Queueing Syst (2010) 66: 369–412
DOI 10.1007/s11134-010-9195-9

# Statistical estimation of delays in a multicast tree using accelerated EM

**Furcy Pin · Darryl Veitch · Bruno Kauffmann**

Received: 11 December 2009 / Revised: 8 September 2010 / Published online: 28 October 2010

**Abstract** Tomography is one of the most promising techniques today to provide spatially localized information about internal network performance in a robust and scalable way. The key idea is to measure performance at the edge of the network, and to correlate these measurements to infer the internal network performance.

This paper focuses on a specific delay tomographic problem on a multicast diffusion tree, where end-to-end delays are observed at every leaf of the tree, and mean sojourn times are estimated for every node in the tree. The estimation is performed using the Maximum Likelihood Estimator (MLE) and the Expectation-Maximization (EM) algorithm.

Using queuing theory results, we carefully justify the model we use in the case of rare probing. We then give an explicit EM implementation in the case of i.i.d. exponential delays for a general tree. As we work with non-discretized delays and a full MLE, EM is known to be slow. We hence present a very simple but, in our case, very effective speed-up technique using Principal Component Analysis (PCA). MLE estimations are provided for a few different trees to evaluate our technique.

F. Pin · B. Kauffmann
Département d'Informatique, École Normale Supérieure, 45 rue d'Ulm, 75230 Paris Cedex, France

F. Pin
e-mail: furcy.pin@ens.fr

B. Kauffmann
e-mail: bruno.kauffmann@gmail.com

D. Veitch (✉)
Centre for Ultra-Broadband Information Networks (CUBIN), Department of Electrical and Electronic Engineering, The University of Melbourne, Parkville, Australia
e-mail: dveitch@unimelb.edu.au

B. Kauffmann
INRIA, 23 Avenue d'Italie, 75013 Paris, France

**Keywords** Network tomography · Multicast trees · Expectation-maximization (EM) algorithm · Acceleration · Principal Component Analysis (PCA)

**Mathematics Subject Classification (2000)** 60K25 · 62F30 · 94A99 · 90B22

## 1 Introduction

The need for comprehensive measurement of the Internet has generated considerable interest in certain classes of statistical inverse problems. A particularly interesting and practically relevant measurement paradigm is afforded by end-to-end probing, where test packets called *probes* are sent across the network between participating sources and receivers. In this paradigm the network transports the probes just like any other packet, and does not cooperate with measurement in any way. The analysis of the probe data aims to estimate network characteristics such as link capacities and server loads, to provide path bottleneck localization and characterization, and to remotely measure traffic characteristics.

The network model underlying end-to-end measurement analysis is typically queueing network inspired. For example, in network *delay tomography*, the measured end-to-end transit times or *delays* of probes are used to infer the delays incurred at nodes internal to the network, which are attributed to queueing in network elements. However, with the exception of the special case of a single source and receiver, the statistical models adopted for network elements in the literature are not actually queueing systems, but are postulated a priori.

The goal of this paper is to study a network inference problem with a firm foundation in queueing networks, thereby contributing simultaneously to network tomography, and to the nascent area of inverse problems in queueing. We study a problem in delay tomography over trees: probes are sent from a single source to multiple destinations over a feedforward network of nodes. We consider *multicast trees*, where each node of the tree copies its departing probes over all of its child links. Hence each probe sent from the root node effectively broadcasts over the entire tree until copies arrive at each leaf. Timestamps at the root and leaves can be compared, so that each multicast probe gives rise to a vector of delay values. Multicasting is supported by today's Internet protocols and represents an economical way to reach many receivers, and most works on delay tomography exploit it.

A typical delay model used in tomography over multicast trees is given as follows. To each node there is a random process controlling the delays imparted to packets. The node processes[1] are mutually independent (spatial independence) and are each individually i.i.d. (temporal independence). Thus the end-to-end delay of each probe at a given leaf is the sum of independent random variables, with (in general) different distributions, corresponding to its ancestor nodes in the tree, as shown in the example of Fig. 1. The normal or *cross-traffic* packets in the network are taken to be responsible for the build up of node queues and hence the delays which are experienced by probes, however they do not enter explicitly in the description. Cross traffic is not

[1]Note that usually the processes are associated to links, not nodes, but as these are in 1–1 correspondence this is not essential.

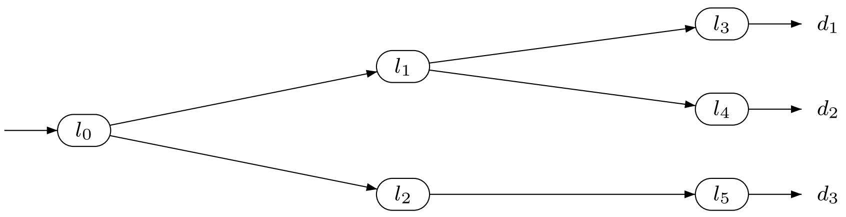


**Fig. 1** Example of a delay tomography problem over a tree: to estimate the means of the six internal random variables $l_0, l_1, l_2, l_3, l_4$ and $l_5$, just by observing samples of the three end-to-end delay variables $d_1, d_2$ and $d_3$, where $d_1 = l_0 + l_1 + l_3$, $d_2 = l_0 + l_1 + l_4$, and $d_3 = l_0 + l_2 + l_5$

assumed to be multicast, indeed the multicast tree is a construct of the probing experiment, whereas cross traffic traverses the full network and simply intersects the tree. Finally, it is assumed that probes are rare enough so as not to significantly perturb the normal traffic over the tree.

In this paper we study the tomography problem as above in the case where the node delay variables are each exponentially distributed. We formulate a Maximum Likelihood Estimation (MLE) problem for their parameters, implemented using the Expectation Maximization (EM) algorithm. Our contributions are as follows. First we show how the tomography model described above corresponds to the delays experienced by probes in an appropriately defined queueing network also carrying cross traffic, thereby justifying the assumptions of a delay tomography problem over a tree in terms of queueing networks for the first time. Secondly, as a delay tomography problem, it is novel in that (see below for details) we do not focus on non-parametric estimation or alternatively with general but discretized delay, but instead work with the full MLE of a continuous density. In particular this involves dealing, both theoretically and practically, with the non-trivial combinatorics inherent in the conditional expectations over a general tree topology. We derive explicit solutions for the combined E and M steps. Finally, we provide a technique for convergence acceleration of the EM algorithm, which is notoriously slow, allowing much larger trees to be considered than would otherwise be the case. The technique has some novel features and may be of broader interest.

Our work is the first to propose a delay tomography model based on exponential delays (see however [13]). Given the accepted queueing origins of network delays, it is surprising that such a canonical choice has escaped attention until now. The chief reason for this omission, as argued for example in [16], is that there is no generally accepted model for the delays in Internet routers, so that flexibility is essential to match reality. While this point is well taken, our view is that realism also requires that node models be consistent with their purported queueing underpinnings, something which has never been shown previously, even in models which introduce, a priori, atomic components in an attempt to reproduce queue-like features [13, 18]. Although the exponential distribution is not considered to be a close fit to packet delays in the Internet today, is it a natural first choice when making a rigorous connection to queueing networks.

To give an example of applications, our techniques could be used by service providers in order to monitor the quality of real-time services. In the case of ADSL 'triple-play boxes' providing IP TV services today, service providers own the end-user equipment, and so can run measurement software as well as operate the backbone and access networks. They therefore have the incentive and the ability to use multicast protocols.

The paper is structured as follows. We begin by discussing prior work in Sect. 2. Section 3 describes the queueing inverse problem and how it maps to the delay tomography problem. Section 4 provides necessary background to the MLE and the EM algorithm, and Sect. 5 shows how these apply in the present case. Section 6 is a technical one showing how expressions for the conditional expectations over the tree which arise can be calculated explicitly. Section 7 exploits these solutions to provide the MLE for a number of example trees, using our EM acceleration technique, which itself is described (and further illustrated) in Sect. 8. We conclude and comment on future work in Sect. 9.

## 2 Related work

We describe three areas of related work: delay tomography, inverse queueing, and EM acceleration.

The first delay tomography paper [7] exploited probe multicasting. It used non-parametric estimators, but only recovered the delay variance at each node rather than the full delay distribution. The related work [16] extended the approach to the entire distribution by discretizing delay, effectively introducing a multinomial model for each node delay, and therefore a large number of parameters. The non-parametric estimators described were based on recursive conditional independence of child subtrees and deconvolution, and have no direct link to the MLE. In [14] a similar multinomial delay model was taken, but a pseudo MLE approach was employed. A full MLE was avoided in order to reduce complexity.

Since multicasting is not always practically feasible, a number of works, including [4, 18, 19] examined unicast alternatives based on a packet-pair scheme where probes are sent in as closely spaced pairs so that they will experience similar delays until a branch point is reached, after which they follow different paths. Here the likelihood is simpler as probes approximately 'multicast' over two paths only, but the packet-pair assumption introduces additional noise and a much higher probing overhead. In [12], hybrid 'flexicast' combinations of unicast and multicast probing are explored in order to tradeoff estimation accuracy against computational and probing costs.

The use of discretized node delay models make tradeoffs between computational cost and accuracy difficult. A small number of papers address this, as we do, by using parametric approaches involving continuous distributions. Using unicast probing, [18] proposes a mixture model for node delays including Gaussian densities and an atom representing the minimum propagation delay. A penalized likelihood was adopted to control the number of Gaussians in the mixture which is maximized using an associated EM algorithm. More recently, using multicast probing [2] also employs a mixture model including a single atom, this time combined with multiple uniform and exponential densities. The analysis is performed in the transform domain with sampled characteristic functions and performs an $L_2$ based optimization using quadratic programming, which scales better than EM to large trees. In [13] a mixture model, consisting of an atom combined with a continuous density satisfying certain conditions, is considered for flexicast probing. Based on examples on simple trees, MLE based approaches are considered but then discarded as intractable in favor of moment based methods using least squares. The study is preliminary but the observations on identifiability important.

In terms of model our work is most closely related to [13] as it treats as a special case the exponential multicast problem as we do, though only over the simplest (two receiver) tree. Otherwise, our work is related to [2] since exponential delays are allowed, and multicasting is used. Our work is less general (and realistic) in insisting on exponential delays alone, but is more realistic in that it is consistent with an actual queueing network model. In terms of solution approach our work is again most closely related to [13] in that an explicit EM solution to the MLE is obtained, but again only for the simplest tree. Over general trees, our work is perhaps closest to [19] in that an EM algorithm is used to find the MLE of a parametric problem involving densities, although the MLE is quite different. Another key difference is that our work does not use distributed methods to evaluate the EM, instead the simplicity of the node model allows closed form solutions for EM to be obtained. We also accelerate the EM in a new way as described below.

The literature on inverse queueing is small, and we know of no work which deals with inference of queue parameters based on observations made across a queueing network which is not a tandem network. In particular we know of no work based on discrete observations of system time over a tree network. The most closely related work is [1], which treats a number of inversion problems, including that posed here in the special case of a tandem queue network. The simpler context allows some more explicit results to be given. On the other hand here we treat arbitrary trees and provide an acceleration method for EM which is also of (great) benefit in the tandem case.

It is well known that the convergence of the EM algorithm can be slow, and there is a considerable literature devoted to speed-up techniques. An element of our technique involves over-relaxation, that is inflating the jump size recommended by EM. This idea is not new, for example it figures in [3, 9, 17], and was explored by Lange and others in the context of EM Gradient Algorithms (see Sect. 4.13, [15]). However, our jump size update rule, which does not bound the allowed increase at any step, is extremely aggressive, and qualitatively different to those we have seen elsewhere, although it shares with [8] the principle that if a candidate step proves too aggressive, in particular if it leads to a decrease in likelihood, then a safer 'fallback' position can be taken (see also 'step decrementing' Sect. 4.14.1, [15]). The other core element of our technique involves using Principal Component Analysis (PCA) to efficiently exploit the information contained in prior evaluations of the likelihood, and to help counter the instability inherent in aggressive updates. This approach was inspired by recent work in robotics [6] in the quite different context of automated path finding. We know of no work which uses similar ideas to accelerate EM or related algorithms.

## 3 A delay tomographic queueing inverse problem

We begin with the model for cross traffic only, and then consider how probes can be introduced.

Consider an open Kelly [11] network of single server FCFS queueing stations connected in a tree topology. Routes corresponding to a given customer class can only move away from the root station (and are not multicast), but are otherwise general, entering the tree at any station and exiting either that same station, or any other further down the tree. The arrival process to route (or class) $r$ is Poisson of intensity $\lambda_r$. All packets have exponential size with mean 1, and the service rate of station $j$ is $\mu_j$. We consider only parameter values consistent with a stationary regime.

We first consider the special case of a tandem network of $K$ stations, as it serves as a building block for what follows. Assume that route $r = 0$ traverses the network from root to leaf, so that for each customer in class 0 we can associate an end-to-end system time, or *delay* $d$. We will call such a route a *path*. Using the well known product form results for such a system in equilibrium, it is straightforward to show that the marginal distribution for the number $N_0^j$ of customers of class 0 in station $j$, $1 \le j \le K$, at a given time instant is

$$\mathbb{P}\big(N_0^j = n_0^j, j = 1, \dots, K\big) = \prod_{j=1}^{K} \left(\frac{\lambda_0}{\gamma_j}\right)^{n_0^j} \left(1 - \frac{\lambda_0}{\gamma_j}\right),$$

where $\gamma_j = \mu_j - \sum_{r \neq 0, j \in r} \lambda_r$ is the residual service capacity of station $j$ available to class 0. By examining the number of customers of class 0 in the system at departure times and conditioning on $d$, it can then be shown [1] that $d$ is the sum of $K$ independent exponential variables, one per station, where the mean parameter for station $j$ is just the reciprocal of the residual service capacity $\gamma_j - \lambda_0$. Furthermore, it is known ([11], Corollary 3.3, p. 62) that the departure processes of the classes exiting the system at station $K$ are Poisson and mutually independent, and that departures from any of these prior to some time $t$ are independent of the system state at time $t$.

Now consider a tree network. The above result for a tandem applies directly to any path, that is the end-to-end delay of each customer of a path is given by the sum of independent exponentials. Note however that this does not imply that the delays seen over different paths are independent. Now the set of stations in any two paths can be partitioned into three tandem subnetworks: a shared portion $S$ from the root down to some last shared station $A$, and two unshared portions $U_1$ and $U_2$ beginning from children of $A$, each terminating at a leaf.

The independence properties given above for the tandem network apply to customers exiting $A$. They imply that the arrival processes to each of $U_1$ and $U_2$ are independent not only of each other, but also of the states of $U_1$ and $U_2$, since the latter are functions only of the prior departures from $S$, which as noted above are independent of the state of $S$ at the departure instant of each probe. Since the service times of the stations in $U_1$ and $U_2$ are also mutually independent, it follows that the delays incurred over $U_1$ and $U_2$ are likewise independent both of each other, and of the delays incurred (by the customers of either path) over $S$. In summary, delays over the tandem subnetworks $S$, $U_1$, and $U_2$ are mutually independent, and inside each of these, delays experienced by customers of a given class (i.e. path) are given by a sum of independent exponentials. This argument extends naturally to the entire tree.

We now introduce multicast probe customers into the system, which behave as follows. The probes arrive as a Poisson process of intensity $\Lambda$ to the root station.

Once a probe has arrived at a station it is treated exactly like a normal customer, but upon exiting, copies are instantaneously made which arrive simultaneously at each of its child stations. Hence each multicast probe traverses all paths (end-to-end routes) but no other routes.

Clearly the system consisting of cross-traffic classes plus the multicast probe class over the tree is not a Kelly network. However, as before, the tandem analysis above applies, showing not only that probe delays over each path are distributed as a sum of independent exponentials, but also that the probe delays on a given path can be analyzed as if the cross traffic were absent, provided the appropriate reduced capacities are used. Furthermore, the above arguments concerning the decoupling of the delays experienced over the shared portion of paths from those below it continue to hold. However, the relationship between delays seen by customers of different paths *within* the shared or the un-shared portions is now substantially different.

To examine this question we revisit our two path example, but now consider the customers on each path to belong to the same multicast probe class.

*Shared part*: there is now only a single probe customer process rather than two. This can be interpreted as perfect station-by-station dependence of the delay components from each path, in contrast to the situation for cross traffic where the service times of customers, for example, were independent.

*Unshared part*: the arrival processes from $A$ to $U_1$ due to path 1, and $A$ to $U_2$ due to path 2, remain Poisson, but are now identical rather than independent, resulting in dependence between the delays of probes (and cross traffic) seen over $U_1$ and $U_2$.

To see why the delays of probes are now dependent on the unshared part, consider the following simple example without cross traffic, where $U_1$ and $U_2$ each consist of a single node of capacity $\mu$. In other words, $U_1$ and $U_2$ are M/M/1 queues with independent service times, fed by the same Poisson Process of intensity $\Lambda$. Each queue has a marginal probability $(1-\rho) := (1-\frac{\Lambda}{\mu})$ of being empty. Now $U_1$ (resp. $U_2$) is empty at the arrival time $t_N$ of the $N$th probe packet if and only if the previous probe had a delay $D_1$ (resp. $D_2$) which is less than the inter-arrival time $t_N - t_{N-1}$. Assume in contradiction that spatial independence holds between $D_1$ and $D_2$; this leads to:

$$\begin{aligned}\mathbb{P}[\text{both queues empty}] &= \int_0^\infty \mathbb{P}[D_1 \le \tau, D_2 \le \tau]\mathbb{P}[t_N - t_{N-1} = \tau]\,d\tau \\ &= 1 - 2\rho + \frac{\rho^2}{2\rho - \rho^2},\end{aligned}$$

which is not equal to $(1-\rho)^2$ (unless $\rho = 1$), the result one would obtain if the waiting times were independent. But this is a contradiction, because the assumptions of independence between $D_1$ and $D_2$, and on the service times, clearly implies independence of waiting times. It follows that the delays must in fact be dependent.

Although multicast probes break the strict spatial independence property of path delays, we expect this dependence to be weak in most cases, since the arrival processes at $U_1$ and $U_2$ remain independent of the states of $U_1$ and $U_2$ (at arrival instants), the service times in $U_1$ and $U_2$ remain independent, and furthermore the cross-traffic arrivals (from paths or other routes) are independent as before. In particular, if we assume that $\Lambda$ is small, so that with high probability there is no more than a single probe in any given station, then the states of $U_1$ and $U_2$ are only slightly perturbed by probes and are thus approximately independent, and so the delays over $U_1$ are $U_2$ are likewise close to independent.

