_u, u \geq 1\}$, is a discrete time Markov chain (DTMC) with R states and transition probability matrix $\boldsymbol{\Pi}$, thus its i, j element, $p_{i,j} = \lim_{u \to \infty} P(\mathcal{Y}_u = j | \mathcal{Y}_{u-1} = i)$, is the probability that after generating an arrival by branch i the next arrival is generated by branch j (see Figure 2).

From the $\{\underline{r}, \underline{\lambda}, \boldsymbol{\Pi}\}$ parameters it is easy to obtain the $\{\boldsymbol{D_0}, \boldsymbol{D_1}\}$ representation. If the initial vector and the transient generator of an order-r Erlang distribution with parameter λ are given by

$$\underline{\beta}(r) = \begin{bmatrix} 1 & 0 & \ldots & 0 \end{bmatrix}_{1 \times r}, \qquad \boldsymbol{B}(r, \lambda) = \begin{bmatrix} -\lambda & \lambda & 0 & \ldots \\ 0 & -\lambda & \lambda & \ldots \\ & & \ddots & \ddots \\ 0 & 0 & 0 & -\lambda \end{bmatrix}_{r \times r},$$

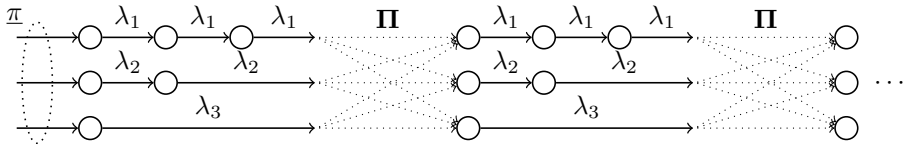

Figure 2: The $\underline{r} = \{3, 2, 1\}$ ER-CHMM structure.

the matrices $\boldsymbol{D_0}$ and $\boldsymbol{D_1}$ can be expressed by

$$\boldsymbol{D_0} = \begin{bmatrix} \boldsymbol{B}(r_1, \lambda_1) & & \\ & \ddots & \\ & & \boldsymbol{B}(r_R, \lambda_R) \end{bmatrix}, \boldsymbol{D_1} = \begin{bmatrix} -\boldsymbol{B}(r_1, \lambda_1)\mathbb{1} \cdot p_{1,1} \cdot \underline{\beta}(r_1) & \dots & -\boldsymbol{B}(r_1, \lambda_1)\mathbb{1} \cdot p_{1,R} \cdot \underline{\beta}(r_R) \\ \vdots & \ddots & \vdots \\ -\boldsymbol{B}(r_R, \lambda_R)\mathbb{1} \cdot p_{R,1} \cdot \underline{\beta}(r_1) & \dots & -\boldsymbol{B}(r_R, \lambda_R)\mathbb{1} \cdot p_{R,R} \cdot \underline{\beta}(r_R) \end{bmatrix}. \tag{5}$$

From a computational point of view, the most beneficial feature of the ER-CHMM structure is that the computation of matrices $\boldsymbol{A}[u]$, the key element of the joint density function, is significantly simpler with the ER-CHMM structure than in case of general MAPs, due to the conditional independence of the inter-arrival time and the next state of the modulating Markov chain, which is provided by the the ER-CHMM structure. As a consequence, the computation of matrices $\boldsymbol{A}[u]$ does not rely on the computationally heavy matrix-exponential function, but simplifies to a scalar product of $p_{i,j}$ and $f_i(x_u)$ as it is demonstrated by the relevant elements of $\boldsymbol{A}[u]$ in the following example. If $\underline{r} = [3, 1]$, the $\boldsymbol{D_0}, \boldsymbol{D_1}$ and $\boldsymbol{A}[u]$ are

$$\boldsymbol{D_0} = \begin{bmatrix} -\lambda_1 & \lambda_1 & 0 & 0 \\ 0 & -\lambda_1 & \lambda_1 & 0 \\ 0 & 0 & -\lambda_1 & 0 \\ 0 & 0 & 0 & -\lambda_2 \end{bmatrix}, \quad \boldsymbol{D_1} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ p_{1,1}\lambda_1 & 0 & 0 & p_{1,2}\lambda_1 \\ p_{2,1}\lambda_2 & 0 & 0 & p_{2,2}\lambda_2 \end{bmatrix},$$

$$\boldsymbol{A}[u] = e^{\boldsymbol{D}_0 x_u}\boldsymbol{D}_1 = \begin{bmatrix} p_{1,1}f_1(x_u) & 0 & 0 & p_{1,2}f_1(x_u) \\ \bullet & 0 & 0 & \bullet \\ \bullet & 0 & 0 & \bullet \\ p_{2,1}f_2(x_u) & 0 & 0 & p_{2,2}f_2(x_u) \end{bmatrix},$$

where $\bullet$ indicate matrix entries which are irrelevant because after an arrival event the phase is either 1 or 4 due to the zero columns. This fact allows for dimension reduction as well, that we utilize in the sequel: instead of relying on size N matrices $\boldsymbol{A}[u]$, we are going to use size R matrices $\boldsymbol{P}[u]$, defined as

$$\boldsymbol{P}[u] = \begin{bmatrix} p_{1,1}f_1(x_u) & \dots & p_{1,R}f_1(x_u) \\ \vdots & \ddots & \vdots \\ p_{R,1}f_R(x_u) & \dots & p_{R,R}f_R(x_u) \end{bmatrix}. \tag{6}$$

Using these size R matrices the joint density function is

$$f(x_1, x_2, \dots x_T) = \underline{\pi} \prod_{u=1}^{T} \boldsymbol{P}[u]\mathbb{1}, \tag{7}$$

where $\underline{\pi}$ is the stationary distribution of $\mathbf{\Pi}$, hence $\underline{\pi}\mathbf{\Pi} = \underline{\pi}, \underline{\pi}\mathbb{1} = 1$. Observe that, since $R \leq N$ holds, (7) involves smaller matrices and lacks matrix-exponential functions compared to (3).

2.3. The EM algorithm for MAP fitting with the ER-CHMM structure

Given matrices $\boldsymbol{D_0}$ and $\boldsymbol{D_1}$, the likelihood of the observations $\underline{x} = \{x_1, x_2, \dots, x_T\}$ can be expressed by

$$\mathcal{L}(\underline{x}, \boldsymbol{D_0}, \boldsymbol{D_1}) = \underline{\alpha} e^{\boldsymbol{D_0} x_1} \boldsymbol{D_1} e^{\boldsymbol{D_0} x_2} \boldsymbol{D_1} \cdots e^{\boldsymbol{D_0} x_T} \boldsymbol{D_1} \mathbb{1} = \underline{\alpha} \prod_{u=1}^{T} \boldsymbol{A}[u]\mathbb{1}. \tag{8}$$

The aim of the EM algorithm is to find the matrices $\boldsymbol{D_0}$ and $\boldsymbol{D_1}$ of size N that maximize the likelihood of the observations, thus

$$\boldsymbol{D_0}, \boldsymbol{D_1} = \arg\max \mathcal{L}(\underline{x}, \boldsymbol{D_0}, \boldsymbol{D_1}). \tag{9}$$

In case of the ER-CHMM structure with $\underline{r}$ the same likelihood optimization problem can be formulated as

$$\underline{\lambda}, \boldsymbol{\Pi} = \arg\max \mathcal{L}(\underline{x}, \underline{\lambda}, \boldsymbol{\Pi}), \quad \text{with} \quad \mathcal{L}(\underline{x}, \underline{\lambda}, \boldsymbol{\Pi}) = \underline{\pi} \prod_{u=1}^{T} \boldsymbol{P}[u] \mathbb{1}. \tag{10}$$

Both [16] and [9] arrived to the interesting conclusion that the structurally restricted ER-CHMM-based EM algorithm, while being significantly faster than the EM algorithm operating on the full (dense) MAP class, often achieves significantly higher likelihood values. The reason is that in case of full MAPs the likelihood function has a large number of local optima due to the high degree of freedom, making the optimization more difficult and more dependent on the initial guess.

Our goal in this paper is to develop a parallel implementation of the EM algorithm for the ER-CHMM class to enable the fitting of large data traces of practical size. We note, however, that the concepts introduced in this paper for the parallel implementation are general enough to be adopted for any MAP with or without structural restrictions. That is why we occasionally indicate both the $\boldsymbol{D_0}, \boldsymbol{D_1}$ and the $\{\underline{r}, \underline{\lambda}, \boldsymbol{\Pi}\}$ based description in the sequel. As the main focus of the paper is the *implementation* of the EM algorithm, here we only summarize the main steps of the method without detailed explanation. For a complete description of the algorithm we refer to [16] and [9].

Two sets of vectors, called forward and backward likelihood vectors, $\underline{a}[u] = \{a_i[u], i = 1, \ldots, R\}, u = 0, \ldots, T$ and $\underline{b}[u] = \{b_i[u], i = 1, \ldots, R\}, u = 1, \ldots, T+1$, play important role in the procedure. Row vectors $\underline{a}[u]$ and column vectors $\underline{b}[u]$ have probabilistic interpretations: $a_i[u]$ is the density that the state of the background DTMC $\mathcal{Y}_u$ is i and inter-arrival times $x_1, \ldots, x_u$ are observed, while $b_j[u]$ is the density of observing inter-arrival times $x_u, \ldots, x_T$ if the state of $\mathcal{Y}_u$ was j initially. They are defined recursively by

$$\begin{aligned} a_i[u] &= \begin{cases} \pi_i, & \text{for } u = 0, \\ \sum_{j=1}^{R} a_j[u-1] f_j(x_u) p_{j,i}, & \text{for } 1 \le u \le T, \end{cases} \\ b_j[u] &= \begin{cases} \sum_{i=1}^{R} f_j(x_u) p_{j,i} b_i[u+1], & \text{for } 1 \le u \le T, \\ 1, & \text{for } u = T+1. \end{cases} \end{aligned} \tag{11}$$

We note that similar recursions can be defined in case of full MAPs, too, see eq. (13) and (14) in [16].