It is a general principle of network probing that $\Lambda$ be kept small, in order to avoid consuming network bandwidth, perturbing the system to be measured, and to prevent probes being confused with network attacks. Since $\Lambda$ is under the control of the prober, it is quite reasonable to assume it is small. This same *rare probing* assumption justifies the assumption of temporal independence in the time series of probe delays associated to each path, used in the MLE formulation below.

In conclusion, the delays of rare multicast probes sent over a Kelly tree network of cross traffic closely hew to the assumptions of a spatially and temporally independent delay tomography problem over a tree with exponential delays. Namely, per-station delays experienced by probes obey a simple structure: perfect dependence over stations on the shared part of the path, and independence between the unshared parts. Cross traffic appears only through the values of the residual capacity parameters $\{\gamma_j - \Lambda\}$ to be estimated. Since $\Lambda$ is known, the residual capacities $\{\gamma_j\}$ relating to cross traffic only can subsequently be recovered. The actual intensities $\{\lambda_r\}$ and the server rates $\{\mu_j\}$ are not identifiable, however they can be recovered in principle by other means, for example using a prior measurement phase with fixed packet sizes, as discussed in the tandem case in [1].[2]

## 4 The EM algorithm

In this section we provide the necessary background to maximum likelihood estimation.

### 4.1 The maximum likelihood estimator

We observe a set $X = (x(1), \dots, x(N))$ of random vectors which are assumed i.i.d. with a multivariate probability density function (p.d.f.) $p_\alpha$, where $\alpha = (\alpha_1, \dots, \alpha_K)$ is some unknown parameter vector we want to estimate.

For a given set of observed data $X = (x(1), \dots, x(N))$, the function $L_X(\theta) = \prod_{i=1}^N p_\theta(x(i))$ of the parameter vector $\theta$ is called the *likelihood function*. The *maximum likelihood estimator* (MLE) is then defined to be

$$\hat{\alpha} = \underset{\theta}{\operatorname{argmax}}\, L_X(\theta) = \underset{\theta}{\operatorname{argmax}} \log L_X(\theta) \tag{1}$$

and can be found by solving the *likelihood equation*:

$$\frac{\partial L_X(\theta)}{\partial \theta} = 0, \quad \text{or equivalently} \quad \frac{\partial \log L_X(\theta)}{\partial \theta} = 0, \tag{2}$$

$\log L(\theta)$ being called the *log-likelihood* of $\theta$.

[2] An interesting discussion, in the context of a priori node models, of how the addition of atoms can assist in identifiability is given in [18].

The MLE is a very popular estimator, which has been shown to be biased but consistent and asymptotically efficient when $N$ tends towards infinity, and is known to exist under suitable regularity conditions [5]. In some cases where the data is incomplete it can be quite hard to calculate, and for this reason the EM algorithm was introduced.

## 4.2 The EM algorithm

Sometimes the observed data is not the complete data itself, but only some subset, or a manifestation of this data. That is, instead of observing a set of data $X = (x(1), \dots, x(N))$, we just observe their images under some measurable function $f$, i.e. we observe $Y = (y(1), \dots, y(N))$ with $y(i) = f(x(i))$ for $i \in \{1, \dots, N\}$, which we write with a slight abuse of notation $Y = f(X)$. The likelihood function for this data is then: $L_Y(\theta) = \prod_{i=1}^{N} q_\theta(y(i))$, where $q_\theta$ is the p.d.f. of the random variable $f(x)$ with $x$ having the p.d.f. $p_\theta$, and can be expressed as $q_\theta(y) = \int_{f^{-1}(y)} p_\theta(x)\,dx$.

This likelihood can be tricky to compute, and it can be even trickier to solve the associated likelihood equation. In such a case, an alternative is to use the EM algorithm, which is based on the following iterative formula:

$$\hat{\alpha}^{(k+1)} = \operatorname*{argmax}_{\theta} Q\bigl(\theta, \hat{\alpha}^{(k)}\bigr) \tag{3}$$

where $Q(\theta, \hat{\alpha}^{(k)}) := \mathbb{E}_{\hat{\alpha}^{(k)}}(\log L_X(\theta) | Y)$ is the conditional expectation of the log-likelihood of the parameter vector $\theta$ with respect to the unknown data $X$, under the assumption that $x$ follows the p.d.f. $p_{\hat{\alpha}^{(k)}}$ and knowing that $f(X) = Y$.

Using the independence of observations and the linearity of expectation, this can be rewritten:

$$\hat{\alpha}^{(k+1)} = \operatorname*{argmax}_{\theta} \sum_{i=1}^{N} \mathbb{E}_{\hat{\alpha}^{(k)}}\bigl(\log p_\theta(x) | f(x) = y(i)\bigr). \tag{4}$$

Usually, this iteration is computed in two steps, the E-step where the conditional expectation is computed, and the M-step where it is maximized over $\theta$. Starting from an initial value $\hat{\alpha}^{(0)}$, the EM algorithm is then the following algorithm:

**Algorithm 1** EM algorithm

```
1: procedure EM( $\hat{\alpha}^{(0)}$, $Y$)
2:     $k \leftarrow 0$;
3:     loop
4:         compute $Q(\theta, \hat{\alpha}^{(k)}) := \mathbb{E}_{\hat{\alpha}^{(k)}}(\log L_X(\theta) | f(X) = Y)$ for all $\theta$;   ▷ E-step
5:         compute $\hat{\alpha}^{(k+1)} := \operatorname{argmax}_\theta Q(\theta, \hat{\alpha}^{(k)})$;   ▷ M-step
6:         if $\hat{\alpha}^{(k+1)} = \hat{\alpha}^{(k)}$ then return $\hat{\alpha}^{(k)}$; else $k \leftarrow k + 1$;
7:     end loop
8: end procedure
```

In other words, we compute the recursive sequence $(\hat{\alpha}^{(k)})_{k\in\mathbb{N}}$, defined by $\hat{\alpha}^{(0)}$ and (4) for all $k$ in $\mathbb{N}$. The starting point $\hat{\alpha}^{(0)}$ is an arbitrary point in the parameter space, or may be chosen using some simple 'first guess' estimator. Of course, in practice the stopping criterion "$\hat{\alpha}^{(k+1)} = \hat{\alpha}^{(k)}$" will typically never be reached and is replaced by a more practical one.

The EM algorithm has the following interesting properties:

**Property 4.1** *For all* $\hat{\alpha}^{((k))}$, $L_Y(\hat{\alpha}^{(k+1)}) \geq L_Y(\hat{\alpha}^{(k)})$, *which is a direct consequence of the following lemma*:

**Lemma 4.1** *For all* $\theta_1$ *and* $\theta_2$, $\log L_Y(\theta_2) - \log L_Y(\theta_1) \geq Q(\theta_2, \theta_1) - Q(\theta_1, \theta_1)$.

**Property 4.2** *Any fixed point of the algorithm in the interior of* $\Theta$ *is a solution of the likelihood equation.*

Property 4.1 states that the likelihood of $\hat{\alpha}^{(k)}$ increases with each iteration of the algorithm. When the likelihood has an upper bound, which is almost always the case in practice, the property also shows that the sequence $(L_Y(\hat{\alpha}^{(k)}))_{k\in\mathbb{N}}$ converges. Property 4.2 shows that if the sequence $(\hat{\alpha}^{(k)})$ converges, it converges towards a solution of the likelihood equation.

Unfortunately, even if the sequence $(L_Y(\hat{\alpha}^{(k)}))$ converges, it might be possible in some cases that the sequence $(\hat{\alpha}^{(k)})$ does not converge. We refer to Wu [20] and McLachlan and Krishnan [15] for sufficient conditions of convergence of $(\hat{\alpha}^{(k)})$, proofs of the properties and more details about EM.

## 5 EM for exponential tomography

In this section we apply the EM algorithm to our delay tomographic problem.

Consider a tree $\mathcal{T}$, and call $T$ the set of its nodes and $V \subset T$ the set of its leaves. We introduce the fixed parameter vector $\alpha = (\alpha_j)_{j\in T}$ and the variable parameter vectors $\hat{\alpha}^{(k)} = (\hat{\alpha}_j^{(k)})_{j\in T}$. The complete data random vector $x$ of the previous section will correspond to the vector $l \in \mathbb{R}^T$ of the delays of each node, which are supposed independent and exponentially distributed with expected value $\alpha$, and the observed data vector $y$ will correspond to $d \in \mathbb{R}^V$, the vector of all end-to-end delays from the root to each leaf. We will have $d = f(l)$ for some linear function $f$ depending on the topology of the tree. We recall that the probability density function of an exponentially distributed variable of mean value $\alpha_j$ is $p_{\alpha_j}(z) = \frac{1}{\alpha_j} e^{-z/\alpha_j}$.

The fixed vector $\alpha$ will be referred as the *ground truth*, and the variables vectors $\hat{\alpha}^{(k)}$ as the current estimates (of EM). We wish to estimate $\alpha$ via $\hat{\alpha}^{(k)}$, hoping that this last sequence will converge 'close to' $\alpha$. In networks context, $\alpha_j$ corresponds to the mean sojourn time of probes in server $j$.

The results given in this section actually hold more generally for any set $T$ and $V$ with any random exponential vector $l \in \mathbb{R}^T$ (with independent coordinates) and any linear function $f : \mathbb{R}^T \to \mathbb{R}^V$. In particular they hold for delay tomography problems where the network topology is not tree-like.

## 5.1 Specialization of the iterative formula

Usually, each iteration of EM can be computed in two steps: the E-step, where we compute the conditional expectation of the log-likelihood, and the M-step where we maximize it. But when the hidden data belongs to the regular exponential family, as is the case here, it is well known [15] that the E- and M-steps can be solved directly in one step. In other words, the iteration can be made more explicit. Indeed, we have from (4):

$$\begin{aligned}\hat{\alpha}^{(k+1)} &:= \operatorname*{argmax}_{\theta} \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\hat{\alpha}^{(k)}}\big(\log p_\theta(l) | f(l) = d(i)\big)\\ &= \operatorname*{argmax}_{\theta} \sum_{i=1}^{N} \frac{\int_{\{l | f(l)=d(i)\}} \log(p_\theta(l)) p_{\hat{\alpha}^{(k)}}(l)\, dl}{q_{\hat{\alpha}^{(k)}}(d)},\end{aligned} \tag{5}$$

where in our case $p_\theta(l) = \prod_{j\in T} \frac{1}{\theta_j} e^{-\frac{l_j}{\theta_j}}$, and $\log p_\theta(l) = \sum_{j\in T}(\log \frac{1}{\theta_j} - \frac{l_j}{\theta_j})$ for every $\theta$.

We notice that $\log p_\theta(l)$ is easily differentiable according to $\theta$, giving:

$$\frac{\partial \log p_\theta(l)}{\partial \theta_j} = -\frac{1}{\theta_j} + \frac{l_j}{\theta_j^2} = -\frac{1}{\theta_j^2}(\theta_j - l_j),$$

and therefore $\mathbb{E}_{\hat{\alpha}^{(k)}}(\log p_\theta(l) | f(l) = d(i))$ is also differentiable, with:

$$\begin{aligned}\frac{\partial \mathbb{E}_{\hat{\alpha}^{(k)}}(\log p_\theta(l) | f(l) = d(i))}{\partial \theta_j} &= -\frac{1}{\theta_j^2} \frac{\int_{\{l | f(l)=d(i)\}} (\theta_j - l_j) p_{\hat{\alpha}^{(k)}}(l)\, dl}{q_{\hat{\alpha}^{(k)}}(d(i))}\\ &= -\frac{1}{\theta_j^2} \mathbb{E}_{\hat{\alpha}^{(k)}}\big(\theta_j - l_j | d(i)\big)\\ &= -\frac{1}{\theta_j^2}\big(\theta_j - \mathbb{E}_{\hat{\alpha}^{(k)}}\big(l_j | d(i)\big)\big).\end{aligned}$$

We then have:

$$\frac{\partial \sum_{i=1}^{N} \mathbb{E}_{\hat{\alpha}^{(k)}}(\log p_\theta(l) | f(l) = d(i))}{N \partial \theta_j} = -\frac{1}{\theta_j^2}\left(\theta_j - \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\hat{\alpha}^{(k)}}\big(l_j | d(i)\big)\right).$$

Thus, setting this derivative to zero leads to $\theta = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\hat{\alpha}^{(k)}}(l | d(i))$, and so

$$\hat{\alpha}^{(k+1)} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\hat{\alpha}^{(k)}}\big(l | d(i)\big). \tag{6}$$

*Remark 5.1* Here we have generalized the conditional expectation to the multivariate case. That is, in $\mathbb{E}_{\hat{\alpha}^{(k)}}(l | d(i))$, $l$ is a vector, where we have defined

$\mathbb{E}_{\hat{\alpha}^{(k)}}(l|d(i)) := (\mathbb{E}_{\hat{\alpha}^{(k)}}(l_j|d(i)))_{j\in T}$, and the sum $\sum_{i=1}^{N}\mathbb{E}_{\hat{\alpha}^{(k)}}(l|d(i))$ is to be understood as a component-wise addition.

We just reduced the E- and M-steps to one, a significant simplification which in many contexts would almost constitute a 'solution' to the problem. However, computing the conditional expectation $\mathbb{E}_{\hat{\alpha}^{(k)}}(l|d)$ remains a challenge as it involves dealing with combinatorics over the tree, and is in fact a main part of our work. In the next section, we explain how it can be computed efficiently. First, we point out an interesting property which will be useful later.

**Property 5.1** *For any linear function $f$, we have for all $k$:*

$$f(\hat{\alpha}^{(k+1)}) = \bar{d} = \frac{1}{N}\sum_{i=1}^{N} d(i) \tag{7}$$

*where again $\bar{d}$ is a vector defined by averaging component-wise.*

*Proof* Thanks to the linearity of the conditional expectation and the linearity of $f$, we have in our case that $\mathbb{E}_{\hat{\alpha}^{(k)}}(f(l)|d) = f(\mathbb{E}_{\hat{\alpha}^{(k)}}(l|d))$. Therefore,

$$f\big(\hat{\alpha}^{(k+1)}\big) = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{\hat{\alpha}^{(k)}}\big(f(l)|d(i)\big) = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{\hat{\alpha}^{(k)}}\big(d|d(i)\big) = \frac{1}{N}\sum_{i=1}^{N} d(i). \qquad \square$$

Because of this relation, we know that each term of the EM sequence $(\hat{\alpha}^{(k)})$ except the first will satisfies $f(\hat{\alpha}^{(k)}) = \bar{d}$. Therefore, the sequence stays in $f^{-1}(\bar{d})$ which, since $f$ is linear, is a linear subspace of $\mathbb{R}^T$.

## 6 Explicit formula for $\mathbb{E}(l|d)$

In this section we compute the conditional expectation $\mathbb{E}_{\hat{\alpha}^{(k)}}(l|d)$, which is the key to the evaluation of the step function (6). Since

$$\mathbb{E}_{\hat{\alpha}^{(k)}}(l|d) = \frac{\int_{f^{-1}(d)} l p_{\hat{\alpha}^{(k)}}(l)\,dl}{\int_{f^{-1}(d)} p_{\hat{\alpha}^{(k)}}(l)\,dl} := \frac{\xi_{\hat{\alpha}^{(k)}}(l|d)}{q_{\hat{\alpha}^{(k)}}(d)}, \tag{8}$$

the calculation can be divided in the computation of the two terms $q_{\hat{\alpha}^{(k)}}(d)$ and $\xi_{\hat{\alpha}^{(k)}}(l|d)$.

By their nature these calculations are detailed. This section is self-contained and could be skipped on a first reading.

### 6.1 Notations

In this section we are interested in a single iteration of the EM algorithm. In order to simplify notations, we will here (and only here) write $\alpha$ instead of $\hat{\alpha}^{(k)}$. Similarly, in order to have another point of view and nicer notations, we will also introduce the *rate* $\gamma_j := \frac{1}{\alpha_j}$ and use the notation $p_{\alpha_j}(z) = \gamma_j e^{-g_j z}$ instead.

We recall that $\mathcal{T}$, $T$, $V \subset T$, $l = (l_j)_{j\in T} \in \mathbb{R}^T$ and $d = (d_j)_{j\in V} \in \mathbb{R}^V$ denote respectively the tree we consider, the set of its node, the set of its leaves, the vector of delays on each nodes and the vector of end-to-end delays in the tree. The variable vectors $\alpha = (\alpha_j)_{j\in T}$ and $\gamma = (\gamma_j)_{j\in T}$ is the current estimate of EM.

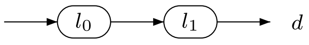
As in Sect. 5, the observed data are the end-to-end delay vectors $d(1), \dots, d(N)$, and are the images under some linear function $f_{\mathcal{T}}$ of the unknown complete data $l(1), \dots, l(N)$, where $f_{\mathcal{T}}$ captures the details of the tree topology.

We provide $T$ with the order $\prec$ defined by: for all $i, j$ in $T$, $i \prec j$ if $i$ is an ancestor of $j$. With these notations, the function $f_{\mathcal{T}} : \mathbb{R}^T \to \mathbb{R}^V$ such that $f_{\mathcal{T}}(l) = d$ is given by $\forall k \in V$, $(f_{\mathcal{T}}(l))_k = d_k = \sum_{\substack{j\in T\\ j\preceq k}} l_j$, and the two terms of the fraction (8) can be written:

$$q_\alpha(\mathcal{T}, d) = \int_{f_{\mathcal{T}}^{-1}(d)} \prod_{j\in T} \gamma_j e^{-\gamma_j l_j}\, dl \quad \text{and} \quad \xi_\alpha(\mathcal{T}, l|d) = \int_{f_{\mathcal{T}}^{-1}(d)} l \prod_{j\in T} \gamma_j e^{-\gamma_j l_j}\, dl. \tag{9}$$

## 6.2 Some simple examples

*(a) 2 nodes tree*



In this simple case, since $l_0$ and $l_1$ are linked to $d$ by $l_0 + l_1 = d$, there is only one unknown. Therefore $q_\alpha$ can be expressed as an integral over $l_0$ only.

$$q_\alpha(\mathcal{T}, d) = \gamma_0\gamma_1 \int_{l_0=0}^{d} e^{-\gamma_0 l_0} e^{-\gamma_1(d-l_0)}\, dl_0 = \gamma_0\gamma_1 \left( \frac{e^{-\gamma_0 d}}{\gamma_1 - \gamma_0} + \frac{e^{-\gamma_1 d}}{\gamma_0 - \gamma_1} \right),$$

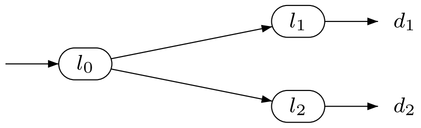
and similarly:

$$\xi_\alpha(\mathcal{T}, l_0|d) = \gamma_0\gamma_1 \left( \frac{e^{-\gamma_0 d}}{\gamma_1 - \gamma_0} \left( d - \frac{1}{\gamma_1 - \gamma_0} \right) + \frac{e^{-\gamma_1 d}}{(\gamma_0 - \gamma_1)^2} \right).$$

Although the figure does not suggest it, the problem is actually symmetric in the nodes 0 and 1. Indeed, what we observe being the sum of two delays, the tree $0 \to 1$ is equivalent to the tree $1 \to 0$. Therefore, we have by symmetry:

$$\xi_\alpha(\mathcal{T}, l_1|d) = \gamma_0\gamma_1 \left( \frac{e^{-\gamma_0 d}}{(\gamma_1 - \gamma_0)^2} + \frac{e^{-\gamma_1 d}}{\gamma_0 - \gamma_1} \left( d - \frac{1}{\gamma_0 - \gamma_1} \right) \right).$$

*(b) Root with 2 leaves*



In this case, since $l_0 + l_1 = d_1$ and $l_0 + l_2 = d_2$, we can consider as before only one unknown $l_0$, and express $q_\alpha$ as an integral over $l_0$ between 0 and $d_0 := \min\{d_1, d_2\}$. Since $l_0, l_1$ and $l_2$ are nonnegative, $l_0$ has to be smaller than $d_1$ and $d_2$. We have:

$$\begin{aligned} q_\alpha(\mathcal{T}, d) &= \gamma_0\gamma_1\gamma_2 \int_{l_0=0}^{d_0} e^{-\gamma_0 l_0} e^{-\gamma_1(d_1-l_0)} e^{-\gamma_2(d_2-l_0)}\, dl_0 \\ &= \gamma_0\gamma_1\gamma_2 \left( e^{-\gamma_0 d_0} \frac{e^{-\gamma_1(d_1-d_0)} e^{-\gamma_2(d_2-d_0)}}{\gamma_1+\gamma_2-\gamma_0} + \frac{e^{-\gamma_1 d_1} e^{-\gamma_2 d_2}}{\gamma_0-\gamma_1-\gamma_2} \right), \end{aligned}$$

and similarly:

$$\begin{aligned} \xi_\alpha(\mathcal{T}, l_0|d) &= \gamma_0\gamma_1\gamma_2 \left( e^{-\gamma_0 d_0} \frac{e^{-\gamma_1(d_1-d_0)} e^{-\gamma_2(d_2-d_0)}}{\gamma_1+\gamma_2-\gamma_0} \left( d_0 - \frac{1}{\gamma_1+\gamma_2-\gamma_0} \right) \right. \\ &\quad \left. + \frac{e^{-\gamma_1 d_1} e^{-\gamma_2 d_2}}{(\gamma_0-\gamma_1-\gamma_2)^2} \right), \\ \xi_\alpha(\mathcal{T}, l_1|d) &= \gamma_0\gamma_1\gamma_2 \left( e^{-\gamma_0 d_0} \frac{e^{-\gamma_1(d_1-d_0)} e^{-\gamma_2(d_2-d_0)}}{(\gamma_1+\gamma_2-\gamma_0)} \left( d_1 - d_0 - \frac{1}{\gamma_0-\gamma_1-\gamma_2} \right) \right. \\ &\quad \left. + \frac{e^{-\gamma_1 d_1} e^{-\gamma_2 d_2}}{\gamma_0-\gamma_1-\gamma_2} \left( d_1 - \frac{1}{\gamma_0-\gamma_1-\gamma_2} \right) \right), \end{aligned}$$

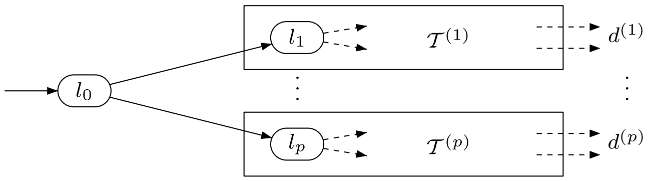
and $\xi_\alpha(\mathcal{T}, l_2|d)$ can be deduced from $\xi_\alpha(\mathcal{T}, l_1|d)$ by symmetry between nodes 1 and 2.

## 6.3 Inductive expression

The last example above can readily be extended to more than two leaves. More generally, it suggests that it be possible to express $q_\alpha$ (resp. $\xi_\alpha$) for any tree as an integral over the delay in the root node from 0 to the minimum of the end-to-end delays, of some term using $q_\alpha$ (resp. $\xi_\alpha$ and $q_\alpha$) inductively applied to the child subtrees of the root. We now show how this can be done.

Let 0 denote the root of the tree, and $p$ the number of its children. In the case where the tree is a single node, i.e. $p = 0$, we have obviously $q_\alpha(d) = \gamma_0 e^{-\gamma_0 d_0}$. When $p \geq 1$ we denote by $\mathcal{T}^{(1)}, \mathcal{T}^{(2)}, \ldots, \mathcal{T}^{(p)}$ the associated child subtrees. Subtree $\mathcal{T}^{(i)}$ has nodes $T^{(i)}$ and leaves $V^{(i)} \subset T^{(i)}$.

We notice that $(V^{(1)}, V^{(2)}, \ldots, V^{(p)})$ forms a partition of $V$, and therefore any vector $d$ in $\mathbb{R}^V$ can be identified with a vector $d = (d^{(1)}, d^{(2)}, \ldots, d^{(p)})$ in $\mathbb{R}^{V^{(1)}} \times \mathbb{R}^{V^{(2)}} \times \cdots \times \mathbb{R}^{V^{(p)}}$. Similarly, any vector $l$ in $\mathbb{R}^T$ can be identified with a vector $l = (l_0, l^{(1)}, \ldots, l^{(p)})$ in $\mathbb{R} \times \mathbb{R}^{T^{(1)}} \times \cdots \times \mathbb{R}^{T^{(p)}}$.



**Theorem 6.1** *Define* $d_0 := \min\{d_j \mid j \in V\}$. *The following inductive relation holds*:

$$q_\alpha(\mathcal{T}, d) = \int_{l_0=0}^{d_0} \gamma_0 e^{-\gamma_0 l_0} \left[ \prod_{i=1}^{p} q_\alpha\big(\mathcal{T}^{(i)}, d^{(i)} - (l_0)\big) \right] dl_0, \tag{10}$$

*where the slight abuse of notation* $d^{(i)} - (l_0)$ *denotes the vector* $(d_j^{(i)} - l_0)_{j \in V^{(i)}} \in \mathbb{R}^{V^{(i)}}$.

*Proof* For a more convenient notation, we introduce for each $i \in \{1, \dots, p\}$ the function $f^{(i)} := f_{\mathcal{T}^{(i)}}$ which is to the tree $\mathcal{T}^{(i)}$ what the function $f_{\mathcal{T}}$ is to the tree $\mathcal{T}$.

We notice that the following relation holds: for all $l$ in $\mathbb{R}^T$, let $d = f(l)$; then for all $k \in V$, there exist one unique $i \in \{1, \dots, p\}$ such that $k \in V^{(i)}$, and:

$$d_k^{(i)} = d_k = \sum_{\substack{j \in T \\ j \preceq k}} l_j = l_0 + \sum_{\substack{j \in T^{(i)} \\ j \preceq k}} l_j^{(i)} = l_0 + \big(f^{(i)}\big(l^{(i)}\big)\big)_k.$$

This gives, for all $i \in \{1, \dots, p\}$, $d^{(i)} = (l_0) + f^{(i)}(l^{(i)})$, and therefore:

$$l^{(i)} \in \big(f^{(i)}\big)^{-1}\big(d^{(i)} - (l_0)\big).$$

Therefore, the integral in (9) over $l \in f_{\mathcal{T}}^{(-1)}(d)$ can be sliced as an external integral over $l_0 \in [0, d_0]$ where $d_0 = \min\{d_k \mid k \in V\}$, and a product of internal integrals over $l^{(i)} \in (f^{(i)})^{-1}(d^{(i)} - (l_0))$ for each $i$, which gives

$$q_\alpha(\mathcal{T}, d) = \int_{l_0=0}^{d_0} \gamma_0 e^{-\gamma_0 l_0} \left[ \prod_{i=1}^{p} \int_{(f^{(i)})^{-1}(d^{(i)} - (l_0))} \prod_{j \in T_i} \gamma_j e^{-\gamma_j (l_j^{(i)})} \, dl^{(i)} \right] dl_0.$$

As we can see, the inner integral looks very similar to the initial integral in (9), and indeed, we can finally write:

$$q_\alpha(\mathcal{T}, d) = \int_{l_0=0}^{d_0} \gamma_0 e^{-\gamma_0 l_0} \left[ \prod_{i=1}^{p} q_\alpha\big(\mathcal{T}^{(i)}, d^{(i)} - (l_0)\big) \right] dl_0. \qquad \square$$

The function $\xi_\alpha(\mathcal{T}, l|d)$ also satisfies an inductive relation, but its expression is slightly more complicated. From (9), similar reasoning shows that the following inductive formula holds. For the root:

$$\xi_\alpha(\mathcal{T}, l_0|d) = \int_{l_0=0}^{d_0} \gamma_0 l_0 e^{-\gamma_0 l_0} \left[ \prod_{i=1}^{p} q_\alpha\big(\mathcal{T}^{(i)}, d^{(i)} - (l_0)\big) \right] dl_0, \tag{11}$$

and for any node $j \in T \setminus \{0\}$, let $i \in \{1, \dots, p\}$ be the unique child of the root such that $j \in T^{(i)}$, we have:

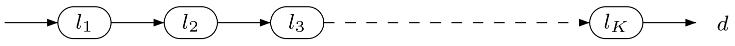
$$\xi_\alpha(\mathcal{T}, l_j|d) = \int_{l_0=0}^{d_0} \gamma_0 e^{-\gamma_0 l_0} \left[ \xi_\alpha\big(\mathcal{T}^{(i)}, l_j^{(i)} | d^{(i)} - l_0\big) \prod_{k \neq i} q_\alpha\big(\mathcal{T}^{(k)}, d^{(k)} - (l_0)\big) \right] dl_0. \tag{12}$$

Using this inductive formula, it is possible to deduce a recursive algorithm computing the expanded symbolic expression for the terms $q_\alpha$ and $\xi_\alpha$. However, we prefer to derive an alternative expression which, as we will see presently, is simpler.

## 6.4 More examples

The following examples can be derived using the inductive expressions above. The first generalizes the case of a unary tree to any number of nodes, and the second is a simple tree for which the expanded expressions of $q_\alpha$ and $\xi_\alpha$ are already quite complicated. Here and below we recommend that the reader first focus on the expressions for $q_\alpha$.

*(c) Unary tree with K nodes*

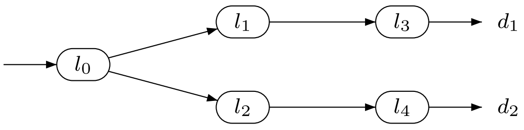


We have:

$$q_\alpha(\mathcal{T},d) = \left(\prod_{j=1}^{K}\gamma_j\right)\sum_{j=1}^{K}\frac{e^{-\gamma_j d}}{\prod_{k\neq j}(\gamma_k-\gamma_j)}, \quad \text{and}$$

$$\xi_\alpha(\mathcal{T},l_i|d) = \left(\prod_{j=1}^{K}\gamma_j\right)\left(\frac{e^{-\gamma_i d}}{\prod_{k\neq i}(\gamma_k-\gamma_i)}\left(d-\sum_{k\neq i}\frac{1}{\gamma_k-\gamma_i}\right)\right.$$
$$\left.+\sum_{j\neq i}\frac{e^{-\gamma_j d}}{(\gamma_i-\gamma_j)^2\prod_{\substack{k\neq i\\ k\neq j}}(\gamma_k-\gamma_j)}\right).$$

*(d) One root with two 2-server leaves*



As we did in Example (b), we introduce $d_0 := \min\{d_1,d_2\}$. We have:

$$q_\alpha(\mathcal{T},d)$$
$$=\gamma_0\gamma_1\gamma_2\gamma_3\gamma_4\left(e^{-\gamma_0 d_0}\left(\frac{e^{-\gamma_1(d_1-d_0)}e^{-\gamma_2(d_2-d_0)}}{(\gamma_1+\gamma_2-\gamma_0)(\gamma_3-\gamma_1)(\gamma_4-\gamma_2)}\right.\right.$$
$$+\frac{e^{-\gamma_1(d_1-d_0)}e^{-\gamma_4(d_2-d_0)}}{(\gamma_1+\gamma_4-\gamma_0)(\gamma_3-\gamma_1)(\gamma_2-\gamma_4)}$$
$$\left.+\frac{e^{-\gamma_3(d_1-d_0)}e^{-\gamma_2(d_2-d_0)}}{(\gamma_3+\gamma_2-\gamma_0)(\gamma_1-\gamma_3)(\gamma_4-\gamma_2)}+\frac{e^{-\gamma_3(d_1-d_0)}e^{-\gamma_4(d_2-d_0)}}{(\gamma_3+\gamma_4-\gamma_0)(\gamma_1-\gamma_3)(\gamma_2-\gamma_4)}\right)$$

$$+\frac{e^{-\gamma_1 d_1}e^{-\gamma_2 d_2}}{(\gamma_0-\gamma_1-\gamma_2)(\gamma_3-\gamma_1)(\gamma_4-\gamma_2)}+\frac{e^{-\gamma_1 d_1}e^{-\gamma_4 d_2}}{(\gamma_0-\gamma_1-\gamma_4)(\gamma_3-\gamma_1)(\gamma_2-\gamma_4)}$$
$$+\frac{e^{-\gamma_3 d_1}e^{-\gamma_2 d_2}}{(\gamma_0-\gamma_3-\gamma_2)(\gamma_1-\gamma_3)(\gamma_4-\gamma_2)}$$
$$\left.+\frac{e^{-\gamma_3 d_1}e^{-\gamma_4 d_2}}{(\gamma_0-\gamma_3-\gamma_4)(\gamma_1-\gamma_3)(\gamma_2-\gamma_4)}\right), \quad \text{and}$$

$$\xi_\alpha(l_0|d)$$
$$=\gamma_0\gamma_1\gamma_2\gamma_3\gamma_4\left(e^{-\gamma_0 d_0}\left(\frac{e^{-\gamma_1(d_1-d_0)}e^{-\gamma_2(d_2-d_0)}}{(\gamma_1+\gamma_2-\gamma_0)(\gamma_3-\gamma_1)(\gamma_4-\gamma_2)}\left(d_0-\frac{1}{(\gamma_1+\gamma_2-\gamma_0)}\right)\right.\right.$$
$$+\frac{e^{-\gamma_1(d_1-d_0)}e^{-\gamma_4(d_2-d_0)}}{(\gamma_1+\gamma_4-\gamma_0)(\gamma_3-\gamma_1)(\gamma_2-\gamma_4)}\left(d_0-\frac{1}{(\gamma_1+\gamma_4-\gamma_0)}\right)$$
$$+\frac{e^{-\gamma_3(d_1-d_0)}e^{-\gamma_2(d_2-d_0)}}{(\gamma_3+\gamma_2-\gamma_0)(\gamma_1-\gamma_3)(\gamma_4-\gamma_2)}\left(d_0-\frac{1}{(\gamma_3+\gamma_2-\gamma_0)}\right)$$
$$\left.+\frac{e^{-\gamma_3(d_1-d_0)}e^{-\gamma_4(d_2-d_0)}}{(\gamma_3+\gamma_4-\gamma_0)(\gamma_1-\gamma_3)(\gamma_2-\gamma_4)}\left(d_0-\frac{1}{(\gamma_3+\gamma_4-\gamma_0)}\right)\right)$$
$$+\frac{e^{-\gamma_1 d_1}e^{-\gamma_2 d_2}}{(\gamma_0-\gamma_1-\gamma_2)^2(\gamma_3-\gamma_1)(\gamma_4-\gamma_2)}+\frac{e^{-\gamma_1 d_1}e^{-\gamma_4 d_2}}{(\gamma_0-\gamma_1-\gamma_4)^2(\gamma_3-\gamma_1)(\gamma_2-\gamma_4)}$$
$$+\frac{e^{-\gamma_3 d_1}e^{-\gamma_2 d_2}}{(\gamma_0-\gamma_3-\gamma_2)^2(\gamma_1-\gamma_3)(\gamma_4-\gamma_2)}$$
$$\left.+\frac{e^{-\gamma_3 d_1}e^{-\gamma_4 d_2}}{(\gamma_0-\gamma_3-\gamma_4)^2(\gamma_1-\gamma_3)(\gamma_2-\gamma_4)}\right), \quad \text{and}$$

$$\xi_\alpha(l_1|d)$$
$$=\gamma_0\gamma_1\gamma_2\gamma_3\gamma_4\left(e^{-\gamma_0 d_0}\left(\frac{e^{-\gamma_1(d_1-d_0)}e^{-\gamma_2(d_2-d_0)}}{(\gamma_1+\gamma_2-\gamma_0)(\gamma_3-\gamma_1)(\gamma_4-\gamma_2)}\right.\right.$$
$$\times\left(d_1-d_0-\frac{1}{(\gamma_0-\gamma_1-\gamma_2)}-\frac{1}{(\gamma_3-\gamma_1)}\right)$$
$$+\frac{e^{-\gamma_1(d_1-d_0)}e^{-\gamma_4(d_2-d_0)}}{(\gamma_1+\gamma_4-\gamma_0)(\gamma_3-\gamma_1)(\gamma_2-\gamma_4)}\left(d_1-d_0-\frac{1}{(\gamma_0-\gamma_1-\gamma_4)}-\frac{1}{(\gamma_3-\gamma_1)}\right)$$
$$\left.+\frac{e^{-\gamma_3(d_1-d_0)}e^{-\gamma_2(d_2-d_0)}}{(\gamma_3+\gamma_2-\gamma_0)(\gamma_1-\gamma_3)^2(\gamma_4-\gamma_2)}+\frac{e^{-\gamma_3(d_1-d_0)}e^{-\gamma_4(d_2-d_0)}}{(\gamma_3+\gamma_4-\gamma_0)(\gamma_1-\gamma_3)^2(\gamma_2-\gamma_4)}\right)$$
$$+\frac{e^{-\gamma_1 d_1}e^{-\gamma_2 d_2}}{(\gamma_0-\gamma_1-\gamma_2)(\gamma_3-\gamma_1)(\gamma_4-\gamma_2)}\left(d_1-\frac{1}{(\gamma_0-\gamma_1-\gamma_2)}-\frac{1}{(\gamma_3-\gamma_1)}\right)$$

$$+ \frac{e^{-\gamma_1 d_1} e^{-\gamma_4 d_2}}{(\gamma_0 - \gamma_1 - \gamma_4)(\gamma_3 - \gamma_1)(\gamma_2 - \gamma_4)} \left( d_1 - \frac{1}{(\gamma_0 - \gamma_1 - \gamma_4)} - \frac{1}{(\gamma_3 - \gamma_1)} \right)$$

$$+ \frac{e^{-\gamma_3 d_1} e^{-\gamma_2 d_2}}{(\gamma_0 - \gamma_3 - \gamma_2)(\gamma_1 - \gamma_3)^2(\gamma_4 - \gamma_2)} + \frac{e^{-\gamma_3 d_1} e^{-\gamma_4 d_2}}{(\gamma_0 - \gamma_3 - \gamma_4)(\gamma_1 - \gamma_3)^2(\gamma_2 - \gamma_4)} \Bigg).$$

$\xi_\alpha(l_2|d)$,$\xi_\alpha(l_3|d)$ and $\xi_\alpha(l_4|d)$ can be deduced by symmetry.