With the above introduced notations the likelihood can be expressed as

$$\mathcal{L}(\underline{x}, \underline{\lambda}, \boldsymbol{\Pi}) = \underline{a}[T]\mathbb{1} = \underline{\pi}\underline{b}[1].$$

According to the EM algorithm the estimates for parameters $\underline{\lambda}$ and $\boldsymbol{\Pi}$ are obtained by

$$\lambda_i' = \frac{\sum_{u=1}^{T} r_i a_i[u-1] b_i[u]}{\sum_{u=1}^{T} x_u a_i[u-1] b_i[u]}, \tag{12}$$

$$p_{i,j}' = \frac{\sum_{u=1}^{T-1} a_i[u-1] f_i(x_u) p_{i,j} b_j[u+1]}{\sum_{u=1}^{T-1} a_i[u-1] b_i[u]}. \tag{13}$$

The estimates of the initial branch probabilities $\underline{\pi} = \{\pi_i, i = 1, \ldots, R\}$ can either be obtained as the stationary solution of $\boldsymbol{\Pi}$, or they can be derived from the likelihood vectors as

$$\pi_i' = \frac{\sum_{u=1}^{T} a_i[u-1] b_i[u]}{T \cdot \underline{a}[T]\mathbb{1}}. \tag{14}$$

The EM algorithm consists of the alternating computation of vectors $\underline{a}[u]$ and $\underline{b}[u]$ according to (11) (also referred to as the *E-step*) and the new estimates according to (12), (13) and (14) (referred to as the *M-step*). When the relative change of the log-likelihood over subsequent iterations is smaller than a pre-defined value, the algorithm stops.

In this paper, we assume that the $r_i, i = 1, \ldots, R$ parameters, the orders of the Erlang branches, are fixed and are not the subject of optimization. Some heuristics to find the right branch orders are provided in [9].

3. The algorithms

3.1. The basic forward-backward algorithm

The standard, naive implementation of the EM algorithm is serial. Computing and saving the likelihood vectors $\underline{a}[u]$ and $\underline{b}[u]$ for $u = 1, \ldots, T$ based on (11) implies $2T$ vector-matrix multiplication operations of size R and saving $2T$ vectors of size R in the memory. Having the likelihood vectors $\underline{a}[u]$ and $\underline{b}[u]$ for $u = 1, \ldots, T$, the λ_i', $p_{i,j}'$, and π_i' estimates ($R^2 + 2R$ parameters) can be computed based on (12), (13) and (14) that needs $\mathcal{O}(T)$ scalar multiplications for all parameters.

There are some straightforward, albeit limited possibilities to make this essentially serial computation parallel. Vectors $\underline{a}[u]$, for $u = 1, \ldots, T$, and vectors $\underline{b}[u]$, for $u = T, \ldots, 1$, can be obtained simultaneously, by two execution threads. The recursive definition of these vectors does not allow a higher level of parallelism. Updating the estimates in the M-step, i.e., computing (12), (13) and (14), can also benefit from parallel hardware, λ_i' and π_i' can be computed simultaneously for $i = 1, \ldots, R$ by $2R$ execution threads, and $p_{i,j}'$ for $i, j = 1, \ldots, R$ by R^2 execution threads.

In the subsequent sections, several options are considered to transform this algorithm to a massively parallel one with different number of passes through the data set.

3.2. The parallel implementation with one pass

We start by reformulating the basic forward-backward algorithm with matrix notation, which is an important step towards making the algorithm massively parallel.

By defining matrix $\vec{\boldsymbol{P}}[u, v]$ for $1 \leq u \leq v \leq T$ as

$$\vec{\boldsymbol{P}}[u, v] = \boldsymbol{P}[u]\boldsymbol{P}[u+1]\ldots\boldsymbol{P}[v] = \prod_{z=u}^{v} \begin{bmatrix} f_1(x_z)p_{1,1} & \ldots & f_1(x_z)p_{1,R} \\ \vdots & \ddots & \vdots \\ f_R(x_z)p_{R,1} & \ldots & f_R(x_z)p_{R,R} \end{bmatrix}, \tag{15}$$

and for $u > v$ as $\vec{\boldsymbol{P}}[u, v] = \boldsymbol{I}$, the forward and backward likelihood vectors can be expressed by

$$\underline{a}[u] = \underline{\pi}\vec{\boldsymbol{P}}[1, u], \quad \text{and} \quad \underline{b}[u] = \vec{\boldsymbol{P}}[u, T]\mathbb{1}. \tag{16}$$

We divide the T inter-arrival times into L partitions: inter-arrival times $x_1, \ldots, x_K$ are assigned to partition 1, $x_{K+1}, \ldots, x_{2K}$ to partition 2, and so on, where the number of inter-arrival times in the first $L-1$ partitions is $K = \lceil T/L \rceil$ and the size of last partition is $T - (L-1)K$ (which can be smaller than K). If u belongs to the ℓth partition, that is, $(\ell - 1)K + 1 \leq u \leq \ell K$, then we have

$$\underline{a}[u] = \underline{\pi} \left(\prod_{z=1}^{\ell-1} \vec{\boldsymbol{P}}[(z-1)K + 1, zK] \right) \vec{\boldsymbol{P}}[(\ell - 1)K + 1, u], \tag{17}$$

$$\underline{b}[u] = \vec{\boldsymbol{P}}[u, \ell K] \left(\prod_{z=\ell+1}^{L-1} \vec{\boldsymbol{P}}[(z-1)K + 1, zK] \right) \vec{\boldsymbol{P}}[(L-1)K + 1, T]\mathbb{1}. \tag{18}$$

The main observation that enables the parallel implementation is that matrices $\vec{\boldsymbol{P}}[(z-1)K+1, zK]$ can be calculated simultaneously for $\ell = 1, \ldots, L$. To simplify the notation we introduce the *likelihood matrix* corresponding to partition ℓ as

$$\boldsymbol{U}[\ell] = \vec{\boldsymbol{P}}[(\ell-1)K+1, \ell K], \text{ for } \ell = 1, \ldots, L-1, \text{ and } \boldsymbol{U}[L] = \vec{\boldsymbol{P}}[(L-1)K+1, T].$$

The likelihood vectors corresponding to the inter-arrival times up to partition ℓ and from partition ℓ on can be expressed from the likelihood matrices by

$$\underline{\pi}^{\ell} = \underline{a}[\ell K] = \pi \vec{\boldsymbol{P}}[1, \ell K] = \pi \prod_{u=1}^{\ell} \boldsymbol{U}[u], \text{ and } \mathbb{1}^{\ell} = \underline{b}[\ell K + 1] = \vec{\boldsymbol{P}}[\ell K + 1, T]\mathbb{1} = \prod_{u=\ell+1}^{L} \boldsymbol{U}[u]\mathbb{1},$$

respectively. By this notation, for $(\ell-1)K + 1 \leq u \leq \ell K$ the likelihood vectors corresponding to each inter-arrival time simplify to

$$\underline{a}[u] = \underline{\pi}^{\ell-1} \vec{\boldsymbol{P}}[(\ell-1)K+1, u], \quad \text{and} \quad \underline{b}[u] = \vec{\boldsymbol{P}}[u, \ell K]\mathbb{1}^{\ell+1}, \tag{19}$$

where $\vec{\boldsymbol{P}}[(\ell-1)K+1, u]$ and $\vec{\boldsymbol{P}}[u, \ell K]$ are likelihood matrices within partition ℓ.

To compute λ_i', $p_{i,j}'$ and π_i' in parallel we rewrite (12), (13), and (14) as

$$\lambda_i' = \frac{\sum_{u=1}^{T} r_i a_i[u-1] b_i[u]}{\sum_{u=1}^{T} x_u a_i[u-1] b_i[u]} = \frac{r_i \cdot S_i^{(1)}}{S_i^{(2)}}, \tag{20}$$

$$p_{i,j}' = \frac{\sum_{u=1}^{T-1} a_i[u-1] f_i(x_u) p_{i,j} b_j[u+1]}{\sum_{u=1}^{T-1} a_i[u-1] b_i[u]} = \frac{S_{i,j}^{(3)}}{S_i^{(3)}}. \tag{21}$$

$$\pi_i' = \frac{\sum_{u=1}^{T} a_i[u-1] b_i[u]}{T \cdot \underline{a}[T]\mathbb{1}} = \frac{1}{T \cdot \underline{a}[T]\mathbb{1}} S_i^{(1)}. \tag{22}$$

where $S_i^{(1)}$ and $S_i^{(3)}$ differs only in a single element

$$S_i^{(1)} = S_i^{(3)} + a_i[T-1] b_i[T], \tag{23}$$

and $S_i^{(3)}$ can be obtained from $S_{i,j}^{(3)}$ as

$$S_i^{(3)} = \sum_{j=1}^{R} S_{i,j}^{(3)}, \tag{24}$$

by which we focus only on $S_i^{(2)}$ and $S_{i,j}^{(3)}$. To make the parallel computation of $S_i^{(2)}$ possible we separate

these sums into parts corresponding to the partitions as

$$
\begin{aligned}
S_i^{(2)} &= \sum_{\ell=1}^{L-1}\sum_{u=1}^{K} a_i[(\ell-1)K+u-1]x_{(\ell-1)K+u}b_i[(\ell-1)K+u] \\
&+ \sum_{u=1}^{T-(L-1)K} a_i[(L-1)K+u-1]x_{(L-1)K+u}b_i[(L-1)K+u] \\
&= \sum_{\ell=1}^{L-1} \underline{\pi}^{\ell-1} \underbrace{\sum_{u=1}^{K} \vec{\boldsymbol{P}}[(\ell-1)K+1,(\ell-1)K+u-1]e_i^T x_{(\ell-1)K+u} e_i \vec{\boldsymbol{P}}[(\ell-1)K+u,\ell K]}_{\bar{\boldsymbol{\Omega}}_i^{(2)}[\ell,K]=\boldsymbol{\Omega}_i^{(2)}[\ell]} \mathbb{1}^{\ell+1} \\
&+ \underline{\pi}^{L-1} \underbrace{\sum_{u=1}^{T-(L-1)K} \vec{\boldsymbol{P}}[(L-1)K+1,(L-1)K+u-1]e_i^T x_{(L-1)K+u} e_i \vec{\boldsymbol{P}}[(L-1)K+u,T]}_{\bar{\boldsymbol{\Omega}}_i^{(2)}[L,T-(L-1)K]=\boldsymbol{\Omega}_i^{(2)}[L]} \mathbb{1},
\end{aligned}
\tag{25}
$$

where

$$
\bar{\boldsymbol{\Omega}}_i^{(2)}[\ell,z] = \sum_{u=1}^{z} \vec{\boldsymbol{P}}[(\ell-1)K+1,(\ell-1)K+u-1]e_i^T x_{(\ell-1)K+u} e_i \vec{\boldsymbol{P}}[(\ell-1)K+u,(\ell-1)K+z]
$$

and we made use of (19). The first (double) summation corresponds to partitions $1,\ldots,L-1$ and the second one to the last (Lth) partition, having potentially less than K elements. Matrices $\bar{\boldsymbol{\Omega}}_i^{(2)}[\ell,z]$ represent the sum in partition ℓ up to term z, while we introduce the shorthand notation $\boldsymbol{\Omega}_i^{(2)}[\ell]$ to denote the sum over all terms of partition ℓ. The main observation is that $\boldsymbol{\Omega}_i^{(2)}[\ell]$ can be computed simultaneously for each partition.

Similarly, for $S_{i,j}^{(3)}$, we have

$$
\begin{aligned}
S_{i,j}^{(3)} &= \sum_{\ell=1}^{L-1}\sum_{u=1}^{K} a_i[(\ell-1)K+u-1]\boldsymbol{P}[(\ell-1)K+u]_{i,j}b_j[(\ell-1)K+u+1] \\
&+ \sum_{u=1}^{T-(L-1)K-1} a_i[(L-1)K+u-1]\boldsymbol{P}[(L-1)K+u]_{i,j}b_j[(L-1)K+u+1] \\
&= \sum_{\ell=1}^{L-1} \underline{\pi}^{\ell-1} \underbrace{\sum_{u=1}^{K} \vec{\boldsymbol{P}}[(\ell-1)K+1,(\ell-1)K+u-1]e_i^T e_i \boldsymbol{P}[(\ell-1)K+u]e_j^T e_j \vec{\boldsymbol{P}}[(\ell-1)K+u+1,\ell K]}_{\bar{\boldsymbol{\Omega}}_{i,j}^{(3)}[\ell,K]=\boldsymbol{\Omega}_{i,j}^{(3)}[\ell]} \mathbb{1}^{\ell+1} \\
&+ \underline{\pi}^{L-1} \underbrace{\sum_{u=1}^{T-(L-1)K-1} \vec{\boldsymbol{P}}[(L-1)K+1,(L-1)K+u-1]e_i^T e_i \boldsymbol{P}[(L-1)K+u]e_j^T e_j \vec{\boldsymbol{P}}[(L-1)K+u+1,T]}_{\bar{\boldsymbol{\Omega}}_{i,j}^{(3)}[L,T-(L-1)K-1]=\boldsymbol{\Omega}_{i,j}^{(3)}[L]} \mathbb{1},
\end{aligned}
\tag{26}
$$

where

$$
\bar{\boldsymbol{\Omega}}_{i,j}^{(3)}[\ell,z] = \sum_{u=1}^{z} \vec{\boldsymbol{P}}[(\ell-1)K+1,(\ell-1)K+u-1]e_i^T e_i \boldsymbol{P}[(\ell-1)K+u]e_j^T e_j \vec{\boldsymbol{P}}[(\ell-1)K+u+1,(\ell-1)K+z].
$$

The next theorem provides an efficient way to compute matrices $\boldsymbol{\Omega}_i^{(2)}[\ell]$ and $\boldsymbol{\Omega}_{i,j}^{(3)}[\ell]$ by forward-only recursions. Similar recursive computation of cumulated measures is used in [20] for hidden Markov chain fitting.

Theorem 1. *For $0 < z \leq K$ and $\ell < L$, as well as for $0 < z \leq T - (L-1)K$ and $\ell = L$, matrices $\bar{\mathbf{\Omega}}_i^{(2)}[\ell, z]$ and $\bar{\mathbf{\Omega}}_{i,j}^{(3)}[\ell, z]$, satisfy the recursive relations*

$$\bar{\mathbf{\Omega}}_i^{(2)}[\ell, z] = \bar{\mathbf{\Omega}}_i^{(2)}[\ell, z-1]\boldsymbol{P}[(\ell-1)K + z] + \vec{\boldsymbol{P}}[(\ell-1)K+1, (\ell-1)K+z-1]e_i^T x_z e_i \boldsymbol{P}[(\ell-1)K+z]$$

and

$$\bar{\mathbf{\Omega}}_{i,j}^{(3)}[\ell, z] = \bar{\mathbf{\Omega}}_{i,j}^{(3)}[\ell, z-1]\boldsymbol{P}[(\ell-1)K + z] + \vec{\boldsymbol{P}}[(\ell-1)K+1, (\ell-1)K+z-1]e_i^T e_i \boldsymbol{P}[(\ell-1)K+z]e_j^T e_j$$

with initial value $\bar{\mathbf{\Omega}}_i^{(2)}[\ell, 0] = \mathbf{0}$ and $\bar{\mathbf{\Omega}}_{i,j}^{(3)}[\ell, 0] = \mathbf{0}$.

Proof. Starting with the definition of matrix $\bar{\mathbf{\Omega}}_i^{(2)}[\ell, z]$ and separating the last term of the sum as

$$\begin{aligned}
\bar{\mathbf{\Omega}}_i^{(2)}[\ell, z] &= \sum_{u=1}^{z} \vec{\boldsymbol{P}}[(\ell-1)K+1, (\ell-1)K+u-1]e_i^T x_{(\ell-1)K+u} e_i \vec{\boldsymbol{P}}[(\ell-1)K+u, (\ell-1)K+z] \\
&= \sum_{u=1}^{z-1} \vec{\boldsymbol{P}}[(\ell-1)K+1, (\ell-1)K+u-1]e_i^T x_{(\ell-1)K+u} e_i \vec{\boldsymbol{P}}[(\ell-1)K+u, (\ell-1)K+z-1] \\
&\quad + \vec{\boldsymbol{P}}[(\ell-1)K+1, (\ell-1)K+z-1]e_i^T x_{(\ell-1)K+z} e_i \boldsymbol{P}[(\ell-1)K+z] \\
&= \bar{\mathbf{\Omega}}_i^{(2)}[\ell, z-1]\boldsymbol{P}[(\ell-1)K+z] + \vec{\boldsymbol{P}}[(\ell-1)K+1, (\ell-1)K+z-1]e_i^T x_z e_i \boldsymbol{P}[(\ell-1)K+z]
\end{aligned}$$

proves the recursive relation for matrices $\bar{\mathbf{\Omega}}_i^{(2)}[\ell, z]$. The relation for matrices $\bar{\mathbf{\Omega}}_{i,j}^{(3)}[\ell, z]$ can be proven similarly. □

With these notations, the algorithm that takes a single pass through the data set consists of the following two phases:

1. The *parallel* computation of matrices $\boldsymbol{U}[\ell]$, $\mathbf{\Omega}_i^{(2)}[\ell]$ and $\mathbf{\Omega}_{i,j}^{(3)}[\ell]$ for $\ell = 1, \ldots, L$ such that each partition is computed by a different parallel thread.

Algorithm 1 Pseudo-code for partition ℓ ($\ell < L$) in the 1-pass method

```
1: procedure PROCESS PARTITION ℓ(x_u, u = (ℓ−1)K+1, ..., ℓK, r, λ, Π)
2:     U = I
3:     ∀i: Ω_i^(2) = 0
4:     ∀i, j: Ω_{i,j}^(3) = 0
5:     for z = 1 to K do
6:         M = P[(ℓ−1)K + z]
7:         ∀i: Ω_i^(2) = Ω_i^(2) M + U x_{(ℓ−1)K+z} e_i^T e_i M
8:         ∀i, j: Ω_{i,j}^(3) = Ω_{i,j}^(3) M + U e_i^T e_i M e_j^T e_j
9:         U = Normalize(UM)
10:    end for
11:    return U, Ω_i^(2), Ω_{i,j}^(3)
12: end procedure
```

Algorithm 1 provides the formal description of the steps for $\ell < L$. For partition L the procedure differs only by the range of the for loop (z goes from 1 to $T - (L-1)K$) and by the fact that $\mathbf{\Omega}_{i,j}^{(3)}$ is summed only up to $T - 1$. Operation *Normalize*() in line 9 applies a special scaling on the matrix to improve the numerical behavior, as detailed in Section 4.1.