## 6.5 Explicit expression

One could use the previous inductive formulae with algebraic computation to generate the expressions for $q_\alpha$ and $\xi_\alpha$. However such a method is not efficient, since terms have to be merged for optimization. For instance, the inductive formula of $q_\alpha$ applied to Example (c) would lead before simplification to a sum of $2^K$ terms, while the simplified expression has only $K$ summands. We therefore give here explicit, already simplified formulae.

### *6.5.1 Vocabulary for tree combinatorics*

Example (d) shows that the formulae for $q_\alpha$ and $\xi_\alpha$ can be expressed as a sum of terms with a distinct structure. These in fact correspond to particular ‘slices’ or ‘cuts’ of the tree. In this section we define cuts and related nomenclature (illustrated in Fig. 2) which will be subsequently used to provide simplified symbolic expressions for $q_\alpha$ and $\xi_\alpha$.

The following definitions are given in the context of a *tree*, but they extend naturally to a *forest*, that is a set of trees.

**Definition 6.1** A *cut* of a tree is a maximal unordered set of nodes for the order $\prec$ defined above. In other words, $C$ is a cut of a tree $\mathcal{T}$ if it satisfies:

(1) $\forall i, j \in C, i \not\prec j$ and $j \not\prec i$.
(2) $\forall i \in T \setminus C, \exists j \in C, i \prec j$ or $j \prec i$.

*Example 6.1* In Example (d) above, the tree has five possible cuts which are: $\{0\}$, $\{1; 2\}$, $\{1; 4\}$, $\{3; 2\}$ and $\{3; 4\}$.

**Definition 6.2** We call *branch* every maximal sequence of nodes $(i_1, \dots, i_n)$ such that for all $k \in \{1, \dots, n-1\}$, the node $i_{k+1}$ is the unique child of $i_k$.

Every tree can then be visualized as a tree of *branches*, each having at least two children. However, a *cut* as defined above will still be a set of *nodes*, not *branches*. See Fig. 2 for an example.

Finally, as shown in the figure, we will talk about the “past”, the “present” and the “future” of a cut as follows: The “present” of a cut is the set of all nodes in the branches intersected by the cut. The “past” is the set of their ancestors in different branches, and the “future” the set of their descendants in different branches. More formally, we will adopt the following notations:

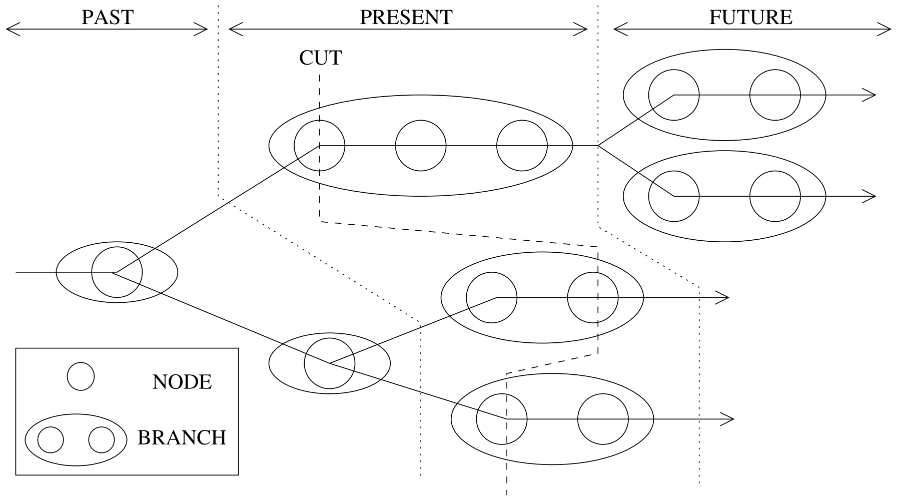


**Fig. 2** A tree with its branches and one of its cuts, with the corresponding past, present, and future of the cut

**Definition 6.3** For each pair of nodes $(i, j)$ in $T$, we write:

(a) $i \sim j$ if $i$ and $j$ belongs to the same branch and we say that $j$ belongs to the present of $i$ and $i$ belongs to the present of $j$.
(b) $i \ll j$ if $i \prec j$ and $i \nsim j$, and we say that $i$ belongs to the past of $j$ and $j$ belongs to the future of $i$.

The past, (resp. present, future) of a node will be the set of all nodes belonging to the past (resp. present, future) of this node, and by extension, the past (resp. present, future) of a cut will be the set of all the nodes belonging to the past (resp. present, future) of at least one of the nodes of the cut. We will denote these by **past**$(i)$, **present**$(i)$, **future**$(i)$ for a node $i$ and **Past**$(C)$, **Present**$(C)$, **Future**$(C)$ for a cut $C$. It is important not to confuse *nodes* and *branches*.

*Remark 6.1* The past, present and future of a cut forms a partition of $T$.

Finally, we extend the vector $d = (d_j)_{j\in V} \in \mathbb{R}^V$ to $d = (d_j)_{j\in T} \in \mathbb{R}^T$ by introducing for each $j$ in $T \setminus V$, $d_j := \min\{d_k \mid k \in V \text{ and } j \prec k\}$.

### 6.5.2 *The explicit form of $q_\alpha$*

In this section we use the cut vocabulary to define an expression for $q_\alpha$ which is not only closer to closed form, but is also more compact and more efficient to evaluate than that produced by the inductive formula. The validity of this formula is proved in the appendix.

For any fixed tree $\mathcal{T}$ and delay $d$, we have $q_\alpha(\mathcal{T}, d) = \Gamma_T \sum_{C \text{ cut of } \mathcal{T}} h_\alpha(\mathcal{T}, d, C)$, where $\Gamma_T := \prod_{j\in T} \gamma_j$ and where each term $h_\alpha(\mathcal{T}, d, C)$ can be expressed as a product of three factors:

$$h_\alpha(\mathcal{T}, d, C) := r(C)s(C)t(C),$$

where $r(C)$ depends only on the cut $C$ and its past, $s(C)$ depends only on $C$ and its present, and $t(C)$ depends only on $C$ and its future.

*(i) Past and present* The factors $r(C)$ and $s(c)$ are given by

$$r(C) := \prod_{k \in \mathbf{Past}(C)} \frac{1}{\gamma_k - \sum_{\substack{j \in C \\ k \ll j}} \gamma_j} \quad \text{and} \quad s(C) := \prod_{j \in C} \frac{e^{-\gamma_j d_j}}{\prod_{\substack{k \sim j \\ k \neq j}} (\gamma_k - \gamma_j)}.$$

*(ii) Future* The factor $t(C) = t(\mathcal{T}, d, C)$ is more complicated as it involves a recursion. To describe it, we regard $h_\alpha(\mathcal{T}, d, C)$ as functions of all its arguments to allow it to apply to subtrees with modified delays, and also extend its definition from a tree $\mathcal{T}$ to a forest $\mathcal{F}$. The set of nodes of a forest $\mathcal{F}$ is denoted by $F$.

If the future of each cut $C$ of $\mathcal{T}$ is empty, i.e. if the tree $\mathcal{T}$ is reduced to a single branch, we have $t(\mathcal{T}, d, C) = 1$. Recursively, we can then define:

$$t(\mathcal{T}, d, C) := \prod_{j \in C} t_j(\mathcal{T}, d, C),$$

where

$$t_j(\mathcal{T}, d, C) := \sum_{C_j \text{ cut of } \mathcal{F}_j} \frac{h_\alpha(\mathcal{F}_j, d - (d_j), C_j)}{(\sum_{k \in C_j} \gamma_k) - \gamma_j},$$

where $\mathcal{F}_j$ is the sub-forest of $\mathcal{T}$ containing all the nodes belonging to the future of j, and therefore $F_j := \{k \in T | j \ll k\}$, where $d - (d_j)$ is the vector $(d_k - d_j)_{k \in F_j}$.

We can interpret $d - (d_j)$ as the best information we have about the delay between $j$ and the leaves, since we know only that the delay between the root and $j$ has to be smaller than any delay between the root and the leaves in the future of $j$.

The formula $q_\alpha(\mathcal{T}, d)$ is now entirely defined. One can verify that it gives the correct expressions for the examples in the paper.

### *6.5.3 The explicit form of $\xi_\alpha(l|d)$*

As we can see in the previous examples, the term $\xi_\alpha(l_i|d)$ has globally the same structure as $q_\alpha(d)$ with additional factors. Therefore it is relatively easy to modify any algorithm computing $q_\alpha(d)$ to compute $\xi_\alpha(l_i|d)$.

For any non-fixed tree $\mathcal{T}$ and any node $i$ in $T$, we have:

$$\xi_\alpha(l_i|d) = \Gamma_T \sum_{C \text{ cut of } \mathcal{T}} h^i_\alpha(\mathcal{T}, d, C, 0),$$

where for any real number $x$:

$$h^i_\alpha(\mathcal{T}, d, C, x) := r^i(C)s^i(C)t^i(\mathcal{T}, d, C, x),$$

where $r^i(C)$ (resp. $s^i(C)$) depends only from the cut $C$ and its past (resp. present), and where $t^i(\mathcal{T}, d, C, x)$ involves a recursion.

*(a) Past and present*

$$r^i(C) := \prod_{k \in \mathbf{Past}(C)} \frac{1}{(\gamma_k - \sum_{\substack{j \in C \\ k \ll j}} \gamma_j)^{1+\delta_k^i}} \quad \text{and} \quad s^i(C) := \prod_{j \in C} \frac{e^{-\gamma_j d_j}}{\prod_{\substack{k \sim j \\ k \neq j}} (\gamma_k - \gamma_j)^{1+\delta_k^i}},$$

where

$$\delta_k^i = \begin{cases} 1 & \text{if } k = i \\ 0 & \text{else} \end{cases}$$

is the Kronecker delta.

*(b) Future* As for $t(\mathcal{T}, d, C)$, the definition of $t^i(\mathcal{T}, d, C, x)$ induce a recursion with the whole formula. First we introduce

$$\rho_i(C) := \sum_{k \in \mathbf{past}(i)} \frac{1}{\gamma_k - \sum_{\substack{j \in C \\ k \ll j}} \gamma_j} \quad \text{and} \quad \sigma_i(C) := \sum_{\substack{k \sim i \\ k \neq i}} \frac{1}{\gamma_k - \gamma_i}.$$

These are terms corresponding respectively to the past and the present of node $i$. Now $t^i(\mathcal{T}, d, C, x) := \prod_{j \in C} t_j^i(\mathcal{T}, d, C, x)$, where:

$$t_j^i(\mathcal{T}, d, C, x) := \begin{cases} (d_i - \sigma_i(C) - \rho_i(C) - x) \quad \text{if } \mathbf{future}(j) = \emptyset \text{ and } j = i \\ 1 \quad \text{if } \mathbf{future}(j) = \emptyset \text{ and } j \neq i \\ \sum_{C_j \text{ cut of } \mathcal{F}_j} \frac{h_\alpha^i(\mathcal{F}_j, d-(d_j), C_j, 0)}{(\sum_{k \in C_j} \gamma_k) - \gamma_j} u^i(\mathcal{T}, d, C, C_j, x) \\ \quad \text{if } \mathbf{future}(j) \neq \emptyset \text{ and } j = i \\ \sum_{C_j \text{ cut of } \mathcal{F}_j} \frac{h_\alpha^i(\mathcal{F}_j, d-(d_j), C_j, \tau_j(C_j))}{(\sum_{k \in C_j} \gamma_k) - \gamma_j} \quad \text{if } \mathbf{future}(j) \neq \emptyset \text{ and } j \neq i, \end{cases}$$

where $u^i(\mathcal{T}, d, C, C_j, x) := (d_i - \sigma_i(C) - \rho_i(C) - x + \tau_i(C_j))$, and where for any node $j$ and cut $C$, $\tau_j(C) := \frac{1}{\gamma_j - \sum_{k \in C} \gamma_k}$.

*Remark 6.2* It is important to remember in which tree each term is computed. When it is not specified, it is implicitly the tree called $\mathcal{T}$ on the current level of recursion.

We notice that the term $x$ is used only in the cases where $j = i$. Therefore, the term $\tau_j(C_j)$ in $h_\alpha^i(\mathcal{F}_j, d - (d_j), C_j, \tau_j(C_j))$ above is used only when $i \in C_j$.

In order to understand the different cases in the definition of $t_j^i(\mathcal{T}, d, C, x)$, we can have a look at the Example (d). In this example, the term $(d_0 - \frac{1}{\gamma_1+\gamma_2-\gamma_0})$ in $\xi_\alpha(l_0|d)$ comes from $(d_i + \tau_i(C_j))$ in $u^i(C, C_j, x)$, while the term $(d_1 - d_0 - \frac{1}{\gamma_0-\gamma_1-\gamma_2} - \frac{1}{\gamma_3-\gamma_1})$ in $\xi_\alpha(l_1|d)$ comes from $(d_i - x - \sigma_i(C))$ with $x = \tau_j(C_j)$ and $C = C_j$ (recursive call), and the term $(d_1 - \frac{1}{\gamma_0-\gamma_1-\gamma_2} - \frac{1}{\gamma_3-\gamma_1})$ comes from $(d_i - \rho_i(C) - \sigma_i(C))$.

### 6.5.4 Alternative informal description

A less formal way to describe $\xi_\alpha(l_i|d)$ is to consider each term in the expanded expression of $q_\alpha(d)$, and each time a $\frac{e^{-\gamma_i d}}{\text{coef}}$ appears, multiply it by $(d - \text{coef}')$, where $\text{coef}'$ is the sum of each multiplicative factor appearing in coef and containing $\gamma_i$ in its expression, and each factor being multiplied by $(-1)$ when $\gamma_i$ appears in it with a positive sign. Further, when a term in the expanded expression of $q_\alpha(d)$ is of the form $\frac{e^{-\gamma_{j_1} d} \dots e^{-\gamma_{j_n} d}}{\text{coef}}$, replace coef with $\text{coef}''$, being equal to coef where all the factors containing $\gamma_i$ are squared.

## 6.6 Implementation

We can finally compute $\mathbb{E}_\alpha(l_i|d) = \frac{\xi_\alpha(l_i|d)}{q_\alpha(d)}$. Our implementation is in the programming language *Objective Caml*. Only the standard packages of the language were used.

We need to compute $\varphi(\alpha) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_\alpha(l|d(i))$ at each step of the EM algorithm. Therefore, the formula $\mathbb{E}_\alpha(l|d)$ has to be efficiently calculated for a fixed tree $\mathcal{T}$ and fixed $\alpha$, but for $N$ distinct values of $d$, $N$ being the number of probes used in the experiment which may not be fixed in advance. It follows that the best way to compute this formula efficiently is to generate the symbolic expression of $\mathbb{E}_\alpha(l|d)$ with $\alpha$ known and $d$ unknown parameters, to simplify it as most as possible and then to compute it for each of the $N$ values $d(i)$.

Efficiency is further improved if $d = (d_j)$ is precomputed for all $j \in T$, and also if the differences $d_i - d_j$ are precomputed and kept easy to access, since they appear frequently and keep the same value at each different step.

Our program generated the symbolic expression of $\mathbb{E}_\alpha(l|d)$ as a symbolic tree. The factors depending only on $\alpha$ were evaluated and simplified during the generation of the tree. The tree was then reduced as much as possible and was finally evaluated for each $d(i)$ using the precomputed matrix $M(i)$.

A first sanity check for program correctness, which is easy to perform, is to exploit the relation $d_k = \mathbb{E}_\alpha(d_k|d) = \sum_{j \preceq k} \mathbb{E}_\alpha(l_j|d)$ for all $k$ in $V$. If the program is correct, this has to be verified for any value of $d$ and any trees $\mathcal{T}$.

## 6.7 Size of the expression and complexity of the EM step

We will only consider the size of the explicit expression of $q_\alpha$ in the particular cases of a unary tree and of a binary tree. In the former case, the size is clearly linear, while in the latter it is exponential in the number of nodes, and thus doubly exponential in the height of the tree. Here, we measure the size of the expression in the number of exponentials appearing in it after reduction, since all the other factors, especially those involving $\gamma$, can be precomputed.

For a unary tree with $K$ nodes, the number of exponentials is exactly $K$, and the size is then linear. For a binary tree of height $h+1$, the size $S$ is given by $S(h+1) = 2 * S(h)^2$, since the terms of the two subtrees of height $h$ are multiplied (giving $S(h)^2$) and are integrated, giving twice as many terms. Since $S(1) = 1$, we deduce that $S(h) = 2^{2^{h-1}-1}$. Since the number of node in a binary tree of height $h$ is $2^{h+1} - 1$, the size is then exponential in the number of nodes.

We get the complexity of one step of EM by multiplication of this size by the number of nodes in the tree and the length of data, since we need to compute $\mathbb{E}_\alpha(l_j|d(i))$ for all $j$ in $T$ and all $i$ in $\{1, \dots, N\}$. It is, however, possible to factorize the computation of the exponentials shared by the expressions, but the number of multiplications will still be the same.