2. The *serial* computation of the forward and backward likelihood vectors $\underline{\pi}^\ell$ and $\mathbb{1}^\ell$, for $\ell = 1, \ldots, L$, recursively according to

$$\underline{\pi}^\ell = \begin{cases} \underline{\pi}, & \text{if } \ell = 0, \\ \underline{\pi}^{\ell-1}\boldsymbol{U}[\ell], & \text{if } \ell \geq 1, \end{cases}, \quad \text{and} \quad \mathbb{1}^\ell = \begin{cases} \mathbb{1}, & \text{if } \ell = L+1, \\ \underline{\pi}^{\ell-1}\boldsymbol{U}[\ell], & \text{if } \ell \geq 1, \end{cases} \tag{27}$$

and the sums $S_i^{(2)}$ and $S_{i,j}^{(3)}$ for $i,j=1,\ldots,R$ according to

$$S_i^{(2)} = \sum_{\ell=1}^{L} \underline{\pi}^{\ell-1} \mathbf{\Omega}_i^{(2)}[\ell] \mathbb{1}^{\ell+1}, \;\; S_{i,j}^{(3)} = \sum_{\ell=1}^{L} \underline{\pi}^{\ell-1} \mathbf{\Omega}_{i,j}^{(3)}[\ell] \mathbb{1}^{\ell+1}, \tag{28}$$

from which λ_i', $p_{i,j}'$ and π_i' are obtained by (20), (21), and (22).

For emphasizing the special step due to the fact that $\mathbf{\Omega}_{i,j}^{(3)}$ is summed only up to $T-1$, we note that in order to compute $S_i^{(1)}$ from $S_i^{(3)}$ it is necessary to compute vectors $\underline{a}[T-1]$ and $\underline{b}[T]$, that can be obtained from $\underline{\pi}^{L-1}$ and $\mathbb{1}^L$ with negligible extra cost.

The computational bottleneck of the procedure is that matrices $\mathbf{\Omega}_{i,j}^{(3)}$ must be computed for all partitions, which means that the memory complexity is $\mathcal{O}(LR^4)$ and the computational complexity is $\mathcal{O}(KR^5)$ in each thread due to the matrix-matrix multiplications in line 8 of Algorithm 1. A more detailed complexity analysis is provided later in Section 4.2. This algorithm, described formally in Algorithm 2, will be referred to as `P-1` in the sequel.

Algorithm 2 Pseudo-code of algorithm `P-1`

```
1: procedure EM-FITTING BY P-1(x_u, u = 1,...,T, r)
2:     λ, Π = random initial guess
3:     while relative change of log-likelihood > ε do
4:         parallel for ℓ = 1 to L do
5:             Compute matrices U[ℓ], Ω_i^(2)[ℓ] and Ω_{i,j}^(3)[ℓ] by Algorithm 1
6:         end parallel for
7:         for ℓ = 1 to L do
8:             Compute vectors π^ℓ and 1^ℓ for ℓ = 1,...,L based on (27)
9:         end for
10:        Compute sums S_i^(1), S_i^(2), S_i^(3) and S_{i,j}^(3) for i,j = 1,...,R based on (28), (23) and (24)
11:        λ, Π = new estimates based on (12), (13) and (14)
12:        Compute the log-likelihood
13:    end while
14:    return λ, Π
15: end procedure
```

3.3. The parallel implementation with two passes

The single pass algorithm in Section 3.2 goes through the inter-arrival times only once in each iteration. If traversing the input twice does not have an overwhelming extra cost, it is possible to develop a variant of the algorithm with different performance characteristics.

The main drawback of the single-pass algorithm is that the computation of matrices $\mathbf{\Omega}_i^{(2)}$ and $\mathbf{\Omega}_{i,j}^{(3)}$ requires matrix-matrix multiplications and a significant amount of memory is required to store them for every partition. The main improvement of the two-pass algorithm is that these matrices are replaced by vectors during the second pass of the computation.

A single EM-iteration of the two-pass algorithm consists of the following phases:

1. The *parallel* computation of matrices $\boldsymbol{U}[\ell]$, for $\ell=1,\ldots,L$, based on $\boldsymbol{U}[\ell] = \prod_{u=1}^{K} \boldsymbol{P}[(\ell-1)K+u]$, such that each partition is computed by a different thread simultaneously. More precisely, $\boldsymbol{U}[\ell]$ is computed according to Algorithm 3.

Algorithm 3 Pseudo-code for computing $\boldsymbol{U}[\ell]$ ($\ell < L$) in the two-pass method

```
1: procedure PROCESS PARTITION ℓ(x_u, u = (ℓ−1)K+1,...,ℓK, r, λ, Π)
2:     U = I
3:     for z = 1 to K do
4:         U = Normalize(UP[(ℓ−1)K+z])
5:     end for
6:     return U
7: end procedure
```

2. The *serial* computation of the forward and backward likelihood vectors $\underline{\pi}^\ell$ and $\mathbb{1}^\ell$ for the partitions $\ell = 1, \ldots, L$ recursively according to

$$\underline{\pi}^\ell = \begin{cases} \underline{\pi}, & \text{if } \ell = 0, \\ \underline{\pi}^{\ell-1}\boldsymbol{U}[\ell], & \text{if } \ell \geq 1, \end{cases}, \text{ and } \quad \mathbb{1}^\ell = \begin{cases} \mathbb{1}, & \text{if } \ell = L+1, \\ \boldsymbol{U}[\ell]\mathbb{1}^{\ell+1}, & \text{if } \ell \leq L. \end{cases} \tag{29}$$

3. The *parallel* computation of vectors $\underline{\omega}_i^{(2)}[\ell] = \underline{\pi}^\ell \boldsymbol{\Omega}_i^{(2)}[\ell]$ and $\underline{\omega}_{i,j}^{(3)}[\ell] = \underline{\pi}^\ell \boldsymbol{\Omega}_{i,j}^{(3)}[\ell]$ for $\ell = 1, \ldots, L$ such that each partition is computed by a different thread. The details are provided in Algorithm 4.

Algorithm 4 Pseudo-code for computing $\underline{\omega}_i^{(2)}[\ell]$ and $\underline{\omega}_{i,j}^{(3)}[\ell]$ ($\ell < L$) in the two-pass method

1: **procedure** PROCESS PARTITION $\ell(x_u, u = (\ell-1)K+1, \ldots, \ell K, \underline{\pi}^{\ell-1}, \underline{r}, \underline{\lambda}, \boldsymbol{\Pi})$
2: $\quad \underline{u} = \underline{\pi}^{\ell-1}$
3: $\quad \forall i$: $\underline{\omega}_i^{(2)} = \underline{0}$
4: $\quad \forall i, j$: $\underline{\omega}_{i,j}^{(3)} = \underline{0}$
5: $\quad$ **for** $z = 1$ **to** K **do**
6: $\quad\quad \boldsymbol{M} = \boldsymbol{P}[(\ell-1)K+z]$ (based on the stored or recomputed $f_i(x_{(\ell-1)K+z})$ values)
7: $\quad\quad \forall i$: $\underline{\omega}_i^{(2)} = \underline{\omega}_i^{(2)}\boldsymbol{M} + \underline{u}x_z e_i^T e_i \boldsymbol{M}$
8: $\quad\quad \forall i, j$: $\underline{\omega}_{i,j}^{(3)} = \underline{\omega}_{i,j}^{(3)}\boldsymbol{M} + \underline{u}e_i^T e_i \boldsymbol{M} e_j^T e_j$
9: $\quad\quad \underline{u} = \mathit{Normalize}(\underline{u}M)$
10: $\quad$ **end for**
11: $\quad$ **return** $\underline{\omega}_i^{(2)}, \underline{\omega}_{i,j}^{(3)}$.
12: **end procedure**

Again, for partition L the procedure differs by the range of the for loop and by the fact that $\underline{\omega}_{ij}^{(3)}$ is summed only up to $T-1$.

4. The *serial* computation of the sums $S_i^{(2)}$ and $S_{i,j}^{(3)}$ for $i, j = 1, \ldots, R$ according to

$$S_i^{(2)} = \sum_{\ell=1}^{L} \underline{\omega}_i^{(2)}[\ell]\mathbb{1}^{\ell+1}, \; S_{ij}^{(3)} = \sum_{\ell=1}^{L} \underline{\omega}_{ij}^{(3)}[\ell]\mathbb{1}^{\ell+1}, \tag{30}$$

from which λ_i', $p_{i,j}'$ and π_i' are obtained by (20), (21), and (22).

With this method, a computational bottleneck of the single-pass algorithm has been eliminated. Instead of matrices, only the vectors $\underline{\omega}_{i,j}^{(3)}$ are stored and used during the second parallel pass of the computations (i.e., in phase 3), which means that the memory complexity is now $\mathcal{O}(LR^3)$ and the computational complexity in each thread is $\mathcal{O}(KR^4)$ due to the vector-matrix multiplications in line 8 of Algorithm 4. A more detailed complexity analysis is provided later in Section 4.2. This algorithm is referred to as `P-2` in the sequel; the corresponding pseudo-code is provided by Algorithm 5.