Finally, the number of steps to convergence is likely to grow with the number of nodes as well, which increase further the global complexity of the EM algorithm. This motivates the speed-up technique presented in Sect. 8.

## 7 Results

We conducted a series of experiments on different kinds of trees. The data were generated by simulating random delays in a tree using the known ground truth. Except in Sect. 7.3, each experiment was conducted on a data set of $N = 10^4$ samples and repeated 200 times.

### 7.1 Unary tree case

This case was first studied in [1] and [10], where two and three nodes cases were tested.

*(a) Two node case* In the simple case of a tree with two nodes, we have some additional results on the convergence of the EM sequence. Property 5.1 becomes $\hat{\alpha}_1^{(k+1)} + \hat{\alpha}_2^{(k+1)} = \bar{d}$. This relation implies that there is only one unknown value, and thanks to this, it is possible to prove the following result by using the intermediate value theorem.

**Theorem 7.1** *In the two node case, the sequence* $(\hat{\alpha}_1^{(k)}, \hat{\alpha}_2^{(k)})$ *converges to a finite limit which is a solution of the likelihood equation* (2).

The general EM properties (usually) ensure that the sequence $(L_d(\hat{\alpha}^{(k)}))_{k\in\mathbb{N}}$ converges to a finite limit. This theorem goes further and ensures that the sequence $(\hat{\alpha}_1^{(k)}, \hat{\alpha}_2^{(k)})$ converges too, which is not a classical result for EM.

The detailed proof can be found in [1] and [10], and uses the fact that, since (for all $k \geq 1$) $\hat{\alpha}_1^{(k)} + \hat{\alpha}_2^{(k)} = \bar{d}$, the likelihood $L_d(\hat{\alpha}^{(k)})$ can be expressed as a function of $\hat{\alpha}_1^{(k)}$ alone. Therefore, the proof cannot be generalized to more than two nodes. Table 1 gives some results obtained in this case.

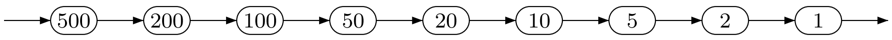
*(b) Nine nodes case* $(U9)$ Table 2 shows the results for the following unary tree with 9 nodes.

The main difficulty for estimating such unary trees comes from the fact that we get the same information for all the nodes. In particular, the estimation can only give the values of the mean delays modulo an unknown permutation. We will also see in Sect. 8 that the convergence of EM for this tree is very slow.

**Table 1** Experimental results of the EM estimator $(\hat{\alpha}_1, \hat{\alpha}_2)$ for various ground truths in the 2-node case

| Gr. truth | Mean | 10% percentile | 90% percentile | Variance$^{1/2}$/Mean |
|---|---|---|---|---|
| (1.1, 1) | (1.114, 0.987) | (1.044, 0.881) | (1.220, 1.057) | (0.0644, 0.0725) |
| (2, 1) | (1.999, 1.003) | (1.933, 0.946) | (2.063, 1.063) | (0.0244, 0.0446) |
| (10, 1) | (10.003, 1.003) | (9.866, 0.942) | (10.148, 1.070) | (0.0115, 0.0501) |
| (100, 1) | (100.044, 1.015) | (98.782, 0.827) | (101.368, 1.214) | (0.0104, 0.1514) |

**Table 2** Experimental results obtained for a unary tree with 9-nodes



| Gr. truth | 500 | 200 | 100 | 50 | 20 | 10 | 5 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| Mean | 498.42 | 194.98 | 99.59 | 49.88 | 26.10 | 10.55 | 5.38 | 4.25 | 3.79 |
| 10%-tile | 481.47 | 150.08 | 48.31 | 13.93 | 5.78 | 1.54 | 0.099 | 0.087 | 0.077 |
| 90%-tile | 512.30 | 236.78 | 155.35 | 79.58 | 56.20 | 24.94 | 12.19 | 10.86 | 9.99 |
| $\sigma$/Mean | 0.025 | 0.175 | 0.428 | 0.486 | 0.769 | 0.990 | 0.897 | 0.904 | 0.974 |

## 7.2 General case

We present here some results obtained for other different trees.

*(a) Binary tree of height 3* $(B1H3)$ In the case of a binary tree, the estimation are accurate too, with again a slight bias on the smallest values, as shown in Table 3. The estimations are in a sense easier for this tree than for the unary ones because we get more information and also because each node can be discriminated from the others, while in the unary trees all the nodes are equivalent.

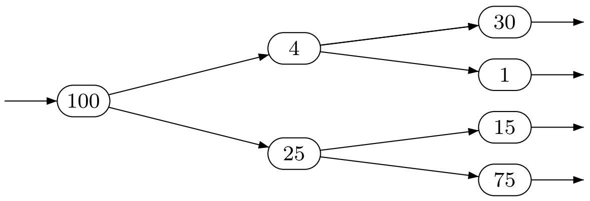
*(b) Binary tree of height 4* $(B1H4)$ Table 4 shows the results for this tree: the estimation are still very good. The difficulty for bigger binary trees arise from the complexity of the EM step computation rather than the accuracy of the estimation. We made some simulation on a tree of height 5 and obtained good estimations too, but they were very long to compute.

*(c) Tree with branches* $(B3H2)$ The results for this tree are given in Table 5. Trees such as the last one, with several parallel branches, are the hardest to estimate because they combine both difficulties seen in unary and binary trees: a difficult estimation in each branch where we have the same information for all nodes, and a high complexity of the EM step because of the multiple parallel subtrees.

## 7.3 Speed of convergence

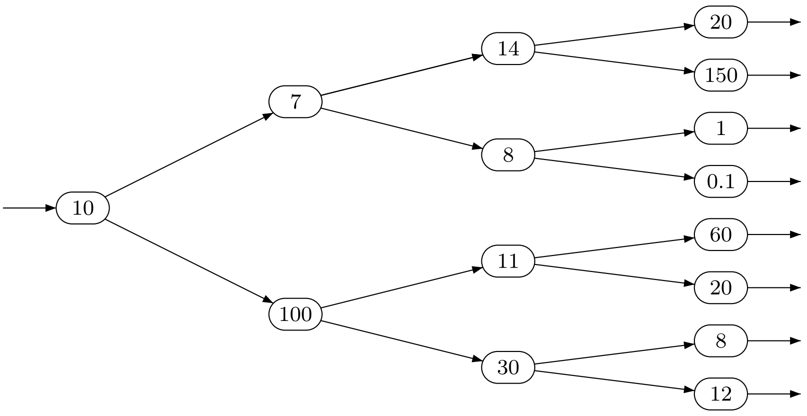
All previous results were conducted with sample size $N = 10^4$. Figure 3 studies the speed of convergence of the maximum likelihood estimator with respect to $N$. In order to save space, we present results only for the tree $B1H3$, but results are similar

**Table 3** Experimental results obtained for a binary tree of height 3



| Ground truth | 100 | 4 | 30 | 1 | 25 | 15 | 75 |
|---|---|---|---|---|---|---|---|
| Mean | 100.03 | 4.01 | 29.99 | 0.9999 | 25.007 | 14.996 | 74.995 |
| 10% percentile | 99.62 | 3.95 | 29.86 | 0.98 | 24.85 | 14.88 | 74.70 |
| 90% percentile | 100.42 | 4.04 | 30.12 | 1.02 | 25.14 | 15.11 | 75.35 |
| Var$^{1/2}$/Mean | 0.0032 | 0.0092 | 0.0031 | 0.012 | 0.0045 | 0.0060 | 0.0034 |

**Table 4** Experimental results obtained for a binary tree of height 4



| Ground truth | 10 | 7 | 14 | 20 | 150 | 8 | 1 | 0.1 |
|---|---|---|---|---|---|---|---|---|
| Mean | 10.01 | 6.99 | 13.99 | 20.03 | 149.95 | 7.999 | 1.0004 | 0.0999 |
| 10% percentile | 9.67 | 6.66 | 13.52 | 19.50 | 147.77 | 7.80 | 0.99 | 0.096 |
| 90% percentile | 10.35 | 8.14 | 15.32 | 20.98 | 152.17 | 8.29 | 1.71 | 0.28 |
| Variance$^{1/2}$/Mean | 0.03 | 0.04 | 0.03 | 0.02 | 0.01 | 0.02 | 0.01 | 0.01 |
| Ground truth | | 100 | 11 | 60 | 20 | 30 | 8 | 12 |
| Mean | | 100.09 | 11.30 | 60.13 | 20.01 | 30.03 | 8.002 | 12.01 |
| 10% percentile | | 99.83 | 10.56 | 59.8 | 19.59 | 29.58 | 7.86 | 11.85 |
| 90% percentile | | 102.16 | 13.13 | 61.47 | 21.31 | 31.65 | 10.27 | 14.19 |
| Variance$^{1/2}$/Mean | | 0.01 | 0.03 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |

**Table 5** Experimental results with the tree "$B1H3$" below

$$\begin{tikzcd}
 & & & 6 \arrow[r] & 5 \arrow[r] & 4 \arrow[r] & {} \\
{} \arrow[r] & 20 \arrow[r] & 2 \arrow[r] & 1 \arrow[ur] \arrow[dr] & & & \\
 & & & 80 \arrow[r] & 60 \arrow[r] & 3 \arrow[r] & {}
\end{tikzcd}$$

| Ground truth | 20 | 2 | 1 | 6 | 5 | 4 | 80 | 60 | 3 |
|---|---|---|---|---|---|---|---|---|---|
| Mean | 19.86 | 2.63 | 1.79 | 6.30 | 4.79 | 2.63 | 78.80 | 61.34 | 1.63 |
| 10% percentile | 19.40 | 1.50 | 0.68 | 5.18 | 3.09 | 0.62 | 70.31 | 54.70 | 0.0052 |
| 90% percentile | 20.35 | 3.92 | 2.98 | 7.58 | 6.03 | 4.86 | 84.89 | 70.28 | 3.74 |
| $\mathrm{Var}^{1/2}/\mathrm{Mean}$ | 0.021 | 0.36 | 0.48 | 0.15 | 0.24 | 0.59 | 0.088 | 0.012 | 0.096 |

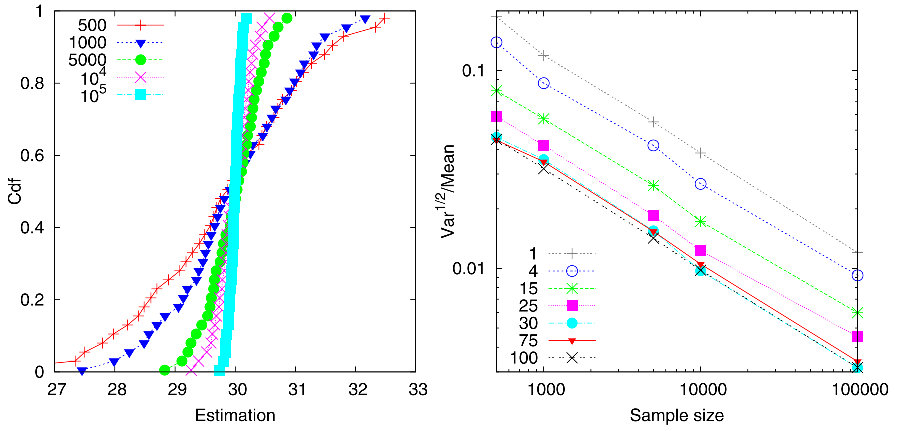


**Fig. 3** *Left plot*: the cumulative distribution function of the maximum likelihood estimator of node 30 for different sample sizes. The *right-side* shows the relative standard error for all nodes on the same tree $B1H3$, depending on the sample size

for the other nodes and trees. One might notice that smaller sample size can lead to good results: with as few as 1000 probes, 90% of the experiments estimate node 30 with less than 10% error. On the right plot, the relative standard error (i.e. the square root of the variance, divided by the mean) are parallel lines of slope $-\frac{1}{2}$, which means that the standard error decreases as $\frac{1}{\sqrt{N}}$ and the variance as $\frac{1}{N}$.

## 7.4 Comparison to the least squares method

In [13], Lawrence et al. discard MLE based approaches due to their computation time in favor of moment based methods using least squares. Table 6 presents the results obtained by their method for 200 independent experiments with the same sample size $N = 10^4$, for the tree $B1H3$. This shall be compared with the results of MLE in Table 3. Other trees lead to similar results, and were omitted for space reasons. As expected, the MLE approach yields better results, especially for nodes that have a small

**Table 6** Experimental results obtained for $B1H3$, using the least squares method from [13]

| Ground truth | 100 | 4 | 30 | 1 | 25 | 15 | 75 |
|---|---|---|---|---|---|---|---|
| Mean | 99.98 | 4.04 | 29.93 | 0.93 | 24.77 | 15.18 | 75.07 |
| 10% percentile | 98.22 | 2.42 | 28.73 | −0.19 | 22.58 | 13.92 | 73.54 |
| 90% percentile | 101.91 | 5.46 | 31.20 | 2.21 | 26.77 | 16.31 | 76.65 |
| $\mathrm{Var}^{1/2}/\mathrm{Mean}$ | 0.014 | 0.32 | 0.033 | 1.009 | 0.060 | 0.062 | 0.017 |

**Table 7** Experimental results obtained for the tree $B1H3$, with imperfect measurements

| Ground truth | 100 | 4 | 30 | 1 | 25 | 15 | 75 |
|---|---|---|---|---|---|---|---|
| Mean | 98.1 | 3.6 | 32.3 | 3.3 | 25.8 | 16.1 | 76.1 |
| 10% percentile | 96.7 | 3.4 | 31.9 | 3.2 | 25.4 | 15.8 | 75.2 |
| 90% percentile | 99.3 | 3.9 | 32.8 | 3.5 | 26.2 | 16.5 | 77.1 |
| $\mathrm{Var}^{1/2}/\mathrm{Mean}$ | 0.01 | 0.05 | 0.01 | 0.04 | 0.01 | 0.02 | 0.01 |

delay. However, the main advantage of moment based methods is their speed. The simulation of the 200 experiments took only about one minute for the least squares approach, whereas our algorithm needed 45 minutes. This difference increases for larger trees.

The complexity of an estimation technique can be expressed in two ways: the number of independent probes needed to reach a given precision, and the time needed to compute the estimator based on those probes. The relative interest of each method will depend on which of these two steps is the most crucial for the specific application considered.

### 7.5 Resilience to measurement noise and imperfect models

They are many ways to introduce model or measurement errors: a full paper could be written on this topic. We will present only one case, which we hope is representative. Table 7 presents the results of the accelerated EM algorithm, when each end-to-end delay was sampled according to the model distribution, then multiplied by a i.i.d. uniform value between 0.95 and 1.05. The results are worse: the standard error is about three times higher for most nodes. However, the errors stay in a reasonable level, which might indicate that the algorithm is resilient to errors.

## 8 Steered jumping for EM

In a number of cases, especially when the tree has long branches, the number of steps before converging to a fixed point can be very large, and since the complexity of each step is proportional to the length of data, the EM algorithm becomes very slow when $N$ becomes large. The complexity of each step rises also very quickly with the size of the tree, since the growth is quadratic in the number of nodes for an unary tree,

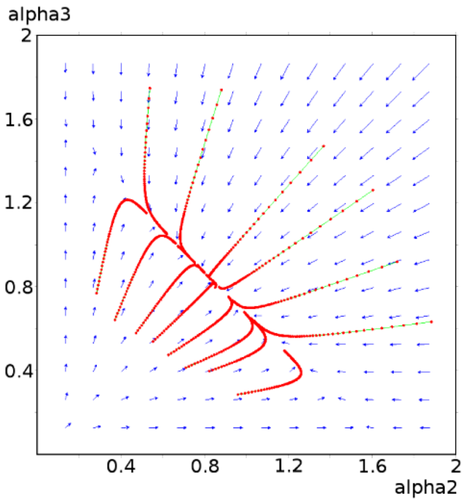


**Fig. 4** Plot of some trajectories of the step function for the ground truth $\alpha = (0.1, 1, 10)$ and for a sample of $N = 1000$ data. The sequences of point are sequences of iterations of the algorithm. The *small arrows* represents the directions of $\hat{\alpha}^{(k+1)} - \hat{\alpha}^{(k)}$

but can be exponential for a binary tree (see Sect. 6.7). It is therefore important for this problem to significantly improve the convergence speed of the EM algorithm. We present a novel such method below.

## 8.1 Analysis of the iteration

We first illustrate some characteristics of the iteration through two examples.

*Example one* Consider a unary tree with ground truth $\alpha = (0.1, 1, 10)$. From Property 5.1, $\hat{\alpha}^{(k)} = (\hat{\alpha}_1^{(k)}, \hat{\alpha}_2^{(k)}, \hat{\alpha}_3^{(k)})$ obeys $\hat{\alpha}_1^{(k)} + \hat{\alpha}_2^{(k)} + \hat{\alpha}_3^{(k)} = \bar{d}$, so that the system has only two independent variables. We therefore plot the trajectories of the EM algorithm in a $(\hat{\alpha}_2^{(k)}, \hat{\alpha}_3^{(k)})$ plot.

Figure 4 shows some sequences of iterations of the algorithm from different initial conditions. We see that the trajectories all converge toward the same point, $\hat{\alpha} = (0.83, 0.83, 8.07)$. During this experiment, the mean delay was $\bar{d} = 9.73$, and Property 5.1 was respected. We also observe that the trajectories seems to converge quickly towards a straight line of equation $x = 1.7 - y$, and once on this line, the steps becomes very small and the convergence is much slower.

*Example two* In this case $\alpha = (1, 1/3, 0.1, 0.01)$.

Figure 5 shows the trajectory of the two smallest coordinates of the sequence $\hat{\alpha}^{(k)}$ for a sample of $N = 10000$ data, with initial condition the ground truth itself. The algorithm stopped after 39113 steps. The red points show the trajectory after 0 steps (starting point), 1000 steps, 10000 steps and 39113 steps when the algorithm finally stopped.

As we can see, the trajectory again mostly falls on a straight line, and not unexpectedly, the size of each step drops while the trajectory approaches the final point. Table 8 shows the evolution of the log-likelihood—it increases extremely slowly.

The extremely small rate of increase in likelihood as a function of $\hat{\alpha}^{(k)}$ along the trajectories means that the usual termination criteria, consisting in stopping when

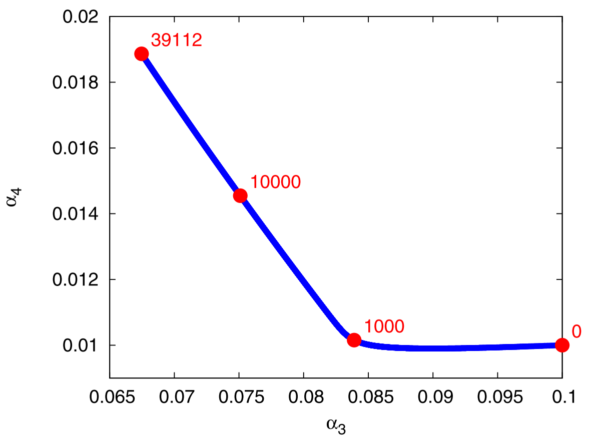


**Fig. 5** Plot of the values of the two smallest coordinate of each $\hat{\alpha}^{(k)}$ for the ground truth $\alpha = (1, 1/3, 0.1, 0.01)$

**Table 8** Evolution of the log-likelihood corresponding to the trajectory on the left side; it increases extremely slowly with $k$

| Number of steps | Log-likelihood |
|---|---|
| 0 | −1.240947 |
| 1 | −1.240910 |
| 10 | −1.240870 |
| 100 | −1.240831 |
| 1000 | −1.240803 |
| 10000 | −1.240787 |
| 20000 | −1.240783 |
| 39113 | −1.240782 |

$\|\hat{\alpha}^{(k+1)} - \hat{\alpha}^{(k)}\| < \varepsilon$, or $L_d(\hat{\alpha}^{(k+1)}) - L_d(\hat{\alpha}^{(k)}) < \varepsilon$ for some small $\varepsilon$, or after a fixed number of iterations, can result in significant errors. The criterion we used to avoid these traps was to stop when $L_d(\hat{\alpha}^{(k+1)}) \le L_d(\hat{\alpha}^{(k)})$, which in theory never happens but occurs in practice because of numerical errors very close to the fixed point.

These two examples illustrate two key characteristics of the EM algorithm (for this problem). First, the trajectory reaches an area relatively close to the final point relatively quickly, where it enters a ‘glide path’ which is relatively linear, corresponding in a sense to a valley of the function $-\log L_d(\theta)$, or ‘reversed valley’ of $\log L_d(\theta)$ (for the sake of simplicity, we will abusively refer to it as a ‘valley’). Secondly, once in the ‘valley’, the speed of the trajectory becomes particularly slow. These two observations inspire the following strategy to accelerate EM: (i) reach a good first approximation as quickly as possible, (ii) once near the ‘valley’, increase the size of the steps to go faster.

## 8.2 The sampling method

This section addresses point (i) above. Our objective is to reach the ‘valley’ leading to the fixed point extremely quickly, using the intuition that even a rough method should be able to achieve this objective.

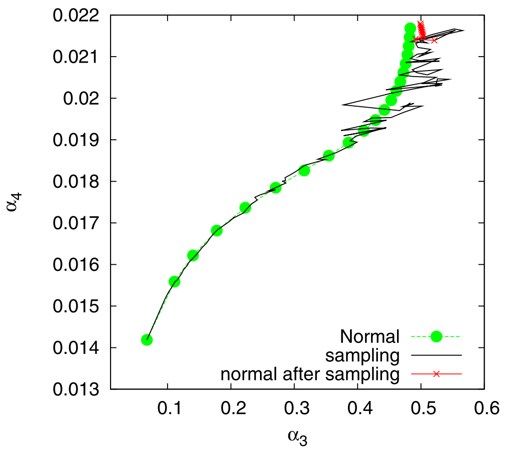


**Fig. 6** Comparison of the first 100 steps of a normal trajectory ("normal", every fifth step is shown) with 100 steps of the sampling method ("sampling"), followed by 10 normal steps ("normal after sampling"), from a random starting point. The ground truth $\alpha = (1, 1/3, 0.1, 0.01)$ and data length $N = 10000$ is as in Fig. 5, but the data set is different

The method consists in cutting the data (of length $N$) into $N/k$ subsets of equal length $k$ (for simplicity we assume that $k$ divides $N$). We then compute some iterations of the EM algorithm using only one of the subsets as data, the subset being chosen randomly at each iteration.

More precisely, at each step, instead of computing the usual iteration $\hat{\alpha}^{(k+1)} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\hat{\alpha}^{(k)}}(l|d(i))$ we compute one of the following iterations:

$$\hat{\alpha}^{(k+1)} = \frac{1}{k} \sum_{i=jk+1}^{(j+1)k} \mathbb{E}_{\hat{\alpha}^{(k)}}\big(l|d(i)\big),$$

where at each step, $j$ is uniformly at random chosen among $\{0, \ldots, N/k - 1\}$.

The advantage of this method is that $N/k$ steps based on subsets will cost only as much as 1 step using the full data, and yet gives a fair first approximation of the fixed point. This is a way to sacrifice precision for speed, but since we only want a first approximation here, low precision is acceptable. In particular, during these cheap steps, the likelihood of the parameters does not necessarily increase, but the parameters does get closer from the final point.

Several choices of $k$ are possible: in this paper, we used $k = \sqrt{N}$ ($N = 10000$ and $k = 100$), which worked well in practice. Any other choice (as long as $k$ is "large enough to be representative, but not too large to gain computation time") makes sense.

As a non-rigorous intuition that these cheap steps will still go in the right direction, note that since the integer $j$ is randomly chosen, the expectation of each random step is:

$$\mathbb{E}_j\big(\hat{\alpha}^{(k+1)}\big) = \frac{k}{N} \sum_{j=0}^{N/k-1} \frac{1}{k} \sum_{i=jk+1}^{(j+1)k} \mathbb{E}_{\hat{\alpha}^{(k)}}\big(l|d(i)\big) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{\hat{\alpha}^{(k)}}\big(l|d(i)\big).$$

So, on average, each cheap step goes in the same direction as a normal step.

Figure 6 shows an example of this method. As we can see the first iterations move in the same direction as the normal EM steps, and 100 iterations of the sampling method leads to a point close to the one obtained after 100 normal steps, but costing only as much as *one* normal step to get there.

## 8.3 The steered jumping method

This section addresses point (ii) above. Namely, once we get a first approximation of the fixed point which is close to or within the 'valley', the steps usually become very small but the trajectory is linear. Our strategy is to exploit this linearity to increase step size dramatically. We will present the method in a more general context, since we believe it could be applied to other cases where the same kind of convergence issues are encountered.

### *8.3.1 General context*

We are given a function $F$ and we want to find a local maximum by using an iterative algorithm (e.g. the EM algorithm, gradient ascent) which can be expressed as follows: Starting from some point $\hat{\alpha}^{(0)}$, construct the sequence $(\hat{\alpha}^{(0)}, \hat{\alpha}^{(1)}, \dots)$ defined by the recursive formula

$$\hat{\alpha}^{(k+1)} = \hat{\alpha}^{(k)} + \Delta_k, \tag{13}$$

where the parameter $\Delta_k$ is chosen such that $F(\hat{\alpha}^{(k)}) \le F(\hat{\alpha}^{(k+1)})$, with equality if and only if $\hat{\alpha}^{(k)}$ is a stationary point of $F$. The algorithm stops when the equality is reached.

The way to compute the parameter $\Delta_k$ depends on the chosen algorithm. In the case of the EM algorithm, we have $\Delta_k = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\hat{\alpha}^{(k)}}(l|d(i)) - \hat{\alpha}^{(k)}$. In the case of a gradient ascent, we have $\Delta_k = \delta_k \nabla F(\hat{\alpha}^{(k)})$, where $\nabla F$ is the gradient of $F$, and where $\delta_k$ is some well chosen positive scalar number.

### *8.3.2 Jumping method*

In a case where the behavior of such an algorithm is as in Fig. 5, namely linear and very slow, we would like to take much larger steps. More precisely, we would like to replace the last equation (13) by

$$\hat{\alpha}^{(k+1)} = \hat{\alpha}^{(k)} + \beta_k \Delta_k, \tag{14}$$

with $\beta_k \ge 1$ and hopefully much bigger than 1, such that the relation $F(\hat{\alpha}^{(k)}) \le F(\hat{\alpha}^{(k+1)})$ still holds at each iteration. In some sense, if $\beta_k = n \in \mathbb{N}$ we can interpret this as assuming that $\Delta_k \simeq \Delta_{k+1} \simeq \cdots \simeq \Delta_{k+n}$ and approximating all of them by $\Delta_k$, and then computing $n$ steps in one. We then say that we "jump" with a factor $\beta_k$. Figure 7 shows an example of this method applied to the example of Fig. 5.

This idea is not new as such, and it has been applied to the EM algorithm in [9] as a generalized conjugate gradient algorithm, and in [17] as an overrelaxed bound optimization.

The method as described above has two main flaws. First, we do not know how to choose the values of $\beta_k$ efficiently at each step. Secondly, and most importantly, it sometimes results in a behavior similar to the one visible on Fig. 7. If we try to jump

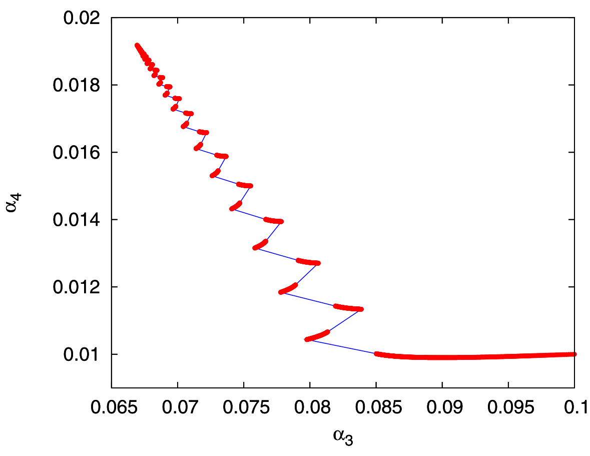


**Fig. 7** Trajectory using the jumping method, projected onto the two smallest coordinates of $\hat{\alpha}^{(k)}$, for the ground truth $\alpha = (1, 1/3, 0.1, 0.01)$ using the same data as in Fig. 5. In this case we tried to jump every 100 steps, i.e. we had $\beta_k = 1$ except when $k = 0 \pmod{100}$ where we set $\beta_k = 1000$ if $F(\hat{\alpha}^{(k)}) \le F(\hat{\alpha}^{(k)} + \beta_k \Delta_k)$, or $\beta_k = 1$ otherwise. The visible gaps between points reveal where jumps actually occurred. The total number of iterations before reaching the minimum was 14684, compared to 39112 for the normal EM

too far from one side of the 'valley', we end up on the other side rather than reaching and tracking the 'valley' floor, resulting in an inefficient zigzag trajectory. In other words, increasing step size can cause instability. To counter this, we next modify not only the size of the steps but also their direction.

### *8.3.3 Steered jumping method*

We modify slightly the formula (14) of the jumping method to read:

$$\hat{\alpha}^{(k+1)} = \hat{\alpha}^{(k)} + \beta_k C_k \Delta_k, \tag{15}$$

where $C_k$ will be a well chosen matrix such that $F(\hat{\alpha}^{(k)}) \le F(\hat{\alpha}^{(k+1)})$ still holds. The choice of $C_k$ can depend on many parameters, like the current state of the algorithm but also its past iterations. It is important to notice that it is always possible to try different choices of $C_k$ and $\beta_k$ and chose the one which gives the highest value of $F(\hat{\alpha}^{(k)} + \beta_k C_k \Delta_k)$.

In our case, we would like the jumps to be in the same direction as the 'valley'. For this, we use at each step the information given by the previous iterations of the algorithm about the global shape of the 'valley' to compute the matrix $C_k$, via Principal Component Analysis (PCA). This was inspired by a method recently developed in robotics to accelerate the growth of Rapidly-exploring Random Trees (RRT) for path finding [6].

### *8.3.4 A short introduction to PCA*

Principal Component Analysis is a method used to find the main axes of concentration of a set of points in a high dimensional space.

Consider a set of points $X = (x_1, \dots, x_n)$, each $x_i$ being a point of $\mathbb{R}^d$. We construct the $d \times d$ matrix $C$ called the *covariance matrix* of the set of points $X$, whose element $(i, j)$ is $(x_i - \bar{x})(x_j - \bar{x})$, where $\bar{x} := \frac{1}{N}\sum_{i=1}^{N} x_i$ is the mean point of the set. The matrix is symmetric positive semi-definite, and has the property that it captures very well the repartition of the points $x_i$ in the space, since it is diagonalizable by the Cayley–Hamilton theorem. The eigenvectors $(e_1, \dots, e_k)$ associated to its biggest eigenvalues $(\lambda_1, \dots, \lambda_k)$ $(k \le d)$ are the axes where the points are the most dispersed.

PCA usually consist in the computation of the $(e_1, \dots, e_k)$, in order to restrict the space $\mathbb{R}^d$ to the subspace $\mathrm{Vect}\{e1, \dots, e_k\}$ with $k$ much smaller than $d$. To avoid the cost of computing these eigenvectors, we instead multiply directly by the covariance matrix, which naturally flattens vectors along the main eigenvector axis.

### *8.3.5 The PCA-jumping method*

We now define the specific method we used based on the principles outlined above. Other variants are clearly possible, and we discuss some of these later.

The matrix $C_k$ is constructed as the covariance matrix of the set of the last $p_k$ iterations of the algorithm: $(\hat{\alpha}^{(k-1)}, \dots, \hat{\alpha}^{(k-p_k)})$ for some well chosen $p_k$. For each iteration, we try three possible values for $C_k$: $C_k^1 = Id$ the identity matrix, $C_k^2 = \mathrm{Cov}(\hat{\alpha}^{(k-1)}, \dots, \hat{\alpha}^{(k-10)})$ the covariance matrix of the 10 last points, and $C_k^3$ the matrix of the 100 last points. We will say that $C_k^1$ corresponds to using *no memory*, $C_k^2$ to a *short memory*, and $C_k^3$ to a *long memory*. When there are insufficient points to fill the memory, we take as many as are available, for example when $k \le 100$ we use $C_k^3 = \mathrm{Cov}(\hat{\alpha}^{(k-1)}, \dots, \hat{\alpha}^{(1)})$.

So that the matrices control the direction but not the size of jumps, at each step $k$ we renormalize to form the steered direction vectors $d_k^i = \frac{C_k^i \Delta_k}{\|C_k^i \Delta_k\|} \|\Delta_k\|$, $i = 1, 2, 3$.

We use the following aggressive algorithm to select the step size $\beta_k^i$ for each $i$:

```
(0) Initialize with $\beta_k^i = 1$;
(1) **If** $F(\hat{\alpha}^{(k)} + \beta_k^i d_k^i) < F(\hat{\alpha}^{(k)} + 2\beta_k^i d_k^i)$ **then** $\beta_k^i \leftarrow 2\beta_k^i$ and repeat **(1)**
    **else** return $\beta_k^i$.
```

In other words, as long as doubling the jump size improves $F$, we double again.

Finally, we combine the step size and direction into three candidates: $\hat{\alpha}^i = \hat{\alpha}^{(k)} + \beta_k^i d_k^i$ for $i = 1, 2, 3$, and set $\hat{\alpha}^{(k+1)}$ to the one giving the highest value of $F$.

It is important to note that despite the opportunistic character of this algorithm, the crucial inequality $F(\hat{\alpha}^{(k)}) < F(\hat{\alpha}^{(k+1)})$ when a maximum is not yet reached is guaranteed, since $F(\hat{\alpha}^{(k)}) < F(\hat{\alpha}^{(k)} + \Delta_k)$ is guaranteed by the definition of $\Delta_k$, and because $F(\hat{\alpha}^{(k)} + \Delta_k) = F(\hat{\alpha}^{(k)} + 1 \times C_k^1 \Delta_k) \le F(\hat{\alpha}^{(k+1)})$.

### *8.3.6 The cost of PCA-jumping*

Figure 8 gives an example of the EM algorithm with no speed-up, with the jumping method alone, and with PCA-jumping. Figure 9 shows the corresponding evolution of the log-likelihood for these trajectories.

The effect of the multiplication by the covariance matrix is that the direction of the basic EM step $\Delta_k$ is steered towards the axis where the previous iterations are

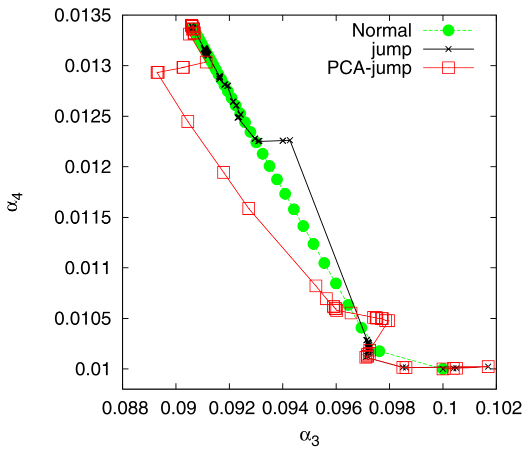


**Fig. 8** Trajectories of the two smallest coordinates of $\hat{\alpha}^{(k)}$, starting from the ground truth $\alpha = (1, 1/3, 0.1, 0.01)$ but with a different data set than that of Fig. 5, using different methods. The "normal" trajectory (1 point per 1000 shown) is the EM algorithm without any speed-up, "jump" trajectory is the jumping method without PCA (choosing always $i = 1$), and "jump+PCA" uses the complete method. The "normal" trajectory has 47073 steps, "jump" has 1215, and "jump+PCA" only 59

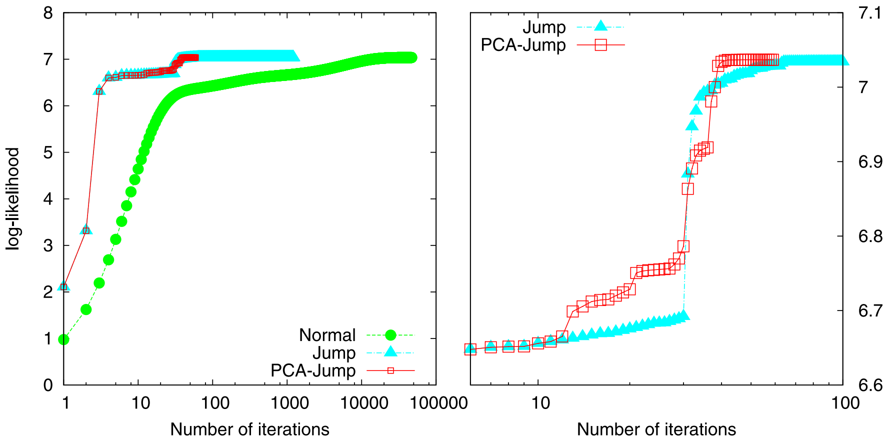


**Fig. 9** Evolution of the log-likelihood as a function of the number of iterations, corresponding to the trajectories of Fig. 8. The *right-side* shows a zoom of the *left-side*

concentrated. Thanks to this, the unwanted oscillation effect of Fig. 7 is avoided, and the size of the jumps (i.e. the values of $\beta_k$) can become much larger.