Algorithm 5 Pseudo-code of algorithm `P-2`

1: **procedure** EM-FITTING BY `P-2`$(x_u, u = 1, \ldots, T, \underline{r})$
2: $\quad\underline{\lambda}, \boldsymbol{\Pi}$ = random initial guess
3: $\quad$**while** relative change of log-likelihood $> \epsilon$ **do**
4: $\quad\quad$**parallel for** $\ell = 1$ **to** L **do**
5: $\quad\quad\quad$Compute matrices $\boldsymbol{U}[\ell]$ by Algorithm 3
6: $\quad\quad$**end parallel for**
7: $\quad\quad$**for** $\ell = 1$ **to** L **do**
8: $\quad\quad\quad$Compute vectors $\underline{\pi}^{\ell}$ and $\mathbb{1}^{\ell}$ for $\ell = 1, \ldots, L$ based on (29)
9: $\quad\quad$**end for**
10: $\quad\quad$**parallel for** $\ell = 1$ **to** L **do**
11: $\quad\quad\quad$Compute vectors $\underline{\omega}_i^{(2)}[\ell]$, $\underline{\omega}_{i,j}^{(3)}[\ell]$ for $i, j = 1, \ldots, R$ by Algorithm 4
12: $\quad\quad$**end parallel for**
13: $\quad\quad$Compute sums $S_i^{(1)}, S_i^{(2)}, S_i^{(3)}$ and $S_{i,j}^{(3)}$ for $i, j = 1, \ldots, R$ based on (30), (23) and (24)
14: $\quad\quad\underline{\lambda}, \boldsymbol{\Pi}$ = new estimates based on (12), (13) and (14)
15: $\quad\quad$Compute the log-likelihood
16: $\quad$**end while**
17: $\quad$**return** $\underline{\lambda}, \boldsymbol{\Pi}$
18: **end procedure**

The single-pass algorithm evaluates the branch densities $f_i(x_u)$ (see (4)) only once for each inter-arrival time (when creating matrix $\boldsymbol{P}$ in line 6 of Algorithm 1), while the two-pass algorithm needs these densities twice. In the first pass (phase 1) these densities are needed to obtain the likelihood matrices $\boldsymbol{U}[\ell]$, and in the second pass (phase 3) they are needed for the calculation of the $\underline{\omega}$ vectors. Hence, to save computational effort, it makes sense to compute the branch densities only once and store them in an auxiliary vector, which comes at the expense of increased memory requirement of storing TR floating point numbers. The variant of the algorithm which computes the branch densities only once and stores them is referred to as `P-2-D` hereafter.

3.4. The parallel implementation with three passes

The single pass and the two-pass algorithms are similar in spirit to each other, in the sense that they recursively compute partition based cumulated measures (matrix $\boldsymbol{\Omega}_i^{(2)}[\ell]$ and $\boldsymbol{\Omega}_{i,j}^{(3)}[\ell]$ in the one-pass algorithm, and vector $\underline{\omega}_i^{(2)}[\ell]$ and $\underline{\omega}_{i,j}^{(3)}[\ell]$ in the two-pass algorithm). The algorithm introduced in this section differs significantly. In fact, this algorithm is rather similar to the naive sequential method (Section 3.1), which does not apply recursively computed partition based cumulated measures, but computes $S_i^{(2)}[\ell]$ and $S_{i,j}^{(3)}[\ell]$ directly from the forward and backward likelihood vectors.

The phases of the algorithm are as follows:

1. The *parallel* computation of matrices $\boldsymbol{U}[\ell]$, for $\ell = 1, \ldots, L$, based on $\boldsymbol{U}[\ell] = \prod_{u=1}^{K} \boldsymbol{P}[(\ell-1)K + u]$, such that each partition is computed by a different thread simultaneously. This phase is the same as the first phase in the two-pass method.
2. The *serial* computation of the forward and backward likelihood vectors $\underline{\pi}^{\ell}$ and $\mathbb{1}^{\ell}$ for the partitions $\ell = 1, \ldots, L$ according to recursions

$$\underline{\pi}^{\ell} = \begin{cases} \underline{\pi}, & \text{if } \ell = 0, \\ \underline{\pi}^{\ell-1}\boldsymbol{U}[\ell], & \text{if } \ell \geq 1, \end{cases} \text{, and } \quad \mathbb{1}^{\ell} = \begin{cases} \mathbb{1}, & \text{if } \ell = L+1, \\ \boldsymbol{U}[\ell]\mathbb{1}^{\ell+1}, & \text{if } \ell \leq L. \end{cases} \tag{31}$$

3. The *parallel* computation of the individual likelihood vectors $\underline{a}[u]$ and $\underline{b}[u]$ belonging to partition ℓ for $\ell = 1, \ldots, L$ based on

$$\underline{a}[u] = \underline{\pi}^{\ell}\vec{\boldsymbol{P}}[(\ell-1)K + 1, u], \text{ and } \underline{b}[u] = \vec{\boldsymbol{P}}[u, \ell K]\mathbb{1}^{\ell+1}, \tag{32}$$

such that each partition is computed by a different thread. Again, for the last partition L the procedure differs a bit by the number of samples. More precisely $\underline{a}[u]$ for $u = (\ell-1)K + 1, \ldots, \ell K$ is computed as in Algorithm 6 and $\underline{b}[u]$ is computed in a similar manner with backward iteration.

Algorithm 6 Pseudo-code for computing $\underline{a}[u]$ ($\ell < L$, $u = (\ell-1)K+1, \ldots, \ell K$) in the three-pass method

procedure PROCESS PARTITION $\ell(x_u, u = (\ell-1)K+1, \ldots, \ell K, \underline{\pi}^{\ell-1}, \underline{r}, \underline{\lambda}, \mathbf{\Pi})$
  $\underline{a}[0] = \underline{\pi}^{\ell-1}$
  for $z = 1$ **to** K **do**
    $\underline{a}[z] = \mathit{Normalize}(\underline{a}[z-1]\boldsymbol{P}[(\ell-1)K+z])$ (based on the stored or recomputed $f_i(x_{(\ell-1)K+z})$ values)
  end for
  return $\underline{a}$.
end procedure

4. The *parallel* computation of the partition related partial sums $S_i^{(2)}[\ell]$ and $S_{i,j}^{(3)}[\ell]$ according to

$$\begin{aligned} S_i^{(2)}[\ell] &= \sum_{u=1}^{K} a_i[(\ell-1)K+u-1]x_{(\ell-1)K+u}b_i[(\ell-1)K+u] \\ S_{i,j}^{(3)}[\ell] &= \sum_{u=1}^{K} a_i[(\ell-1)K+u-1]\boldsymbol{P}[(\ell-1)K+u]_{i,j}b_j[(\ell-1)K+u+1], \end{aligned} \tag{33}$$

   using the stored $\underline{a}[u]$, $\underline{b}[u]$ values.
5. The *serial* computation of the sums $S_i^{(2)}$ and $S_{i,j}^{(3)}$ according to

$$S_i^{(2)} = \sum_{\ell=1}^{L} S_i^{(2)}[\ell], \text{ and } \quad S_{ij}^{(3)} = \sum_{\ell=1}^{L} S_{ij}^{(3)}[\ell], \tag{34}$$

   from which λ_i', $p_{i,j}'$ and π_i' are obtained by (20), (21), and (22).

The computational bottleneck of this procedure is phase 1, that consists of matrix-matrix multiplications of size R, leading to $\mathcal{O}(KR^3)$ for each parallel thread. However, the memory consumption has increased significantly compared to the previously described `P-1` and `P-2` methods. Vectors $\underline{a}[u]$ and $\underline{b}[u]$ need to be stored for each inter-arrival time, that gives $2TR$ floating point numbers. Note that the memory consumption of neither `P-1` nor `P-2` is proportional to T in a direct way, they are proportional only to the number of partitions L.

Algorithm 7 Pseudo-code of algorithm `P-2`

procedure EM-FITTING BY `P-3`$(x_u, u = 1, \ldots, T, \underline{r})$
  $\underline{\lambda}, \mathbf{\Pi}$ = random initial guess
  while relative change of log-likelihood $> \epsilon$ **do**
    parallel for $\ell = 1$ **to** L **do**
      Compute matrices $\boldsymbol{U}[\ell]$ by Algorithm 3
    end parallel for
    for $\ell = 1$ **to** L **do**
      Compute vectors $\underline{\pi}^{\ell}$ and $\mathbb{1}^{\ell}$ for $\ell = 1, \ldots, L$ based on (31)
    end for
    parallel for $\ell = 1$ **to** L **do**
      Compute vectors $\underline{a}[u]$ and $\underline{b}[u]$ for $u = (\ell-1)K+1, \ldots, \ell K$, by Algorithm 6
    end parallel for
    parallel for $\ell = 1$ **to** L **do**
      Compute partition sums $S_i^{(2)}[\ell]$ and $S_{i,j}^{(3)}[\ell]$ according to (33)
    end parallel for
    Compute sums $S_i^{(1)}, S_i^{(2)}, S_i^{(3)}$ and $S_{i,j}^{(3)}$ for $i, j = 1, \ldots, R$ based on (30), (23) and (24)
    $\underline{\lambda}, \mathbf{\Pi}$ = new estimates based on (12), (13) and (14)
    Compute the log-likelihood
  end while
  return $\underline{\lambda}, \mathbf{\Pi}$
end procedure

This procedure consists of three parallel phases through the data set (phase 1, phase 3 and phase 4 are parallel), hence we refer to it as the 3-pass algorithm, denoted as `P-3`, and describe it in Algorithm

7. In this procedure the branch densities need to be evaluated three times (in phases 1, 3 and 4), thus the execution time can benefit from computing the branch densities once and storing them similar to the two-pass algorithm. This variant of the algorithm, called `P-3-D`, has an even higher memory requirement.

4. Computational details and performance comparison

4.1. Avoiding of the numerical difficulties

It is widely known that computing the likelihood for a long sequence of observations is often affected by numerical issues. The common "trick" to overcome these issues is to compute the logarithm of the likelihood instead. As long as the observations are independent, the log-likelihood can be easily obtained by simple summation, as it was done in many EM algorithms for PH distributions, including [1] and [19].

Unfortunately, in case of MAPs, the likelihood function is obtained through matrix multiplications (see (8) and (10)), whose logarithm cannot be obtained by a simple summation and we need to cope with the arising numerical issues. Namely, due to the finite resolution of the machine representation of the floating point numbers, the large number of matrix multiplications can cause underflow or overflow.

The EM algorithm for MAP fitting is a special case of this numerical phenomena; due to the definition of the forward and backward likelihood vectors (11) exponential functions are multiplied a large number of times (T) which makes the occurrence of underflow or overflow almost sure. A common way to overcome this numerical issue is to adopt an appropriate scaling technique. We adopt the scaling technique introduced and successfully used for TMAP fitting in [3], for a similar EM-based, albeit serial, algorithm.