We used three levels of memory to capture the shape of the trajectory on different 'spatial' scales. We noticed that usually all three alternatives are employed by the algorithm, and that on average the biggest jumps were made with the short memory.

Each level of memory involves computing $F$, however the cost of this is not larger than that of computing the initial $\Delta_k$. In our problem, computing one normal EM step costs more than computing the log-likelihood $|T|$ times, since we need to compute $\frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\hat{\alpha}^{(k)}}(l_j|d(i))$ for each $j \in T$, each $\mathbb{E}_{\hat{\alpha}^{(k)}}(l_j|d(i))$ being more complicated to compute than the likelihood. Therefore a single step of the PCA-jumping algorithm usually costs no more than 2 or 3 times a normal EM step, while the total number of steps is greatly reduced. The computation of the covariance matrices is, again for this problem, very cheap compared to the time needed to compute $F$ or $\Delta_k$.

Clearly, the above strategy has parameters which could be optimized. In particular the number of memory levels, and their durations, could be altered. It would also be possible to use the power $C^n$ of a covariance matrix instead of $C$ to increase the steering effect, or even to use alternative matrices. In [6], PCA (even the eigenvectors) was computed by using a recursive method allowing points to be added successively until the number of principal dimensions ceased dropping. Such a method could be employed here too.

More elaborate methods will always come at increased cost. The advantage of our particular strategy within the PCA-jumping class is its simplicity, since by selecting from only three possible steps, we capture the essence of traditional EM, as well as knowledge of the local and global trajectory shape, and by multiplying by the covariance matrix, we employ PCA without the usual costs of eigenvector evaluation. As for the choice of $\beta_k$, our strategy has the advantage of being both simple and very aggressive, allowing large values of $\beta_k$ to be found quickly and in a single iteration, with no arbitrary upper limit imposed. (We investigated the possibility of selecting $\beta_k$ based on maximizing $F(\hat{\alpha}^{(k)} + \beta_k C_k \Delta_k)$, through a binary search, however, the overhead of the search was not compensated by the gain in jump size.)

#### 8.3.7 *PCA-jumping with sampling initialization: results*

Our final method consists of using a number of steps of the sampling method to get a rough approximation, which is then used to initialize the PCA-jumping method. To control the resources used by the sampling method phase, we set the number of sampling steps to be equivalent computationally to a single step of normal EM. This method was used for most of the experiments presented in Sect. 7.

Figure 10 shows a comparison against the normal EM, using the same ground truth and data as Figs. 6 and 8, starting from a random point. The speed-up due to the method is very significant in this case. Table 9 shows a speed comparison against normal EM for one experiment for each of the 4 main examples of Sect. 7, starting from the ground truth. All simulations were made on the same Intel Core2 Duo 2.40 GHz laptop, but using only one CPU. Under the iterations column, "$100 + x$" means 100 sampling method steps followed by $x$ steps of PCA-jumping. In all cases these 100 steps cost as much as one normal step. The cost of the sampling steps was omitted when computing the average step time for PCA-jumping (last column).

We see that the acceleration method was particularly effective for the trees $U9$ and $B3H2$, where the convergence of the normal algorithm is extremely slow, but still produces a substantial gain for the binary trees $B1H3$ and $B1H4$ where the convergence of the normal EM was however already fairly good. The reason for such a difference is, we believe, the fact that $U9$ and $B3H2$ each contain long branches. The reason might be that we get exactly the same information for all the nodes in one same branch, and it becomes thus much harder to discriminate between them, resulting in slow convergence. This tendency has been also observed on other trees.

We used the ground truth as the initial condition here as we noticed that, in some cases, the normal EM converged towards a local maximum with a likelihood much lower than the one found by the accelerated method. When starting both methods

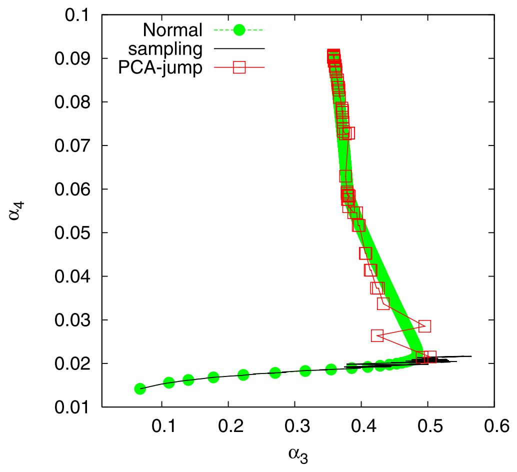


**Fig. 10** Trajectories of the two smallest coordinates of $\hat{\alpha}^{(k)}$, for the ground truth $\alpha = (1, 1/3, 0.1, 0.01)$, starting from a random point. The "normal" trajectory (1 point per 5 shown) is the EM algorithm without any speed-up. The "sampling" trajectory correspond to 100 iterations of the sampling method, (with cost equal to 1 step of the normal EM), and the "jump+PCA" trajectory was obtained by initializing the PCA-jumping method from the final point of the "sampling" trajectory. The "normal" trajectory has 58034 steps, while the "jump+PCA" has only 78. The computing time for the "sampling" and "jump+PCA" trajectories together was 195 times less than for "normal"

**Table 9** Comparison between the normal EM and the sampling + PCA-jumping method for one estimation on the 4 trees in Sect. 7, starting from the ground truth

| Tree | Method | Final Log-L | # Iterations | CPU time | Av. step |
|---|---|---|---|---|---|
| $U9$ | Normal EM | −7.517 | 341845 | 1373 m 24 s | 0.24 s |
| | PCA-jump | −7.517 | 100 + 699 | 6 m 46 s | 0.58 s |
| $B1H3$ | Normal EM | −20.107 | 153 | 1 m 21 s | 0.53 s |
| | PCA-jump | −20.107 | 100 + 14 | 20 s | 1.42 s |
| $B1H4$ | Normal EM | −35.614 | 207 | 62 m 20 s | 18.07 s |
| | PCA-jump | −35.614 | 100 + 28 | 14 m 12 s | 30.4 s |
| $B3H2$ | Normal EM | −10.155 | 122941 | 2853 m 16 s | 1.4 s |
| | PCA-jump | −10.155 | 100 + 429 | 21 m 40 s | 3.0 s |

from the ground truth they converged to the same fixed point, facilitating a direct speed comparison. Table 10 shows the comparison when starting from a random point. For the trees $U9$ and $H3B2$ the normal EM converged quite quickly to the final point (thought still less quickly than the accelerated EM), but more importantly, this point was **not** the MLE as its likelihood was smaller than the fixed point found by the accelerated EM. In all the experiments we performed starting from the same initial conditions, our method converged quicker than the normal EM, and always gave a likelihood as good as the normal method, if not better.

**Table 10** Comparison between the normal EM and the PCA-jumping method for one estimation on the 4 trees in Sect. 7, starting from a random point

| Tree | Method | Final Log-L | # Iterations | CPU time | Av. step |
|---|---|---|---|---|---|
| $U9$ | Normal | $-7.570$ | 3933 | 19 m 26 s | 0.29 s |
| | PCA-jump | $-7.517$ | $100 + 755$ | 6 m 53 s | 0.55 s |
| $B1H3$ | Normal EM | $-20.107$ | 216 | 1 m 57 s | 0.54 s |
| | PCA-jump | $-20.107$ | $100 + 14$ | 20 s | 1.42 s |
| $B1H4$ | Normal EM | $-35.614$ | 315 | 89 m 25 s | 17.03 s |
| | PCA-jump | $-35.614$ | $100 + 26$ | 12 m 44 s | 29.38 s |
| $B3H2$ | Normal EM | $-10.255$ | 2043 | 62 m 10 s | 1.8 s |
| | PCA-jump | $-10.155$ | $100 + 676$ | 41 m 41 s | 3.7 s |

## 9 Conclusions and future work

We have considered a network tomography problem based on a finite number of end-to-end delay measurements made of multicast probes sent over a tree, where each node of the tree imparts an exponentially distributed delay to each passing probe. We showed how its assumptions of spatial independence, and sum-of-exponentials delay marginals, follow naturally from the properties of Kelly networks in the case of rare probing, thereby firmly establishing for the first time a connection between a delay tomography problem and an inverse queueing problem over a network with non-trivial topology.

The problem was formulated as the search for a maximum likelihood estimator for the parameter, being the vector of mean delays for each node in the tree, which due to its complexity was solved using the EM algorithm. We showed how the E- and M-steps could be solved explicitly and combined, reducing the problem to the evaluation of a set of conditional probabilities of internal node states, given the observed delays. We provided two solution methods for these with formal proofs, one based on a recursion beginning from the root node, the other an explicit expression (though with some recursive components). The latter has far fewer terms and is amenable to efficient implementation, and it was used to provide solutions for a number of examples.

The EM algorithm is notoriously slow to converge, and moreover since the combinatorics of the tree make each step very expensive, only trivial trees can be solved in practice without acceleration techniques. We developed a new technique, *PCA-jumping with Sampling Initialization*, which provided a speed-up of between one and three orders of magnitude for our problem. Its novel features include the use of Principal Component Analysis (yet without the need to calculate eigenvectors) to efficiently mine local and global information about the EM trajectory in order to control jump direction, and an efficient and aggressive geometric rule for the size of jumps which allows large steps to be made when profitable, as is the case for our problem. Initialization is performed using a 'Sampling' method which has very low and bounded cost, yet is capable of finding a starting point from which PCA-jumping can be effective. The speed of the method is compared to standard EM in a variety of examples, and was also shown to provide better estimates in some cases.

The main directions for future work lie in a more formal analysis of the acceleration technique, its optimization with respect to a number of parameters, generalizations, and comparison against alternatives. Of particular interest is to understand to what extent the 'valley' phenomenon which inspired the technique holds for other problems, in particular non-linear ones where fixed points have small basins of attraction.

**Acknowledgements** This work was performed during the internship of F. Pin at CUBIN in Melbourne. This research was supported under Australian Research Council's *Discovery Projects* funding scheme (project number DP0985673). The authors wish also to thank J.P. Laumond and S. Dalibard for their very fast answers, and Earl Lawrence and George Michailidis for their kind answers and the access to their code for least squares estimation.

## Appendix

We recall here that we use the notation simplification from Sect. 6. In particular, we use $\alpha$ in place of $\hat{\alpha}^{(k)}$ (as a single iteration is considered), and we set $\gamma = (\gamma_j)_{j\in T} = (1/\alpha_j)_{j\in T}$.

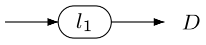
### A.1 Proof of the density formula

We will now prove that the description we gave for the expanded expression of $q_\alpha(\mathcal{T}, d)$ is correct. For this, we will show that the following equality holds:

$$q_\alpha(\mathcal{T}, d) = \Gamma_T \sum_{C \text{ cut of } \mathcal{T}} h_\alpha(\mathcal{T}, d, C), \tag{16}$$

for $q_\alpha(\mathcal{T}, d)$ defined by its integral expression (9) and for the right-side terms as defined previously in Sect. 6.5.2. For this we will have an inductive reasoning over the tree $\mathcal{T}$, and we will use the inductive relation of Theorem 6.1.

*Initialization of the induction*

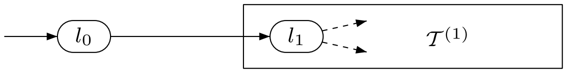


It is obvious that the formula holds for a single-node tree. In this case, we have $q_\alpha(d) = \gamma_1 e^{-\gamma_1 d}$, and it is easy to verify that it correspond to the formula given previously, since there is only one cut $C = \{1\}$ with no past nor future. Therefore, we have: $q_\alpha(d) = \gamma_1 r(C)s(C)t(C)$ with $r(C) = 1$, $t(C) = 1$, and $s(C) = e^{-\gamma_1 d_1}$, which proves the initial step of the induction.

*Induction step* We consider a general tree, assume by induction that the formula is true for each subtree of the root, and then prove that the formula is also true for the whole tree.

For this we will have to make a distinction between two cases: if the root has only one child and if it has at least two children. This distinction will explain why we introduced the notion of branch, since in the one-child case, some terms from the same branch can be combined.

*(a) Root with only one child*



We have $T^{(1)} = T \setminus \{0\}$, $V^{(1)} = V$, and $d^{(1)} = d$ and the inductive formula (10) becomes:

$$q_\alpha(\mathcal{T}, d) = \int_{l_0=0}^{d_0} \gamma_0 e^{-\gamma_0 l_0} q_\alpha\big(\mathcal{T}^{(1)}, d - (l_0)\big)\, dl_0.$$

By induction, we suppose the formula true for the subtree $\mathcal{T}^{(1)}$, therefore:

$$q_\alpha\big(\mathcal{T}^{(1)}, d - (l_0)\big) = \Gamma_{\mathcal{T}^{(1)}} \sum_{C \text{ cut of } \mathcal{T}^{(1)}} h_\alpha\big(\mathcal{T}^{(1)}, d - (l_0), C\big),$$

and since $\Gamma_T = \gamma_0 \Gamma_{\mathcal{T}^{(1)}}$:

$$q_\alpha(\mathcal{T}, d) = \Gamma_T \sum_{C \text{ cut of } \mathcal{T}^{(1)}} \int_{l_0=0}^{d_0} e^{-\gamma_0 l_0} h_\alpha\big(\mathcal{T}^{(1)}, d - (l_0), C\big)\, dl_0. \tag{17}$$

Let us consider a cut $C$ of $\mathcal{T}^{(1)}$. We have:

$$h_\alpha\big(\mathcal{T}^{(1)}, d - (l_0), C\big) = r\big(\mathcal{T}^{(1)}, d - (l_0), C\big) s\big(\mathcal{T}^{(1)}, d - (l_0), C\big) t\big(\mathcal{T}^{(1)}, d - (l_0), C\big).$$

*Remark A.1* For more convenient notations, we will denote $h_\alpha^{(1)}(C) := h_\alpha(\mathcal{T}^{(1)}, d - (l_0), C)$ and for $r, s, t$ as well, such that the last relation becomes $h_\alpha^{(1)}(C) = r^{(1)}(C)s^{(1)}(C)t^{(1)}(C)$.

We now want to compute this integral: $\int_{l_0=0}^{d_0} e^{-\gamma_0 l_0} r^{(1)}(C)s^{(1)}(C)t^{(1)}(C)\, dl_0$. As we can see in their definition, the variable $l_0$ does not appear in the terms $r^{(1)}(C)$ and $t^{(1)}(C)$. This fact is obvious for $r$, but the recursive nature of $t$ make it a little more difficult to see. But looking back at the definition, we notice that the recursion is made with "$d - (d_j) := (d_k - d_j)_{k \in F_j}$". Since here our whole formula is applied to $d' = d - (l_0)$, we see that the term $l_0$ is annihilated in the expression of $d' - (d'_j) = ((d_k - l_0) - (d_j - l_0))_{k \in F_j}$. And therefore, $l_0$ does not appear in $t^{(1)}(C)$.

Thanks to this, we have $r^{(1)}(C) = r(\mathcal{T}^{(1)}, d, C)$ and $t^{(1)}(C) = t(\mathcal{T}^{(1)}, d, C)$ and we can take $r^{(1)}(C)$ and $t^{(1)}(C)$ out of the integral, leaving us with this integral to expand:

$$\begin{aligned}&\int_{l_0=0}^{d_0} e^{-\gamma_0 l_0} s^{(1)}(C)\, dl_0\\ &\quad = \int_{l_0=0}^{d_0} e^{-\gamma_0 l_0} \prod_{j \in C} \frac{e^{-\gamma_j (d_j - l_0)}}{\prod_{\substack{k \sim j \\ k \neq j}} (\gamma_k - \gamma_j)}\, dl_0\end{aligned}$$

$$
\begin{aligned}
&= \left[ \frac{1}{(\sum_{j\in C}\gamma_j) - \gamma_0} \prod_{j\in C} e^{-\gamma_0 l_0} \frac{e^{-\gamma_j(d_j - l_0)}}{\prod_{\substack{k\sim j\\ k\neq j}} (\gamma_k - \gamma_j)} \right]_{l_0=0}^{d_0} \\
&= \frac{1}{(\sum_{j\in C}\gamma_j) - \gamma_0} \prod_{j\in C} e^{-\gamma_0 d_0} \frac{e^{-\gamma_j(d_j - d_0)}}{\prod_{\substack{k\sim j\\ k\neq j}} (\gamma_k - \gamma_j)} \\
&\quad + \frac{1}{\gamma_0 - \sum_{j\in C}\gamma_j} \prod_{j\in C} \frac{e^{-\gamma_j d_j}}{\prod_{\substack{k\sim j\\ k\neq j}} (\gamma_k - \gamma_j)} \\
&= e^{-\gamma_0 d_0} \frac{s(\mathcal{T}^{(1)}, d - (d_0), C)}{(\sum_{j\in C}\gamma_j) - \gamma_0} + \frac{s(\mathcal{T}^{(1)}, d, C)}{\gamma_0 - \sum_{j\in C}\gamma_j} \\
&= e^{-\gamma_0 d_0} K(C) + \frac{s(\mathcal{T}^{(1)}, d, C)}{\gamma_0 - \sum_{j\in C}\gamma_j}.
\end{aligned}
$$

As we will see, the exact value of $K(C)$ is in fact not really important for the proof. Once here, the hardest part remains: we have to put all these terms together to prove that the formula is true for the whole tree $\mathcal{T}$.

Putting the last equation back in (17), we get

$$
q_\alpha(\mathcal{T}, d) = \Gamma_T \left[ e^{-\gamma_0 d_0} K' + \sum_{C \text{ cut of } \mathcal{T}^{(1)}} \frac{r^{(1)}(C) s(\mathcal{T}^{(1)}, d, C)}{\gamma_0 - \sum_{j\in C}\gamma_j} t^{(1)}(C) \right], \tag{18}
$$

where $K' = \sum_{C \text{ cut of } \mathcal{T}^{(1)}} r^{(1)}(C) t^{(1)}(C) K(C)$.