First of all, we note that all entries of the likelihood vectors and matrices are non-negative, and their elements are only multiplied and added during the computation, which eliminates the problem of considering numbers with a negative sign. In order to avoid underflow and overflow problems of floating point numbers, we represent each matrix of size $n \times m$ by $nm + 1$ values in the form of

$$\boldsymbol{A} = \mathring{\boldsymbol{A}} \cdot 2^{\dot{a}} = \{\mathring{a}_{i,j}\} \cdot 2^{\dot{a}},$$

and we say that the representation is normalized when $0.5 < \max_{i,j} \mathring{a}_{i,j} \leq 1$ holds.

The multiplication with such a matrix representation is straightforward, since the i, j element of matrix $\boldsymbol{AB}$ is computed as

$$(\boldsymbol{AB})_{ij} = \sum_k \mathring{a}_{i,k}\mathring{b}_{k,j} \cdot 2^{\dot{a}+\dot{b}} = \mathring{c}_{i,j} \cdot 2^{\dot{c}}.$$

If the representations of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are normalized then the resulting $\mathring{c}_{i,j} = \sum_k \mathring{a}_{i,k}\mathring{b}_{k,j}$, $\dot{c} = \dot{a} + \dot{b}$ representation does not necessarily satisfy $0.5 < \max_{i,j} \mathring{c}_{i,j} \leq 1$, i.e., the results is not necessarily normalized.

The same applies for matrix summation. For computing matrix $\boldsymbol{A} + \boldsymbol{B}$, let $\hat{c} = \max(\dot{a}, \dot{b})$ be the biggest of the two exponents. We compute the (i, j) element of matrix $\boldsymbol{A} + \boldsymbol{B}$ as

$$(\boldsymbol{A} + \boldsymbol{B})_{ij} = \left(\mathring{a}_{i,j} \cdot 2^{\dot{a}-\hat{c}} + \mathring{b}_{i,j} \cdot 2^{\dot{b}-\hat{c}}\right) \cdot 2^{\hat{c}} = \mathring{c}_{i,j} \cdot 2^{\dot{c}}.$$

Similar to the result of matrix multiplication the obtained representation is not necessarily normalized.

To obtain a normalized representation from any $\{\mathring{c}_{i,j}\} \cdot 2^{\dot{c}}$ representation let $c_n = \lfloor \log_2(\max_{i,j} \mathring{c}_{i,j}) \rfloor$ then the normalized representation is

$$\mathring{c}_{i,j} \leftarrow \mathring{c}_{i,j} 2^{-1-c_n} \text{ and } \dot{c} \leftarrow \dot{c} + 1 + c_n. \tag{35}$$

This computational step is referred to as *Normalize*() in the description of the algorithms in Section 3. Since n and m in the matrix dimension can be equal to one, the same scaling technique applies for vectors as well.

There is an other numerical pitfall in the algorithms, related to floating point summations involving terms with different orders of magnitudes (for instance, in (12), (13), and (14) the sum of T values is computed). The issue occurs when the sum becomes too large and does not change when further small values are added

one-by-one, even if the sum of these small values is not negligible. The root of the problem is that when two numbers $x = \mathring{x} \cdot 2^{\dot{x}}$ and $y = \mathring{y} \cdot 2^{\dot{y}}$ having different orders of magnitudes are added together, some precision is lost. If $x > y$, the exponent of the sum $z = x + y = \mathring{z} \cdot 2^{\dot{z}}$ will be $\dot{z} = \dot{x}$, and the mantissa will be $\mathring{z} = \mathring{x} + \mathring{y} \cdot 2^{-(\dot{x}-\dot{y})}$. Since the number of bits available to represent the mantissa $\mathring{z}$ is fixed, $\dot{x} - \dot{y}$ precious bits will be lost from $\mathring{y}$.

To improve the accuracy of such numerically sensitive summations, we apply the following simple solution. When a new number y is to be added to the sum x, the loss of precision is calculated as $\phi = |\log_2 x/y|$. When ϕ is greater than a predefined threshold, y is not added to x. Instead, a new accumulator variable is created (initialized to zero) and the further terms (including y) are added to the newly created accumulator variable, as long as the precision loss ϕ is below the threshold. In the end, all the accumulator variables are summed up using the same technique.

4.2. Detailed comparison of the complexities

Table 1: The memory consumption of the algorithms measured in floating point values to store

Data	serial FB	`P-1`	`P-2`	`P-3`
parameters $\mathbf{\Pi}$, $\underline{r}$, $\underline{\lambda}$, $\underline{\pi}$	$R^2 + 3R$	$R^2 + 3R$	$R^2 + 3R$	$R^2 + 3R$
inter-arrival times x_u	T	T	T	T
sums $S_{ij}^{(3)}$, $S_i^{(n)}, n{=}1,2,3$	$R^2 + 3R$	$R^2 + 3R$	$R^2 + 3R$	$R^2 + 3R$
vectors $\underline{\pi}^\ell$, $\mathbb{1}^\ell$		$2LR(+2L)$	$2LR(+2L)$	$2LR(+2L)$
matrices $\boldsymbol{U}[\ell]$		$LR^2(+L)$	$LR^2(+L)$	$LR^2(+L)$
partial sums		$\mathbf{\Omega}_i^{(2)}[\ell]$: $LR^3(+L)$	$\underline{\omega}_i^{(2)}[\ell]$: $LR^2(+L)$	$S_i^{(2)}[\ell]$: $LR(+L)$
partial sums		$\mathbf{\Omega}_{i,j}^{(3)}[\ell]$: $LR^4(+L)$	$\underline{\omega}_{i,j}^{(3)}[\ell]$: $LR^3(+L)$	$S_{i,j}^{(3)}[\ell]$: $LR^2(+L)$
vectors $\underline{a}[u]$, $\underline{b}[u]$	$2TR(+2T)$			$2TR(+2T)$
branch densities $f_i(x)$	TR		`P-2-D` only: TR	`P-3-D` only: TR

Table 1 depicts the memory consumption of the algorithms presented in Section 3.1 - 3.4. In case of the serial forward-backward (FB) method (Section 3.1), the branch density vectors and the likelihood vectors occupy the majority of the memory space. The $+2T$ term in the parentheses corresponds to the scaling exponents ($\dot{a}$) introduced in Section 4.1. Similarly, the memory requirement of the scaling exponents is given in parentheses in the consecutive columns.

In case of the single-pass method (Section 3.2), on the one hand, storing matrices $\mathbf{\Omega}_i^{(2)}[\ell]$ and $\mathbf{\Omega}_{i,j}^{(3)}[\ell]$ for each partition can be very memory consuming when R and L are large; on the other hand, the only component that directly depends on T is the space occupied by the inter-arrival times. Interestingly, as $R << L << T$ typically holds, `P-1` algorithm needs much less memory than the serial FB algorithm (and at the same time, as shown later, provides much better execution speed).

Compared to the `P-1` algorithm, the memory requirement of the `P-2` algorithm (Section 3.3) is identical in all components except the partial sums whose memory consumption reduces by a factor of R. The memory consumption of the `P-2-D` variant, which is expected to improve on the running times by caching the branch densities, is still better than the one of the serial FB algorithm (in case of the typical setting: $R << L << T$).

The methods `P-3` and `P-3-D` (introduced in Section 3.4) are the most memory intensive ones among the parallel algorithms due to the storage of the likelihood vectors, $\underline{a}[u]$ and $\underline{b}[u]$ for $u = 1, \ldots, T$. In the `P-3-D` variant the branch densities increase the memory consumption even further.

The comparison of the computational complexities is more difficult because of the algorithms are composed of multiple parallel and serial computation steps. For a reasonable comparison, we divide the parallel executed computation steps by the number of parallel threads. This approach might be inaccurate in some specific computational environments, e.g., with high overhead of parallel execution, but without focusing on a particular computational infrastructure, we resort to the assumption that parallel execution with L threads is exactly L times faster than the serial one.

The `orders` of the complexity of the algorithms are summarized in Table 2, where the rightmost column refers to serial (S) and parallel execution in pass one (P1), two (P2) and three (P3). The table displays only the order of the highest order terms.

Table 2: The `order` of computational complexity of the algorithms measured in floating point multiplications

Data to compute	serial FB	`P-1`	`P-2-D`	`P-3-D`	
branch densities $f_i(x)$	RT	RT/L	RT/L	RT/L	
matrices $\boldsymbol{U}[\ell]$		R^3T/L	R^3T/L	R^3T/L	P1
partial sums		$\boldsymbol{\Omega}^{(3)}_{i,j}[\ell]$: R^5T/L			
vectors $\underline{\pi}^\ell$, $\mathbb{1}^\ell$		R^2L	R^2L	R^2L	S
branch densities $f_i(x)$			`P-2` only: RT/L	`P-3` only: RT/L	
partial sums			$\underline{\omega}^{(3)}_{i,j}[\ell]$: R^4T/L		P2
$\underline{a}[u]$, $\underline{b}[u]$	R^2T			R^2T/L	
branch densities $f_i(x)$				`P-3` only: RT/L	P3
partial sums				$S^{(3)}_{i,j}[\ell]$: R^2T/L	
$\boldsymbol{\Pi}$, $\underline{\lambda}$, $\underline{\pi}$	R^2T	R^4L	R^3L		S

Similar to the memory consumption, the relation of the three characterizing parameters R, L, and T determines the relation of the computational complexity of the algorithms. One can draw general consequences based on the tables of memory consumption and execution time. E.g., when a sufficiently large memory is available, then the `P-3-D` method gives the lowest computation time order, but already this basic statement needs to be handled with care. For example, when R is small (e.g., $R = 2$) the difference between R^2 and R^4 might be negligible with respect to other constant terms of the execution time which remain hidden in the table of computational complexity `orders`.

However, there is an even more dominant factor of the computation time, the hardware and software implementation of the algorithms. One of the most decisive elements with this respect is the memory access times of different computing units, which is known to affect the running time significantly. The next section summarizes our numerical experiments, which essentially follows the main trends presented in this section, but shows particular differences in some cases.

5. Numerical experiments

We have implemented the algorithms in C++ language. All the developed procedures use single precision floating point numbers for storing real values (i.e., the type of 'float' in C++). The serial procedure is executed on the CPU (i5-4690, 3.5GHz) and the parallel procedures, in addition, utilize a GeForce GTX 1070 graphics processing unit (GPU) having $L^* = 1920$ parallel processing elements clocked at 1.5GHz and 8GB of RAM.

The GPU architecture facilitates several levels of memory having different capacities and latencies. The global memory has the largest capacity (in our case it can store up to 8GB of data). We use it for storing all the inter-arrival times copied from the host environment. Reading/writing the global memory is executed for 128 bytes per transaction, and these operations are cached; there are both L2 and L1 caches available. To improve the memory latency, we have implemented coalesced access where it was possible. We did not find a use for shared memory, and since technically it is the same memory as used for L1 cache, we configured the GPU device to dedicate more memory to L1 cache instead. The fastest type of memory is the register. To utilize them, we have statically allocated arrays in register file using C++ parametrized templates. Our GPU device supports up to 255 registers per thread. Luckily, we have not exceeded this per-thread register usage limitation up to $R = 10$. For $R > 10$, however, register spilling might occur, which would result in the usage of local memory, which has negative performance consequences. Furthermore, to store the parameters of the ER-CHMM structure, we have used the constant memory, which is cached, too. Also, to exploit instruction level parallelism (ILP) we have statically unrolled all the loops for R.

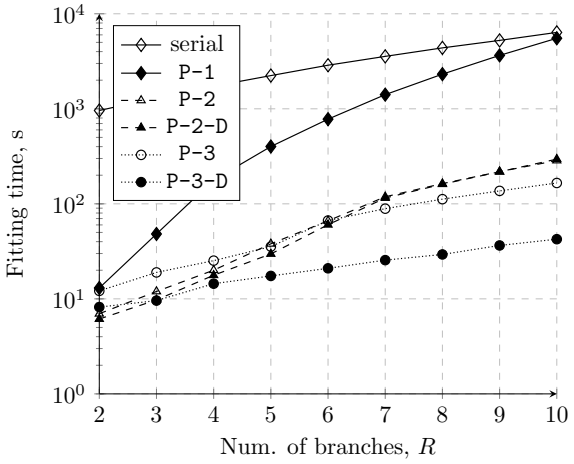

Figure 3: The runtimes of procedures for $L = L^*$

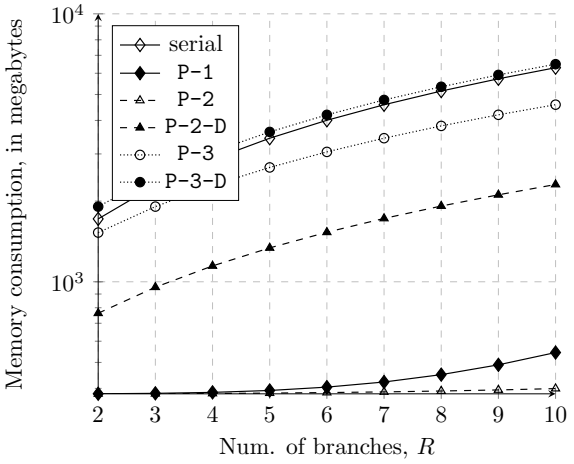

Figure 4: The memory consumption of procedures for $L = L^*$

We do not provide more specific GPU related implementation details here; we just note that we did our best to optimize the algorithms for as fast execution as possible and made the source code available at `https://github.com/minbraz/parmap`.

For testing and demonstration purposes we used a data set of $T = 50,000,000$ inter-arrival time samples, which we obtained by simulating a MAP. The samples were generated by an ER-CHMM with parameters

$$\underline{r} = \{3,3,2,1,1\}, \underline{\lambda} = \{1,3,2,1,3\}, \mathbf{\Pi} = \frac{1}{10}\begin{bmatrix} 1 & 2 & 1 & 4 & 2 \\ 3 & 1 & 2 & 1 & 3 \\ 2 & 3 & 1 & 1 & 3 \\ 4 & 1 & 2 & 2 & 1 \\ 2 & 2 & 1 & 3 & 2 \end{bmatrix}.$$

To point out how high this T parameter is, we note that one of the first papers on the topic ([5]) reported that the EM-based MAP fitting procedure is limited to a few thousands of samples due to the high computation effort.

A critical parameter of the procedures is L, the number of parallel threads used by the implementation. The parallel computing ability is not fully utilized when $L < L^*$, and according to Section 4.2, the memory requirement increases with L, suggesting that $L = L^*$ is an optimal choice. However, when the amount of memory available is not a bottleneck, there might be cases when a higher level of parallelism decreases the computation time. It is due to the pipeline executions of the parallel threads which might benefit from utilizing the computational resources while other threads are blocked. This feature is closely related with the particular HW/SW implementation, and we do not consider it in details, we only conclude that $L = L^*$ is an almost optimal choice and in case of large memory resources, one can try with $L = nL^*$ where n is a small integer number.

In the rest of the section, we provide the memory consumption and the computation time for 100 iterations of all the presented algorithms for $L = L^* = 1920$, $L = 2L^* = 3840$, and $L = 4L^* = 7680$. In all of the numerical experiences, starting from the same initial guess, and using the same samples, the resulted MAP produced by the different versions of the algorithm were the same up to the first 6 digits.

The execution times and the memory requirements are depicted in Figures 3 and 4 for $L = L^*$, in Figures 5 and 6 for $L = 2L^*$ and in Figures 7 and 8 for $L = 4L^*$.

The figures indicate the following conclusions. The speed improvement over the serial algorithm is essential (up to two orders of magnitude), the presented algorithms do benefit from the parallel hardware indeed. In line with our analysis in Section 4.2, method `P-1` slows down the fastest as R increases. Switching to vector-based operations (`P-2` and `P-2-D`) leads to better running times, especially when R is large. In our

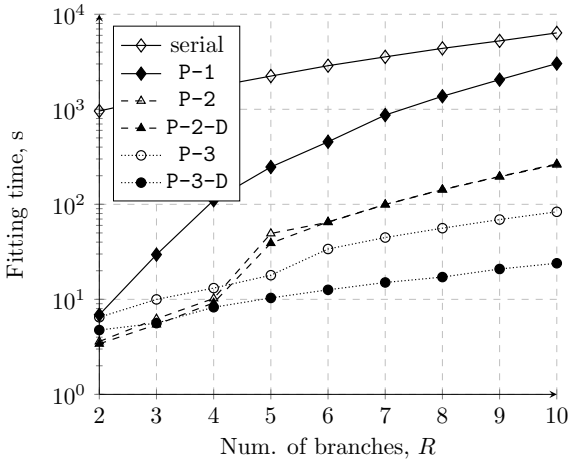

Figure 5: The runtimes of procedures for $L = 2L^*$

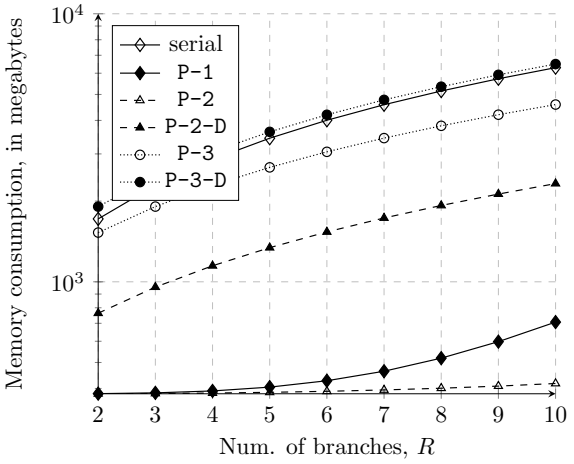

Figure 6: The memory consumption of procedures for $L = 2L^*$

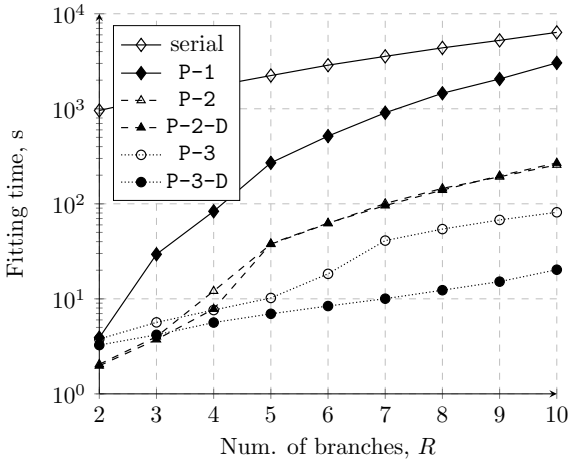

Figure 7: The runtimes of procedures for $L = 4L^*$

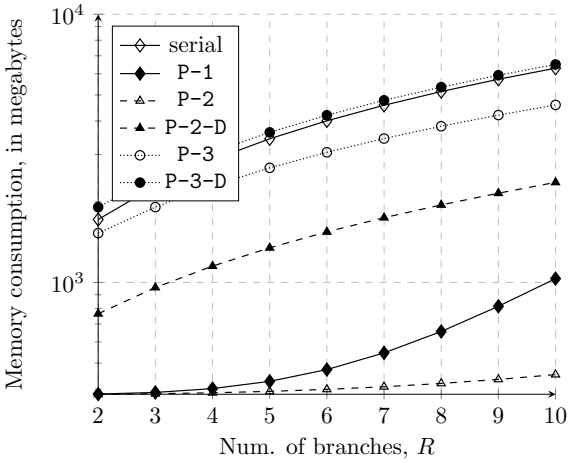

Figure 8: The memory consumption of procedures for $L = 4L^*$

implementation and hardware environment, it turned out that storing the branch densities (method `P-2-D`) does not lead to further improvement. When R is large, the fastest procedures are the 3-pass methods (`P-3` and `P-3-D`). In this case, storing the branch densities leads to significant improvement.