As we noticed already, we have $r^{(1)}(C) = r(\mathcal{T}^{(1)}, d, C)$ and $t^{(1)}(C) = t(\mathcal{T}^{(1)}, d, C)$. But, since the term $t(\mathcal{T}^{(1)}, d, C)$ is a term depending only from the cut $C$ and its future in the tree $\mathcal{T}^{(i)}$, which are the same in the tree $\mathcal{T}$, we have $t(\mathcal{T}^{(1)}, d, C) = t(\mathcal{T}, d, C)$.

We will now prove that:

$$
\frac{r(\mathcal{T}^{(1)}, d, C) s(\mathcal{T}^{(1)}, d, C)}{\gamma_0 - \sum_{j\in C}\gamma_j} = r(\mathcal{T}, d, C) s(\mathcal{T}, d, C).
$$

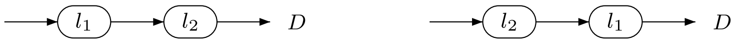
It is relatively easy to see by looking at the definitions of $r$ and $s$, but two cases must be distinguished: when $C$ is a singleton, like $C = \{1\}$ here, or when $C$ contains 2 nodes or more. Indeed, if $C$ is a singleton, say $C = \{i\}$, then $i$ belongs to the present of 0 and 1 (note that here $0 \sim 1$), and we have $s(\mathcal{T}, d, C) = \frac{s(\mathcal{T}^{(1)}, d, C)}{\gamma_0 - \gamma_i}$ and $r(\mathcal{T}, d, C) = r(\mathcal{T}^{(1)}, d, C) = 1$ since the past of $C$ is empty in this case. In the other case where $C$ has 2 nodes or more, then we see that 0 and 1 belong to the past of $C$ and we have $s(\mathcal{T}, d, C) = s(\mathcal{T}^{(1)}, d, C)$ and $r(\mathcal{T}, d, C) = \frac{r(\mathcal{T}^{(1)}, d, C)}{\gamma_0 - \sum_{j\in C}\gamma_j}$. In both cases, the identity is then proved.

This results in:

$$q_\alpha(\mathcal{T},d) = \Gamma_T\left[e^{-\gamma_0 d_0}K' + \sum_{\substack{C \text{ cut of } \mathcal{T}\\ C\neq\{0\}}} h_\alpha(\mathcal{T},d,C)\right], \tag{19}$$

since we notice that all the cuts of $\mathcal{T}^{(1)}$ plus the cut $\{0\}$ forms exactly all the cuts of $\mathcal{T}$.

Finally, in order to show that the formula is true for the tree $\mathcal{T}$, all we have to do left is to show that the term $e^{-\gamma_0 d_0}K'$ is equal to $h_\alpha(\mathcal{T},d,\{0\})$. If we look back at the expression of $K' = \sum_{C \text{ cut of } \mathcal{T}^{(1)}} r^{(1)}(C)t^{(1)}(C)K(C)$, it seems difficult to show directly that this sum of terms combine into just one and is equal to $h_\alpha(\mathcal{T},d,\{0\})$. Luckily, we will not have to do this, since a simple argument of symmetry will suffice. All we have to do is to notice that the function $q_\alpha(\mathcal{T},d)$ stays the same if we exchange the nodes 0 and 1, i.e. if we exchange the values of $\gamma_0$ and $\gamma_1$. This becomes obvious if we take a look at the two station case, since $l_1+l_2=l_2+l_1$, the two following trees are equivalent:



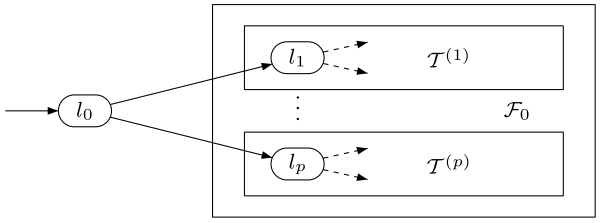
More generally, no permutation inside a branch of $\mathcal{T}$ will change $q_\alpha(\mathcal{T},d)$. Thanks to this, we see that by exchanging the nodes 0 and 1, we get from (19):

$$q_\alpha(\mathcal{T},d) = \Gamma_T\left[e^{-\gamma_1 d_1}K'' + \sum_{\substack{C \text{ cut of } \mathcal{T}\\ C\neq\{1\}}} h_\alpha(\mathcal{T},d,C)\right], \tag{20}$$

and by combining (19) and (20), we get $e^{-\gamma_0 d_0}K' + h_\alpha(\mathcal{T},d,\{1\}) = e^{-\gamma_1 d_1}K'' + h_\alpha(\mathcal{T},d,\{0\})$, and we can finally identify the two terms $e^{-\gamma_0 d_0}K'$ and $h_\alpha(\mathcal{T},d,\{0\})$ since the term $h_\alpha(\mathcal{T},d,\{0\})$ contains an exponential function of the form $e^{-\gamma_0 d_0}$ and $h_\alpha(\mathcal{T},d,\{1\})$ does not.

Finally, we proved that in this first case the formula (16) is also true for the tree $\mathcal{T}$.

*(b) Root with two children or more*



The idea here is roughly the same as in the previous case, but the symmetry argument is no longer required, since here the new terms in $e^{-\gamma_0 d_0}$ will not combine as they did earlier.

By induction, we suppose the formula true for all the subtrees $\mathcal{T}^{(1)}, \dots, \mathcal{T}^{(p)}$. We therefore have for all $i \in \{1, \dots, p\}$:

$$q_\alpha\big(\mathcal{T}^{(i)}, d - (l_0)\big) = \Gamma_{\mathcal{T}^{(i)}} \sum_{C \text{ cut of } \mathcal{T}^{(i)}} h_\alpha\big(\mathcal{T}^{(i)}, d^{(i)} - (l_0), C\big).$$

In this case, the inductive formula (10) gives

$$q_\alpha(\mathcal{T}, d) = \int_{l_0=0}^{d_0} \gamma_0 e^{-\gamma_0 l_0} \left[\prod_{i=1}^{p} q_\alpha\big(\mathcal{T}^{(i)}, d^{(i)} - (l_0)\big)\right] dl_0.$$

We already have that $\gamma_0 \prod_{i=1}^{p} \Gamma_{T^{(i)}} = \Gamma_T$. Therefore we get:

$$q_\alpha(\mathcal{T}, d) = \Gamma_T \int_{l_0=0}^{d_0} e^{-\gamma_0 l_0} \left[\prod_{i=1}^{p} \sum_{C \text{ cut of } \mathcal{T}^{(i)}} h_\alpha\big(\mathcal{T}^{(i)}, d^{(i)} - (l_0), C\big)\right] dl_0.$$

The product of sums can be easily expanded by noticing that each $p$-tuple $(C_1, \dots, C_p)$ of cuts of the trees $\mathcal{T}^{(1)}, \dots, \mathcal{T}^{(p)}$ forms exactly a partition of the cut $C = \bigcup_{i=1}^{p} C_i$ of the forest $\mathcal{F}_0 = (\mathcal{T}^{(1)}, \dots, \mathcal{T}^{(p)})$. We therefore have a bijection between the cuts $C$ of $\mathcal{F}_0$ and the $p$-tuple of cuts $(C_1, \dots, C_p)$ of $\mathcal{T}^{(1)}, \dots, \mathcal{T}^{(p)}$. We can then write that:

$$\begin{aligned} q_\alpha(\mathcal{T}, d) &= \Gamma_T \int_{l_0=0}^{d_0} e^{-\gamma_0 l_0} \left[\sum_{C \text{ cut of } \mathcal{F}_0} \prod_{i=1}^{p} h_\alpha\big(\mathcal{T}^{(i)}, d^{(i)} - (l_0), C_i\big)\right] dl_0 \\ &= \Gamma_T \sum_{C \text{ cut of } \mathcal{F}_0} \int_{l_0=0}^{d_0} e^{-\gamma_0 l_0} \prod_{i=1}^{p} h_\alpha\big(\mathcal{T}^{(i)}, d^{(i)} - (l_0), C_i\big)\, dl_0. \end{aligned}$$

Let us then consider a cut $C$ of $\mathcal{F}_0$. As we did in the first case, we introduce the lighter notation $h_\alpha^{(i)}(C_i) = h_\alpha(\mathcal{T}^{(i)}, d^{(i)} - (l_0), C_i)$ and for $r, s$ and $t$ as well, such that $h_\alpha^{(i)}(C_i) = r^{(i)}(C_i) s^{(i)}(C_i) t^{(i)}(C_i)$. Here again, we notice that the variable $l_0$ does not appear in $r^{(i)}(C_i)$ nor $t^{(i)}(C_i)$. We can then take these two terms out of the integral:

$$\int_{l_0=0}^{d_0} e^{-\gamma_0 l_0} \prod_{i=1}^{p} h_\alpha^{(i)}(C_i)\, dl_0 = \prod_{i=1}^{p} r^{(i)}(C_i) t^{(i)}(C_i) \int_{l_0=0}^{d_0} e^{-\gamma_0 l_0} \prod_{i=1}^{p} s^{(i)}(C_i)\, dl_0.$$

By looking back at the definition of $t$, and since $l_0$ does not appear in $t^{(i)}(C_i)$, we have $t^{(i)}(C_i) = t(\mathcal{T}^{(i)}, d^{(i)} - (l_0), C_i) = t(\mathcal{T}^{(i)}, d^{(i)}, C_i)$, and therefore:

$$\begin{aligned} \prod_{i=1}^{p} t^{(i)}(C_i) &= \prod_{i=1}^{p} \prod_{j \in C_i} t_j\big(\mathcal{T}^{(i)}, d^{(i)}, C_i\big) = \prod_{i=1}^{p} \prod_{j \in C_i} t_j(\mathcal{F}_0, d, C) = \prod_{j \in C} t_j(\mathcal{F}_0, d, C) \\ &= t(\mathcal{F}_0, d, C). \end{aligned}$$

The fact that $t_j(\mathcal{T}^{(i)}, d^{(i)}, C_i) = t_j(\mathcal{F}_0, d, C)$ comes from the fact that $t_j(\mathcal{F}_0, d, C)$ depends only on the node $j$ and its future.

By looking at the definition of $r$, we also deduce that:

$$\prod_{i=1}^{p} r^{(i)}(C_i) = \prod_{i=1}^{p} \prod_{j \in \mathbf{Past}(\mathcal{T}^{(i)}, C_i)} \frac{1}{\gamma_j - \sum_{\substack{k \in C_i \\ j \ll k}} \gamma_k} = \prod_{j \in \mathbf{Past}(\mathcal{F}_0, C)} \frac{1}{\gamma_j - \sum_{\substack{k \in C \\ j \ll k}} \gamma_k}$$
$$= r(\mathcal{F}_0, d, C),$$

where we recall that $\mathbf{Past}(\mathcal{T}^{(i)}, C_i)$ is the past of the cut $C_i$ in the tree $T^{(i)}$, and $\mathbf{Past}(\mathcal{F}_0, C)$ is the past of the cut $C$ in the sub-forest $\mathcal{F}_0$. We especially pay attention to the fact that indeed $\{k \in C_i \mid j \ll k\} = \{k \in C \mid j \ll k\}$.

We then focus on the integral:

$$\int_{l_0=0}^{d_0} e^{-\gamma_0 l_0} \prod_{i=1}^{p} s^{(i)}(C_i)\, dl_0 = \int_{l_0=0}^{d_0} e^{-\gamma_0 l_0} \prod_{i=1}^{p} \prod_{j \in C_i} \frac{e^{-\gamma_j(d_j - l_0)}}{\prod_{\substack{k \sim j \\ k \neq j}} (\gamma_k - \gamma_j)}\, dl_0$$
$$= \int_{l_0=0}^{d_0} e^{-\gamma_0 l_0} \prod_{j \in C} \frac{e^{-\gamma_j(d_j - l_0)}}{\prod_{\substack{k \sim j \\ k \neq j}} (\gamma_k - \gamma_j)}\, dl_0$$
$$= e^{-\gamma_0 d_0} \frac{s(\mathcal{F}_0, d - (d_0), C)}{(\sum_{j \in C} \gamma_j) - \gamma_0} + \frac{s(\mathcal{F}_0, d, C)}{\gamma_0 - \sum_{j \in C} \gamma_j},$$

the last equality having been already seen in the first case. Putting all back together we get

$$q_\alpha(\mathcal{T}, d) = \Gamma_T \sum_{\substack{C \text{ cut of } \mathcal{T} \\ C \neq \{0\}}} \Bigg[ e^{-\gamma_0 d_0} \frac{h_\alpha(\mathcal{F}_0, d - (d_0), C)}{(\sum_{j \in C} \gamma_j) - \gamma_0}$$
$$+ \frac{r(\mathcal{F}_0, d, C) s(\mathcal{F}_0, d, C) t(\mathcal{F}_0, d, C)}{\gamma_0 - \sum_{j \in C} \gamma_j} \Bigg]$$

since the cuts of $\mathcal{T}$ are the cuts of $\mathcal{F}_0$ plus the cut $\{0\}$, and because we already noticed in the first case that $r(\mathcal{F}_0, d, C) = r(\mathcal{F}_0, d - (d_0), C)$ and $t(\mathcal{F}_0, d, C) = t(\mathcal{F}_0, d - (d_0), C)$. We then notice that for any cut $C \neq \{0\}$, we have $\frac{r(\mathcal{F}_0, d, C)}{\gamma_0 - \sum_{j \in C} \gamma_j} = r(\mathcal{T}, d, C)$, $s(\mathcal{F}_0, d, C) = s(\mathcal{T}, d, C)$, and $t(\mathcal{F}_0, d, C) = t(\mathcal{T}, d, C)$. Therefore:

$$q_\alpha(\mathcal{T}, d) = \Gamma_T \Bigg[ e^{-\gamma_0 d_0} \sum_{C \text{ cut of } \mathcal{F}_0} \frac{h_\alpha(\mathcal{F}_0, d - (d_0), C)}{(\sum_{j \in C} \gamma_j) - \gamma_0} + \sum_{\substack{C \text{ cut of } \mathcal{T} \\ C \neq \{0\}}} h_\alpha(\mathcal{T}, d, C) \Bigg].$$

Finally, since $s(\mathcal{T}, d, \{0\}) = e^{-\gamma_0 d_0}$ and $r(\mathcal{T}, d, \{0\}) = 1$, we see that:

$$e^{-\gamma_0 d_0} \sum_{C \text{ cut of } \mathcal{F}_0} \frac{h_\alpha(\mathcal{F}_0, d - (d_0), C)}{(\sum_{j \in C} \gamma_j) - \gamma_0} = e^{-\gamma_0 d_0} t(\mathcal{T}, d, \{0\}) = h_\alpha(\mathcal{T}, d, \{0\}),$$

and we finally prove that the formula (16) holds also for the tree $\mathcal{T}$.

To conclude, the induction being now proved in both cases, we conclude that our formula is true for any tree $\mathcal{T}$. The proof of the expression of $\xi_\alpha(l|d)$ follows a similar but slightly more complicated reasoning and is omitted for space reasons.

## References

1. Baccelli, F., Kauffmann, B., Veitch, D.: Inverse problems in queueing theory and internet probing. Queueing Syst. **63**(1–4), 59–107 (2009)
2. Chen, A., Cao, J., Bu, T.: Network tomography: identifiability and Fourier domain estimation. In: IEEE INFOCOM, pp. 1875–1883 (2007)
3. Chrétien, S., Hero, A.O.: Kullback proximal algorithms for maximum likelihood estimation. Inria report 3756 (2009)
4. Coates, M., Nowak, R.: Network tomography for internal delay estimation. In: Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP), Salt Lake City, Utah (2001)
5. Cramér, H.: Mathematical Methods of Statistics. Princeton University Press, Princeton (1946)
6. Dalibard, S., Laumond, J.P.: Control of probabilistic diffusion in motion planning. In: 8th International Workshop on the Algorithmic Foundations of Robotics (WAFR 2008) (2008)
7. Duffield, N., Presti, F.L.: Multicast inference of packet delay variance at interior network links. In: IEEE INFOCOM, Tel Aviv, Israel, pp. 1351–1360 (2000)
8. Huang, H.S., Yang, B.H., Hsu, C.N.: Triple jump acceleration for the EM algorithm. In: IEEE International Conference on Data Mining (ICDM'05) (2005)
9. Jamshidian, M., Jennrich, R.I.: Conjugate gradient acceleration of the EM algorithm. J. Am. Stat. Assoc. (1993)
10. Kauffmann, B., Baccelli, F., Veitch, D.: Towards multihop available bandwidth estimation—inverse problems in queueing networks. In: Proc. ACM Sigmetrics/Performance'09, Seattle, WA, USA (2009)
11. Kelly, F.: Reversibility and Stochastic Networks. Wiley, New York (1979)
12. Lawrence, E., Michailidis, G., Nair, V.N.: Network delay tomography using flexicast experiments. J. R. Stat. Soc., Ser. B **68**, 785–813 (2006)
13. Lawrence, E., Michailidis, G., Nair, V.N.: Statistical inverse problems in active network tomography. In: Complex Datasets and Inverse Problems: Tomography, Networks and Beyond. IMS Lecture Notes-Monograph Series, vol. 54, pp. 24–44. IMS, Beachwood (2007). doi:10.1214/074921707000000049
14. Liang, G., Yu, B.: Maximum pseudo likelihood estimation in network tomography. IEEE Trans. Signal Process. **51**(8), 2043–2053 (2003) (Special Issue on Data Networks)
15. McLachlan, G.J., Krishnan, T.: The EM Algorithm and Extensions, 2nd edn. Wiley Series in Probability and Statistics. Wiley, New York (2008)
16. Presti, F.L., Duffield, N.G., Horowitz, J., Towsley, D.: Multicast-based inference of network-internal delay distributions. IEEE/ACM Trans. Netw. **10**(6), 761–775 (2002)
17. Salakhutdinov, R., Roweis, S.: Adaptive overrelaxed bound optimization methods. In: International Conference on Machine Learning (ICML-2003) (2003)
18. Shih, M.F., Hero, A.O.: Unicast-based inference of network link delay distributions with finite mixture models. IEEE Trans. Signal Process. **51**(8), 2219–2228 (2003) (Special Issue on Data Networks)
19. Tsang, Y., Coates, M., Nowak, R.: Network delay tomography. IEEE Trans. Signal Process. **51**(8), 2125–2136 (2003) (Special Issue on Data Networks)
20. Wu, C.J.: On the convergence properties of the EM algorithm. Ann. Stat. **11**(1), 95–103 (1983)