According to Section 4.2, methods `P-2-D` and `P-3-D` need extra memory to store the branch densities, while `P-3` and `P-3-D` need a large amount of memory to store the forward and backward likelihood vectors for every sample of the trace. Our numerical experiments confirm these considerations and demonstrate that `P-1` and `P-2` have the lowest memory demand, in fact, one order of magnitude lower than the serial algorithm.

All parallel procedures did profit from the higher number of partitions (L), especially when R is small. At the same time, the memory consumption for the memory efficient procedures `P-1` and `P-2` increased linearly with L in accordance with Table 1.

In general, algorithms with higher memory consumption are faster, but if T is large, the memory limitation of the hardware can be reached easily. In case the memory size poses a limitation, the more memory efficient procedures need to be applied.

The primary focus of this paper is on the parallel implementation of EM-based MAP fitting, but thanks

to the efficient parallel implementation we have run several fitting experiments for a reasonably large data set, and we have drawn some conclusions on the fitting properties of the EM method. These properties are very much aligned with the related properties of other EM fitting methods (e.g., EM-based phase-type distribution fitting [1]). When the initial guess of the EM method was identical with the ER-CHMM which was used to generate the samples the obtained likelihood value was $-6.4384612 \cdot 10^7$ and the elements of the fitted $\underline{\lambda}$ and $\mathbf{\Pi}$ were identical with the initial ER-CHMM in the first 4 digits. Apart of this artificial experiment, we have also executed fitting experiments with general initial guess for all ER-CHMM structures with 10 states, i.e., where $\underline{r}$ is such that $\sum_{i=1}^{R} r_i = 10$. The initial guess in these experiments was generated by drawing $\mathbf{\Pi}$ randomly and then setting $\lambda_i = \lambda^*$ for $i = 1, \ldots, R$, where λ^* is chosen to fit the sample mean. With this general initial guess the best likelihood is obtained by the $\underline{r} = \{4, 3, 1, 1, 1\}$ ER-CHMM structure and it was $-6.4397824 \cdot 10^7$ the second best likelihood was obtained by the $\underline{r} = \{4, 2, 2, 1, 1\}$ ER-CHMM structure ($-6.4406064 \cdot 10^7$) and the third best one by the $\underline{r} = \{3, 3, 2, 1, 1\}$ structure ($-6.4406496 \cdot 10^7$). In spite of the close likelihood values, the parameters of this best fitting $\underline{r} = \{3, 3, 2, 1, 1\}$ structure were rather different from the ones used for generating the samples:

$$\underline{\lambda}_{fit} = [1.007464, 0.963276, 1.791007, 1.301413, 1.59442],$$

$$\mathbf{\Pi}_{fit} = \begin{bmatrix} 0.032330 & 0.139179 & 0.395032 & 0.203579 & 0.229877 \\ 0.073530 & 0.017319 & 0.192946 & 0.401569 & 0.314634 \\ 0.121469 & 0.245083 & 0.043730 & 0.441435 & 0.148281 \\ 0.001723 & 0.068507 & 0.326094 & 0.381525 & 0.222148 \\ 0.272118 & 0.218650 & 0.108775 & 0.246126 & 0.154328 \end{bmatrix}.$$

This observation refers to an optimization problem with a flat surface and several local minima, which was reported also in previous EM fitting experiments.

6. Conclusions

We have presented three parallel algorithm variants for EM-based MAP fitting and provided a primary performance evaluation of them. We evaluated the algorithms in an implementation independent way in Section 4.2 and collected experimental results of our GPU based implementation in Section 5.

As it is indicated by the performance characterization, all of the proposed algorithm versions might have advantages over the other variants. It is not possible to name an ultimate winner over the algorithm variants because the best choice depends on the model parameters (number of samples T and number of branches R) and the properties of the particular hardware and software environment (e.g., number of parallel threads L, and other specific properties of the HW and SW implementation as the memory access time of different memory units). However, the `P-2-D` and the `P-3-D` methods are often among the best ones when there is no severe memory limitation.

To apply the presented parallel approach in practice, we recommend to implement all the three parallel algorithm versions and find the most appropriate one by experiments.

7. Acknowledgment

This work is supported by the OTKA K-123914 project and by the ÚNKP-17-4-III New National Excellence Program of the Ministry of Human Capacities.

References

[1] Søren Asmussen, Olle Nerman, and Marita Olsson. Fitting phase-type distributions via the EM algorithm. *Scandinavian Journal of Statistics*, 23(4):419–441, 1996.

[2] Levente Bodrog, Armin Heindl, Gábor Horváth, and Miklós Telek. A Markovian canonical form of second-order matrix-exponential processes. *European Journal of Operational Research*, 190(2):459–477, 2008.

[3] Mindaugas Bražėnas, Gábor Horváth, and Miklós Telek. Efficient implementations of the EM-algorithm for transient Markovian arrival processes. In *International Conference on Analytical and Stochastic Modeling Techniques and Applications*, pages 107–122. Springer, 2016.

[4] P. Buchholz, M. Telek, and G. Horvath. A MAP fitting approach with independent approximation of the inter-arrival time distribution and the lag correlation. In *Proceedings. Second International Conference on the Quantitative Evaluation of Systems*, pages 124–133, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[5] Peter Buchholz. An EM-algorithm for MAP fitting from real traffic data. In *Computer Performance Evaluation. Modelling Techniques and Tools: 13th International Conference, TOOLS 2003, Urbana, IL, USA, September 2-5, 2003. Proceedings*, pages 218–236, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[6] Peter Buchholz, Iryna Felko, and Jan Kriege. Transformation of acyclic phase type distributions for correlation fitting. In Alexander Dudin and Koen De Turck, editors, *Analytical and Stochastic Modeling Techniques and Applications: 20th International Conference, ASMTA 2013, Ghent, Belgium, July 8-10, 2013. Proceedings*, pages 96–111, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[7] Peter Buchholz and Jan Kriege. A heuristic approach for fitting MAPs to moments and joint moments. In *QEST 2009, Sixth International Conference on the Quantitative Evaluation of Systems, Budapest, Hungary, 13-16 September 2009*, pages 53–62. IEEE Computer Society, 2009.

[8] Peter Buchholz, Jan Kriege, and Iryna Felko. *Input Modeling with Phase-Type Distributions and Markov Models - Theory and Applications*. SpringerBriefs in Mathematics. Springer, 2014.

[9] Gábor Horváth and Hiroyuki Okamura. A fast EM algorithm for fitting marked Markovian arrival processes with a new special structure. In Maria Simonetta Balsamo, William J. Knottenbelt, and Andrea Marin, editors, *Computer Performance Engineering: 10th European Workshop, EPEW 2013, Venice, Italy, September 16-17, 2013. Proceedings*, pages 119–133, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[10] Jan Kriege and Peter Buchholz. An empirical comparison of MAP fitting algorithms. In Bruno Müller-Clostermann, Klaus Echtle, and Erwin P. Rathgeb, editors, *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance: 15th International GI/ITG Conference, MMB&DFT 2010, Essen, Germany, March 15-17, 2010. Proceedings*, pages 259–273, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[11] Jan Kriege and Peter Buchholz. PH and MAP fitting with aggregated traffic traces. In Kai Fischbach and Udo R. Krieger, editors, *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance: 17th International GI/ITG Conference, MMB & DFT 2014, Bamberg, Germany, March 17-19, 2014. Proceedings*, pages 1–15, Cham, 2014. Springer International Publishing.

[12] L. Li, B. Fu, and C. Faloutsos. Efficient Parallel Learning of Hidden Markov Chain Models on SMPs. *IEICE Transactions on Information and Systems*, 93:1330–1342, 2010.

[13] András Mészáros and Miklós Telek. A two-phase MAP fitting method with APH interarrival time distribution. In *Proceedings of the Winter Simulation Conference*, WSC '12, pages 3939–3950, Berlin, Germany, 2012. Winter Simulation Conference.

[14] Marcel F Neuts. A versatile Markovian point process. *Journal of Applied Probability*, 16(4):764–779, 1979.

[15] J. Nielsen and A. Sand. Algorithms for a parallel implementation of hidden Markov models with a small state space. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pages 452–459, May 2011.

[16] Hiroyuki Okamura and Tadashi Dohi. Faster maximum likelihood estimation algorithms for Markovian arrival processes. In *QEST 2009, Sixth International Conference on the Quantitative Evaluation of Systems, Budapest, Hungary, 13-16 September 2009*, pages 73–82. IEEE Computer Society, 2009.

[17] Hiroyuki Okamura, Tadashi Dohi, and Kishor S. Trivedi. Markovian arrival process parameter estimation with group data. *IEEE/ACM Trans. Netw.*, 17(4):1326–1339, August 2009.

[18] Miklós Telek and Gábor Horváth. A minimal representation of Markov arrival processes and a moments matching method. *Performance Evaluation*, 64(9):1153–1168, 2007.

[19] A. Thümmler, P. Buchholz, and M. Telek. A novel approach for phase-type fitting with the EM algorithm. *IEEE Transactions on Dependable and Secure Computing*, 3(3):245–258, 2006.

[20] William Turin. Unidirectional and parallel Baum-Welch algorithms. *IEEE Trans. Speech and Audio Processing*, 6(6):516–523, 1998.